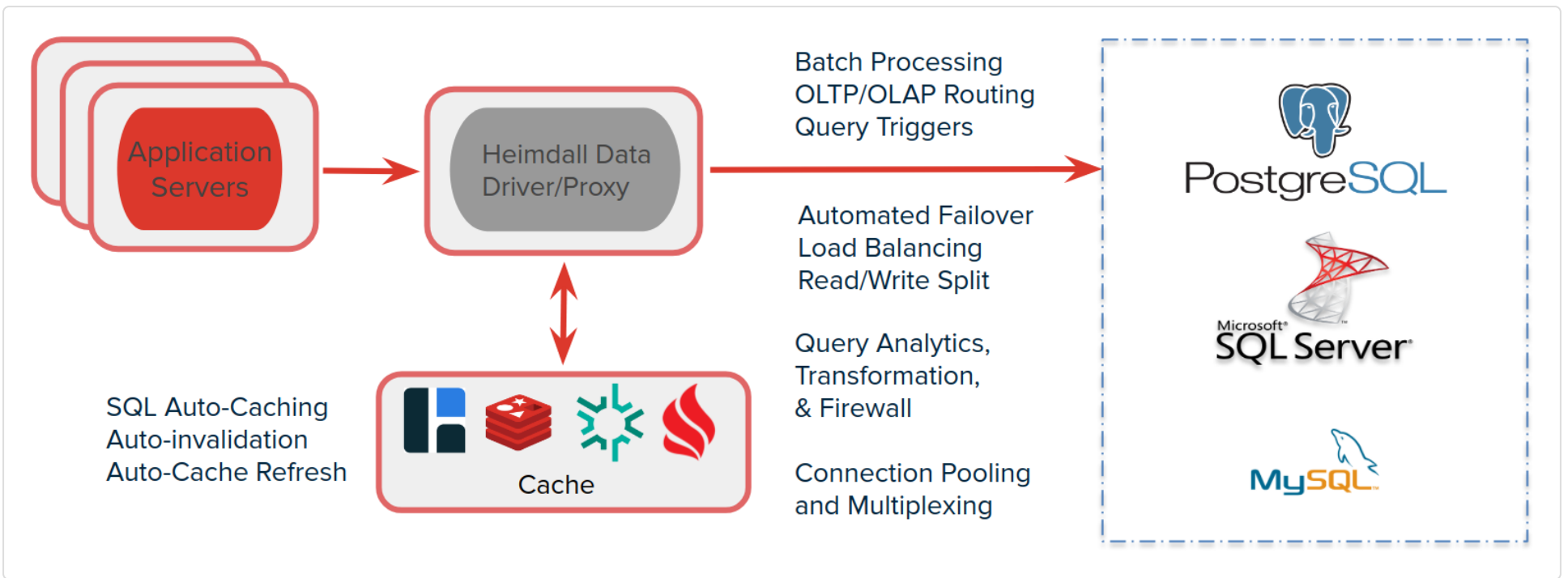
Introduction

The Heimdall Data proxy is a software platform for application and infrastructure owners. Whether on-premise or cloud, Heimdall helps organizations deliver faster, more reliable, and secure content generation. Heimdall Data is a distributed Database Proxy that improves database performance from an application perspective. We give users visibility and control over their SQL environment.

Use cases for the software

The key use cases for Heimdall database proxy include:

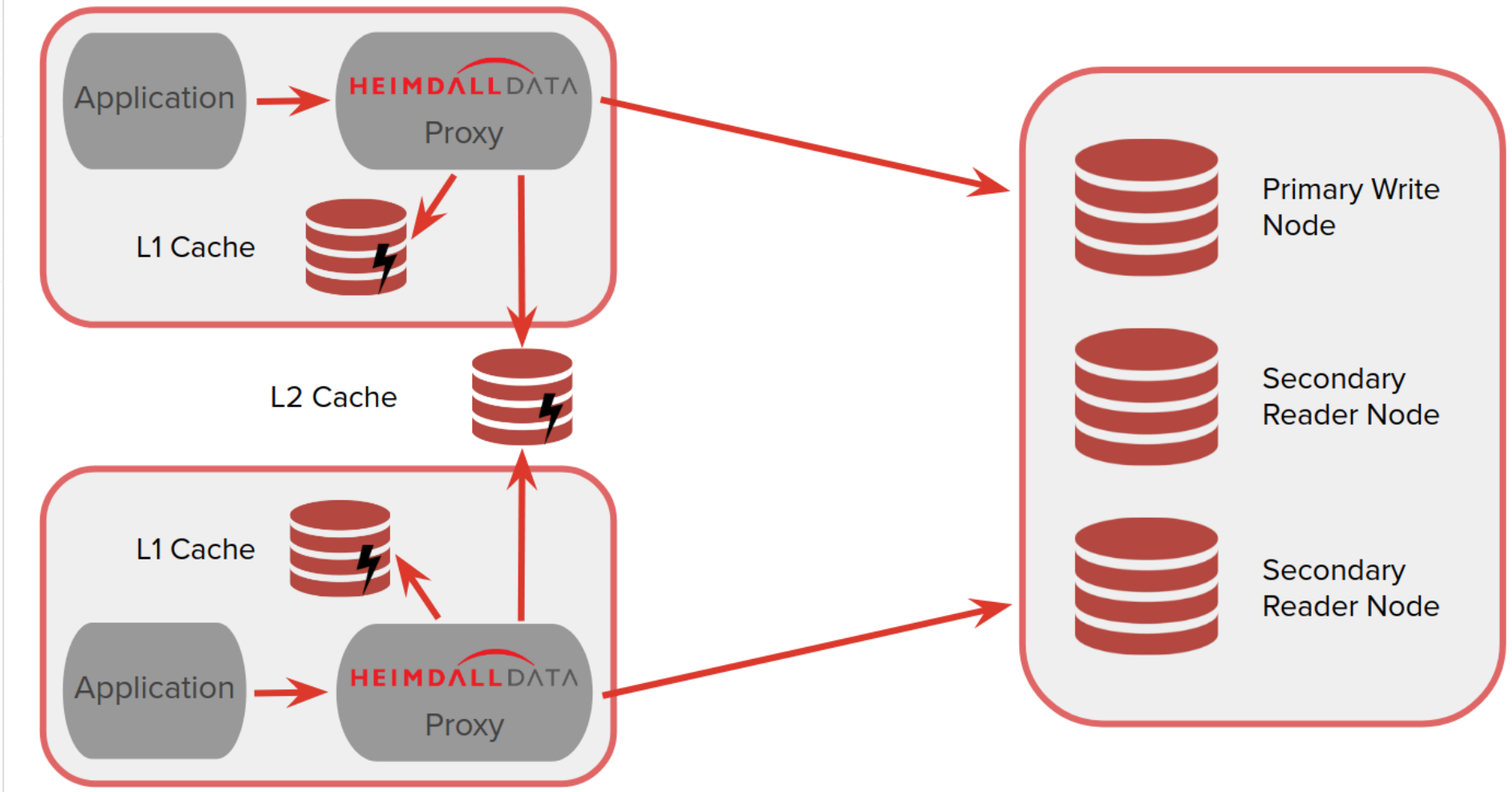
- Connection pooling from serverless applications, including support for multi-user pooling
- Automated caching and invalidation to the cache of your choice (e.g. Redis)
- Read/Write split (i.e. query routing) to reduce load on write-masters in a multi-node cluster
- Improving the performance of DML Ingestion, in particular in Analytics Database environments (Pivotal Greenplum and AWS Redshift)
- Analytics



Heimdall supports a variety of databases including Postgres & compatible (Amazon RDS, Aurora and Redshift; VMWare Greenplum, Google Cloud SQL & AlloyDB), MySQL and SQL Server compatible databases. For caching, it supports Hazelcast and Redis.

Caching Architecture

Heimdall optimally operates via a two-layer cache system. The first layer (L1) is an in-heap cache, the size of which is controlled by the heap size in the VDB->proxy settings. The second layer (L2) is used as secondary storage and as a means of communication amongst the deployed proxies. The user will choose any of the supported cache engines to provide a distributed cache between nodes.



Performance Impact

In implementing Heimdall, one frequent question is "what is the performance impact"? This is a difficult question to answer as the variables can be quite complex including:

- Size of requests, i.e. multi-page queries vs. simple single table queries;
- Number of requests/second;
- Size of results and the number of rows;
- If logging is enabled;
- Cache hit rates--the higher the better;
- Number of cores available for processing;
- Number of connections in use at one time;
- Deployment model (centralized vs. distributed), see below;

In a highly optimized scenario with 6 physical cores (12 hyperthreading), with 100% cache hit rate and simple results, Heimdall can achieve over 250k queries/second, WITH the test client (jmeter) on the same host consuming cpu time as well. Heimdall is designed to operate distributed however, so can reasonably scale to nearly any performance level. In most environments, it is suggested that up to 8 cores be used, and that additional scaling based on CPU load be done horizontally, adding additional nodes. In an AWS environment, this can be done in auto-scaling groups to account for surges very easily.

Please note: while jmeter CAN be used to benchmark Heimdall, it is not the proper tool to directly connect to the database to test and validate caching and read/write split. Customers should use their own applications to drive traffic in a lower environment (test). In such a setup, jmeter can be used to generate application load that THEN triggers database traffic through Heimdall. Using applications such as JMeter, DBeaver, or other similar tools will not reflect accurately how Heimdall will behave with other applications so should be avoided for this purpose as a general rule.

Prerequisites and Requirements

In order to install and use Heimdall, the following will be needed:

- A login for the database. For full capabilities, this user should have the ability to login, execute a health check query, access (or create) a schema/database of "heimdall", and have full access to this schema or database. This is used for health checking and various other features such as lag detection.
- Understanding of the configuration of the application, and how to adjust what database it points to. This varies from application to application, but a few common locations for this are:
    - The IIS configuration for a .net application, under connect strings

        - a configuration json file
        - a configuration PHP file
    - Preferably a test environment to test with before going into production.
    - A Linux server or Docker environment (see below for deployment options) with at least 20GB of space and 4GB of RAM

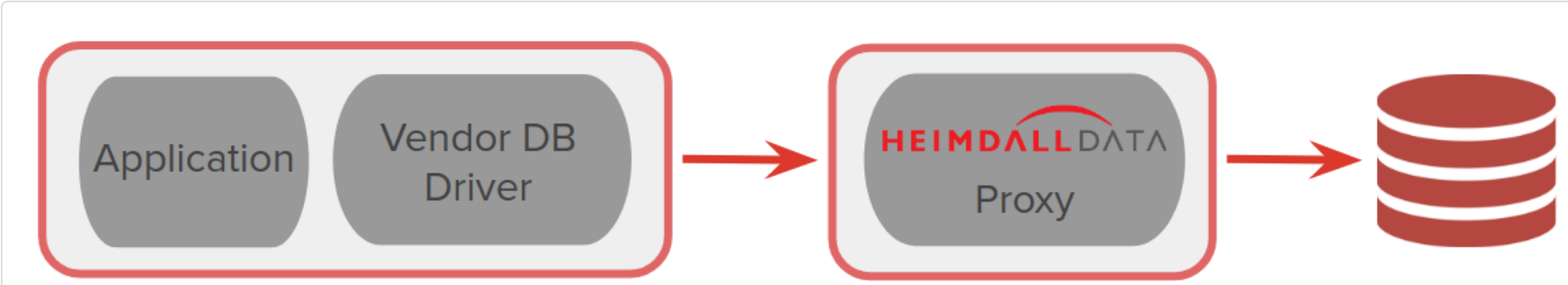
Supported versions of Linux include vendor supported versions of (in alphabetical order):

- Alpine
- Amazon Linux 2+
- Centos
- Debian
- Oracle
- Redhat/Fedora
- Suse
- Ubuntu

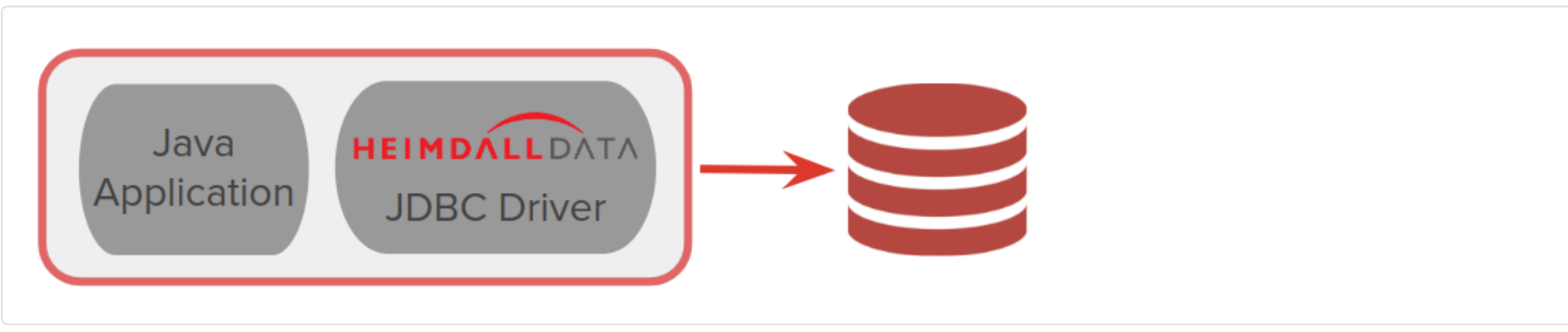
As Heimdall is a Java based program with no cpu architecture dependent code, it can be installed on any architecture supported by Linux and Java. In the case of AWS, the Graviton 2 & 3 instances provides a roughly 30-50% better price/performance ratio vs. the standard Intel VMs in our testing, and should be the first choice in deploying out an auto-scaling cluster. No special instructions are needed to use Heimdall on non-Intel architecture systems.

Deployment Models

Heimdall can be installed as a proxy:



Alternatively, it can be installed as a JDBC Driver (only for Java Applications) and can support other JDBC data sources as well:



For most customers, it is advisable to start with a simple proxy deployment on a single node, and once tested, to work with Heimdall to support more advanced configurations.

In proxy mode, Heimdall supports auto-scaling, multi-zone configurations. It is recommended that in AWS, the AWS CloudFormation Template should be used, which prompts for the various configuration options in order to deploy such a configuration. For other environments, please contact Heimdall support for guidance on how to configure such a configuration.

Deployment Methods

In order to install Heimdall, one of several methods can be used:

- Leverage our single command installer into a supported version of Linux as a single server (documentation)

- Purchase a VM with the software installed from a cloud marketplace (AWS, Azure or GCP)
- Use the docker file to create a docker image (documentation)
- Install into the application as a JDBC driver (please contact Heimdall support for assistance).

For a detailed breakdown on each of the install methods, please see the detailed install page.

One note of importance--all the install methods except a pre-build VM will by default pull our newest version of code from our S3 file distribution site at the time of install except for the manual install option. **When installing, make sure that security groups or ACLs do not block this out-bound request**. Once installed and online, the system can be locked down if desired see security.

Additionally, the deployment model will impact pricing. In the cloud marketplace images, purchase is based on the size of instance and time used, although many offer a free trial for testing or a free trial can be requested. In on-premises installs, pricing will be direct from Heimdall, and negotiated based on the number of instances used and volume, along with the support model needed.

In most cases, an initial configuration can be made within 30 minutes, and tested using tools such as DBeaver, SQL Server Management Studio, or PGAdmin 2.

Initial Configuration

Once deployed, connect to the server on port 8087/HTTP or 8443/HTTPS. It is recommended that all users leverage the configuration wizard once logged in, in order to do the initial install, as it guides the configuration via a series of questions and prompts in order to build a valid configuration. Please see the AWS specific instructions for information on setting an IAM role for AWS configuration auto-detection.

Once setup, the data source tab provides a test source button to validate access to the database. Likewise, a test VDB option is available on the VDB to pass some basic traffic to the VDB. **Note**: When using the test VDB option, it requires access to create a new schema or database (named Heimdall) and tables within it. If the data source is not configured with a user that has access to do this, then the test VDB option will fail, even if everything is configured properly for application access.

Install Process (Detailed)

For instructions in AWS, Azure or GCP, please see the environment specific details.

Typical Install

To begin with, ensure to provision a Linux VM or bare metal instance with at least two CPU cores and 4GB of RAM. Login and install with the following command:

```
sudo bash -c 'bash <(curl -s http://s3.heimdalldata.com/hdinstall.sh) server'
```

While executing, output similar to the following should be generated (this is on Ubuntu 18.04, your output may vary):

```
This script may take time to install dependencies and download, please be patient!
Detected distro: ubuntu
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
...
Get:28 http://security.ubuntu.com/ubuntu bionic-security/multiverse Translation-en [2356 B]
Fetched 17.9 MB in 4s (4173 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  libcurl4 libhavege1 libopts25 libpython-stdlib libpython2.7-minimal libpython2.7-stdlib python python-meld3 python-minimal pytl
Suggested packages:
  ntp-doc python-doc python-tk python-setuptools python2.7-doc binutils binfmt-support supervisor-doc zip
Recommended packages:
  sntp
The following NEW packages will be installed:
  haveged libhavege1 libopts25 libpython-stdlib libpython2.7-minimal libpython2.7-stdlib ntp python python-meld3 python-minimal
  unzip
The following packages will be upgraded:
  curl libcurl4
2 upgraded, 15 newly installed, 0 to remove and 60 not upgraded.
Need to get 5663 kB of archives.
After this operation, 21.8 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 libopts25 amd64 1:5.18.12-4 [58.2 kB]
...
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic/universe amd64 supervisor all 3.3.1-1.1 [256 kB]
Fetched 5663 kB in 0s (31.4 MB/s)
Selecting previously unselected package libopts25:amd64.
(Reading database ... 56638 files and directories currently installed.)
Preparing to unpack .../0-libopts25_1%3a5.18.12-4_amd64.deb ...
Unpacking libopts25:amd64 (1:5.18.12-4) ...
...
Selecting previously unselected package supervisor.
Preparing to unpack .../8-supervisor_3.3.1-1.1_all.deb ...
Unpacking supervisor (3.3.1-1.1) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Setting up libcurl4:amd64 (7.58.0-2ubuntu3.8) ...
Setting up unzip (6.0-21ubuntu1) ...
Setting up libhavege1:amd64 (1.9.1-6) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for systemd (237-3ubuntu10.24) ...
Setting up libopts25:amd64 (1:5.18.12-4) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up libpython2.7-stdlib:amd64 (2.7.15-4ubuntu4~18.04.1) ...
Setting up curl (7.58.0-2ubuntu3.8) ...
Setting up ntp (1:4.2.8p10+dfsg-5ubuntu7.1) ...
Created symlink /etc/systemd/system/network-pre.target.wants/ntp-systemd-netif.path → /lib/systemd/system/ntp-systemd-netif.path
Created symlink /etc/systemd/system/multi-user.target.wants/ntp.service → /lib/systemd/system/ntp.service.
ntp-systemd-netif.service is a disabled or a static unit, not starting it.
Setting up python2.7 (2.7.15-4ubuntu4~18.04.1) ...
Setting up haveged (1.9.1-6) ...
Created symlink /etc/systemd/system/default.target.wants/haveged.service → /lib/systemd/system/haveged.service.
Setting up libpython-stdlib:amd64 (2.7.15~rc1-1) ...
Setting up python (2.7.15~rc1-1) ...
Setting up python-meld3 (1.0.2-2) ...
Setting up python-pkg-resources (39.0.1-2) ...
Setting up supervisor (3.3.1-1.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/supervisor.service → /lib/systemd/system/supervisor.service.
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.24) ...
Reading package lists...
Building dependency tree...
Reading state information...
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fontconfig-config fonts-dejavu-core java-common libasound2 libasound2-data libavahi-
  libfontconfig1 libjpeg-turbo8 libjpeg8 liblcms2-2 libnspr4 libnss3 libpcsclite1 libxi6 libxrender1 libxtst6 openjdk-11-jdk-hea
Suggested packages:
  default-jre libasound2-plugins alsa-utils cups-common liblcms2-utils pcscd openjdk-11-demo openjdk-11-source libnss-mdns fonts
  fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  ca-certificates-java default-jdk-headless default-jre-headless fontconfig-config fonts-dejavu-core java-common libasound2 liba
  libavahi-common3 libcups2 libfontconfig1 libjpeg-turbo8 libjpeg8 liblcms2-2 libnspr4 libnss3 libpcsclite1 libxi6 libxrender1 l
  x11-common
0 upgraded, 25 newly installed, 0 to remove and 60 not upgraded.
Need to get 232 MB of archives.
```

```
After this operation, 383 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libjpeg-turbo8 amd64 1.5.2-0ubuntu5.18.04.1 [110
...
Get:25 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-updates/main amd64 default-jdk-headless amd64 2:1.11-68ubuntu1~18.0
Fetched 232 MB in 4s (54.9 MB/s)
Selecting previously unselected package libjpeg-turbo8:amd64.
(Reading database ... 57703 files and directories currently installed.)
Preparing to unpack .../00-libjpeg-turbo8_1.5.2-0ubuntu5.18.04.1_amd64.deb ...
Unpacking libjpeg-turbo8:amd64 (1.5.2-0ubuntu5.18.04.1) ...
Selecting previously unselected package java-common.
...
Setting up libxi6:amd64 (2:1.7.9-1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Setting up liblcms2-2:amd64 (2.9-1ubuntu0.1) ...
Setting up libpcsclite1:amd64 (1.8.23-1) ...
Setting up fonts-dejavu-core (2.37-1) ...
Setting up libasound2-data (1.1.3-5ubuntu0.2) ...
Setting up java-common (0.68ubuntu1~18.04.1) ...
Setting up libjpeg-turbo8:amd64 (1.5.2-0ubuntu5.18.04.1) ...
Setting up libnspr4:amd64 (2:4.18-1ubuntu1) ...
Setting up libasound2:amd64 (1.1.3-5ubuntu0.2) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for systemd (237-3ubuntu10.24) ...
Setting up libxrender1:amd64 (1:0.9.10-1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up x11-common (1:7.7+19ubuntu7.1) ...
update-rc.d: warning: start and stop actions are no longer supported; falling back to defaults
Processing triggers for ca-certificates (20180409) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
Setting up libavahi-common-data:amd64 (0.7-3.1ubuntu1.2) ...
Setting up libjpeg8:amd64 (8c-2ubuntu8) ...
Setting up fontconfig-config (2.12.6-0ubuntu2) ...
Setting up libnss3:amd64 (2:3.35-2ubuntu2.3) ...
Setting up libxtst6:amd64 (2:1.2.3-1) ...
Setting up libavahi-common3:amd64 (0.7-3.1ubuntu1.2) ...
Setting up libfontconfig1:amd64 (2.12.6-0ubuntu2) ...
Setting up libavahi-client3:amd64 (0.7-3.1ubuntu1.2) ...
Setting up libcups2:amd64 (2.2.7-1ubuntu2.7) ...
Setting up ca-certificates-java (20180516ubuntu1~18.04.1) ...
head: cannot open '/etc/ssl/certs/java/cacerts' for reading: No such file or directory
Adding debian:SecureTrust_CA.pem
...
Adding debian:OISTE_WISeKey_Global_Root_GA_CA.pem
done.
Setting up default-jre-headless (2:1.11-68ubuntu1~18.04.1) ...
Setting up openjdk-11-jre-headless:amd64 (11.0.4+11-1ubuntu2~18.04.3) ...
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/rmid to provide /usr/bin/rmid (rmid) in auto mode
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/java to provide /usr/bin/java (java) in auto mode
...
update-alternatives: using /usr/lib/jvm/java-11-openjdk-amd64/bin/jhsdb to provide /usr/bin/jhsdb (jhsdb) in auto mode
Setting up default-jdk-headless (2:1.11-68ubuntu1~18.04.1) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for ureadahead (0.100.0-21) ...
Processing triggers for systemd (237-3ubuntu10.24) ...
Processing triggers for ca-certificates (20180409) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
/dev/fd/63: line 203: [[: 0 2019-07-16: syntax error in expression (error token is "2019-07-16")   # ignore if observed, due to (
/dev/fd/63: line 207: [[: 0 2019-07-16: syntax error in expression (error token is "2019-07-16")   # ignore if observed
Downloading Heimdall install package--this may take a while.
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  158M  100  158M    0     0  52.6M      0  0:00:03  0:00:03 --:--:-- 52.6M
Decompressing zip file
Archive:  heimdall.zip
   creating: heimdall/
   creating: heimdall/devops/
  inflating: heimdall/devops/packer-build.sh
  inflating: heimdall/devops/packer-template
  inflating: heimdall/devops/packer-setup.sh
  inflating: heimdall/statechange-greenplum-source.py
  inflating: heimdall/heimdalldriver.jar
```

```
 creating: heimdall/config/
inflating: heimdall/config/Oracle11Java6_1.conf
inflating: heimdall/config/Oracle12cJava6_1.conf
inflating: heimdall/config/MySQL_1.conf
inflating: heimdall/config/MS-SQL-Server_1.conf
inflating: heimdall/config/admin_1.conf
inflating: heimdall/config/Oracle11_1.conf
inflating: heimdall/config/PostgreSQL_1.conf
inflating: heimdall/config/heimdall.conf
inflating: heimdall/proxy-start.sh
inflating: heimdall/heimdallserver.sh
inflating: heimdall/heimdallserver.jar
inflating: heimdall/license.rtf
inflating: heimdall/NOTICE.txt
 creating: heimdall/static/
 creating: heimdall/static/drivers/
 creating: heimdall/static/drivers/MS-SQL-Server/
inflating: heimdall/static/drivers/MS-SQL-Server/mssql-jdbc-6.4.0.jre7.jar
 creating: heimdall/static/drivers/PostgreSQL/
inflating: heimdall/static/drivers/PostgreSQL/postgresql-42.2.5.jre6.jar
 creating: heimdall/static/drivers/MySQL/
inflating: heimdall/static/drivers/MySQL/mysql-connector-java-5.1.47.jar
 creating: heimdall/static/drivers/Oracle11Java6/
inflating: heimdall/static/drivers/Oracle11Java6/Oracle11gR2-ojdbc6.jar
 creating: heimdall/static/drivers/Oracle12cJava6/
inflating: heimdall/static/drivers/Oracle12cJava6/Oracle12cR1-ojdbc6.jar
 creating: heimdall/static/modules/
inflating: heimdall/static/modules/heimdallsqlserver-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallforward-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallloadbalancer-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallfirewall-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallhazelcast-3.6.8-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallredis-3.5.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallpw-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallauth-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallpostgres-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdalltrigger-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallextractplan-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallmysql-1.0-19.7.21.1.jar
inflating: heimdall/static/modules/heimdallasync-1.0-19.7.21.1.jar
 creating: heimdall/super/
inflating: heimdall/super/readme.txt
inflating: heimdall/super/heimdallproxy.conf
inflating: heimdall/super/heimdallserver.conf
inflating: heimdall/hdinstall.sh
Failed to stop supervisord.service: Unit supervisord.service not loaded.
Failed to disable unit: Unit file supervisord.service does not exist.
Created symlink /etc/systemd/system/multi-user.target.wants/supervisord.service → /etc/systemd/system/supervisord.service.
started supervisord via systemctl
The server should now be online, and available on http://<server ip>:8087/
Please refer to the help on the login page for the default username and password
```

This should bring online the UI on port 8087. To check, execute:

```
netstat -alnp | grep "8087.*java"
```

A line should appear similar to:

```
tcp6       0      0 :::8087                 :::*                    LISTEN      7464/java
```

Manual Install Behind Firewall

If your linux instance is behind a firewall, and can't access the Internet, please use the following to install Heimdall:

1. Install the following dependencies manually per your Linux distribution's standards: OpenJDK 8+ (Oracle Java is acceptable as well) unzip supervisord (/etc/supervisor/conf.d is assumed for service configuration)
2. mkdir /opt; cd /opt
3. Download the following files into /opt: https://s3.amazonaws.com/s3.heimdalldata.com/hdinstall.sh https://s3.amazonaws.com/s3.heimdalldata.com/heimdall.zip
4. chmod a+x /opt/hdinstall.sh
5. ./hdinstall.sh server

This should result in output similar to the following:

```
isProxy: false
This script may take time to install dependencies and download, please be patient!
Detected distro: ubuntu
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 252 kB in 0s (764 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
unzip is already the newest version (6.0-21ubuntu1).
haveged is already the newest version (1.9.1-6).
supervisor is already the newest version (3.3.1-1.1).
curl is already the newest version (7.58.0-2ubuntu3.8).
ntp is already the newest version (1:4.2.8p10+dfsg-5ubuntu7.1).
0 upgraded, 0 newly installed, 0 to remove and 60 not upgraded.
./hdinstall.sh: line 205: [[: 0 2019-07-16: syntax error in expression (error token is "2019-07-16")   # ignore if observed, due
./hdinstall.sh: line 209: [[: 0 2019-07-16: syntax error in expression (error token is "2019-07-16")   # ignore if observed
Decompressing zip file
Archive:  heimdall.zip
   creating: heimdall/
   creating: heimdall/devops/
  inflating: heimdall/devops/packer-build.sh
  inflating: heimdall/devops/packer-template
  inflating: heimdall/devops/packer-setup.sh
  inflating: heimdall/statechange-greenplum-source.py
  inflating: heimdall/heimdalldriver.jar
   creating: heimdall/config/
  inflating: heimdall/config/Oracle11Java6_1.conf
  inflating: heimdall/config/Oracle12cJava6_1.conf
  inflating: heimdall/config/MySQL_1.conf
  inflating: heimdall/config/MS-SQL-Server_1.conf
  inflating: heimdall/config/admin_1.conf
  inflating: heimdall/config/Oracle11_1.conf
  inflating: heimdall/config/PostgreSQL_1.conf
  inflating: heimdall/config/heimdall.conf
  inflating: heimdall/proxy-start.sh
  inflating: heimdall/heimdallserver.sh
  inflating: heimdall/heimdallserver.jar
  inflating: heimdall/license.rtf
  inflating: heimdall/NOTICE.txt
   creating: heimdall/static/
   creating: heimdall/static/drivers/
   creating: heimdall/static/drivers/MS-SQL-Server/
  inflating: heimdall/static/drivers/MS-SQL-Server/mssql-jdbc-6.4.0.jre7.jar
   creating: heimdall/static/drivers/PostgreSQL/
  inflating: heimdall/static/drivers/PostgreSQL/postgresql-42.2.5.jre6.jar
   creating: heimdall/static/drivers/MySQL/
  inflating: heimdall/static/drivers/MySQL/mysql-connector-java-5.1.47.jar
   creating: heimdall/static/drivers/Oracle11Java6/
  inflating: heimdall/static/drivers/Oracle11Java6/Oracle11gR2-ojdbc6.jar
   creating: heimdall/static/drivers/Oracle12cJava6/
  inflating: heimdall/static/drivers/Oracle12cJava6/Oracle12cR1-ojdbc6.jar
   creating: heimdall/static/modules/
  inflating: heimdall/static/modules/heimdallsqlserver-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallforward-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallloadbalancer-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallfirewall-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallgeode-1.8.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallcoherence-12.2.1.3.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallhazelcast-3.6.8-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallredis-3.5.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallignite-2.6.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallpw-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallauth-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallpostgres-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdalltrigger-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallextractplan-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallmysql-1.0-19.7.21.1.jar
  inflating: heimdall/static/modules/heimdallasync-1.0-19.7.21.1.jar
   creating: heimdall/super/
  inflating: heimdall/super/readme.txt
  inflating: heimdall/super/heimdallproxy.conf
  inflating: heimdall/super/heimdallserver.conf
  inflating: heimdall/hdinstall.sh
Removed /etc/systemd/system/multi-user.target.wants/supervisord.service.
Created symlink /etc/systemd/system/multi-user.target.wants/supervisord.service → /etc/systemd/system/supervisord.service.
```

```
    started supervisord via systemctl
    The server should now be online, and available on http://<server ip>:8087/
    Please refer to the help on the login page for the default username and password
```

This should bring online the UI on port 8087. To check, execute:

```
netstat -alnp | grep "8087.*java"
```

A line should appear similar to:

```
tcp6       0      0 :::8087                 :::*                    LISTEN      7464/java
```

Docker Install

First download the Docker file:

```
wget https://s3.amazonaws.com/s3.heimdalldata.com/Dockerfile
```

Next, build the base image, specifying the version or name you wish to use as a tag:

```
docker build -t "heimdall:current" .
```

This should result in logs similar to:

```
    Sending build context to Docker daemon    351MB
    Step 1/9 : FROM ubuntu:latest
    latest: Pulling from library/ubuntu
    35c102085707: Pull complete
    251f5509d51d: Pull complete
    8e829fe70a46: Pull complete
    6001e1789921: Pull complete
    Digest: sha256:d1d454df0f579c6be4d8161d227462d69e163a8ff9d20a847533989cf0c94d90
    Status: Downloaded newer image for ubuntu:latest
     ---> a2a15febcdf3
    Step 2/9 : EXPOSE 8087
     ---> Running in dc59ab49c878
    Removing intermediate container dc59ab49c878
     ---> aa80982c4f15
    Step 3/9 : EXPOSE 5432
     ---> Running in f06c8b383fcb
    Removing intermediate container f06c8b383fcb
     ---> d1ddf2b32310
    Step 4/9 : EXPOSE 5050
     ---> Running in 9c1b043ac1d3
    Removing intermediate container 9c1b043ac1d3
     ---> 0346b8d31eeb
    Step 5/9 : RUN DEBIAN_FRONTEND=noninteractive apt-get -y update && DEBIAN_FRONTEND=noninteractive apt-get -y upgrade && DEBIAN_FI
     ---> Running in a2107cee8a89
    Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
    Get:2 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [4952 B]
    ...
      inflating: heimdall/static/modules/heimdallasync-1.0-19.7.21.1.jar
       creating: heimdall/super/
      inflating: heimdall/super/readme.txt
      inflating: heimdall/super/heimdallproxy.conf
      inflating: heimdall/super/heimdallserver.conf
      inflating: heimdall/hdinstall.sh
    Removing intermediate container 37ddb6bcb143
     ---> 92b29de604ae
    Step 7/9 : RUN mv heimdall /opt && cp /opt/heimdall/super/heimdallserver.conf /etc/supervisor/conf.d
     ---> Running in 48e149131223
    Removing intermediate container 48e149131223
     ---> 3b4b5a66bd11
    Step 8/9 : RUN sed -i 's/^\(\[supervisord\]\)$/\1\nnodaemon=true/' /etc/supervisor/supervisord.conf
     ---> Running in 230a85b07000
    Removing intermediate container 230a85b07000
     ---> 460248ac1b87
    Step 9/9 : CMD ["supervisord", "-c", "/etc/supervisor/supervisord.conf"]
     ---> Running in 9c20424bec22
    Removing intermediate container 9c20424bec22
     ---> b3372e92bd83
    Successfully built b3372e92bd83
    Successfully tagged heimdall:current
```

Now, run the instance, exposing the ports desired for management and proxy ports:

```
docker run -d --name heimdall-instance -p 8087:8087 -p 3306:3306 -p 5432:5432 -p 1433:1433 heimdall:current
```

Finally, verify that the docker instance is running:

```
docker ps -a
```

Result should be similar to:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
1c5ed6913450        heimdall:current    "supervisord -c /etc…"   2 minutes ago       Up 2 minutes        0.0.0.0:1433->1433/tcp, 0.0
```

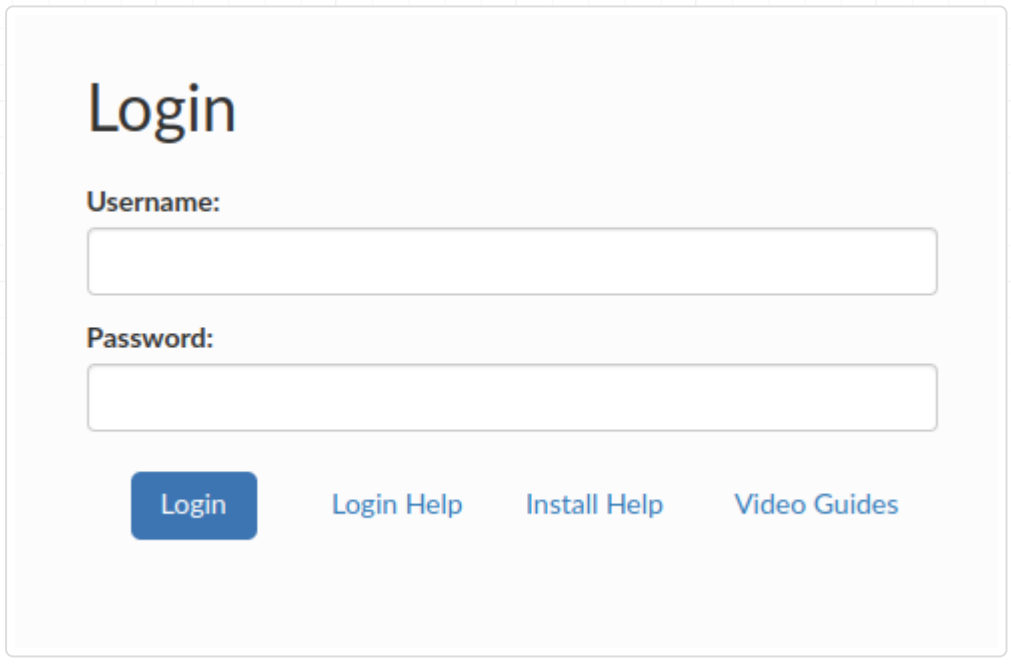
Application Configuration

If everything is setup properly, then an application's configuration can be set to point to the Heimdall Hostname (or the NLB hostname for a Cloudformation deployment), wit the port as specified in the proxy configuration (typically matching the default for the database type). This configuration is application and framework dependent, so please consult the application's documentation on how to adjust this configuration.

Lambda and Container Configuration

When using lambdas and containers, no special instructions are needed--the application will have a database configuration, and you would configure it to point to either the proxy or the NLB endpoint pointing to the proxy, as appppropriate.

Initial Login



The default Heimdall server username is **“admin”** and the password is:

“heimdall”.

In AWS and GCP and similar environments (such as OpenStack), the **Cloud instance ID** is the default password. For Oracle cloud, it is the instance ocid. For Azure, it is the subscription ID.

If the password is not set to “heimdall” please see the server log (/var/log/heimdallserver.out.log or /opt/heimdall/log/heimdall.log) and look for a line such as:

```
2016-12-11 20:10:32,516 5153 [INFO] [http-nio-8087-exec-1] Initializing access control, user=admin,
password=i-0aa1234a1a1a12abc
```

Note: It doesn't matter what install method you use--if in a cloud provider, as long as the software can detect the instance ID, it will default to the instance ID for the password. The following sources will be used to attempt to detect a cloud instance ID:

- http://169.254.169.254/latest/meta-data/instance-id (AWS & compatible environments)
- http://169.254.169.254/metadata/instance/compute/subscriptionId?api-version=2020-09-01&format=text (Azure)
- http://169.254.169.254/computeMetadata/v1/instance/id (GCP)
- http://169.254.169.254/opc/v2/instance/id (Oracle Cloud)

Note: As Azure doesn't expose the vmID to users easily, the subscription ID is used instead

User Reset

The default username and password for a server can be controlled explicitly by creating a file /etc/heimdall.conf prior to the server being first run, and add the lines:

- hduser={username}
- hdpassword={password}

For more details of a file `/etc/heimdall.conf`, please see heimdall.conf configuration.

When the http query “http://169.254.169.254/latest/meta-data/instance-id” returns a value, this value is used as the default password if no configuration is set in the /etc/heimdall.conf. This will happen in AWS instances, and some other cloud environments. This value can be obtained with:

```
echo `curl -s http://169.254.169.254/latest/meta-data/instance-id`
```

In the event the passwords need to be deleted from a current configuration, the following steps can be used to delete the configured accounts:

1. Open the file $heimdallhome/config/heimdall.conf (the default install directory is /opt/heimdall/) in an editor (notepad, vi, etc)
2. Remove the json array values for the “address” variable, i.e. replace:

“addresses”:[{“enabled”:true,”file”:”config/admin_1.conf”,”name”:”admin”,”version”:1}]

With

```
“addresses”:[]
```

Once saved, if all accounts are removed, then the default admin account will be re-initialized once the Heimdall server is restarted.

To automate the password reset process you can do the following:

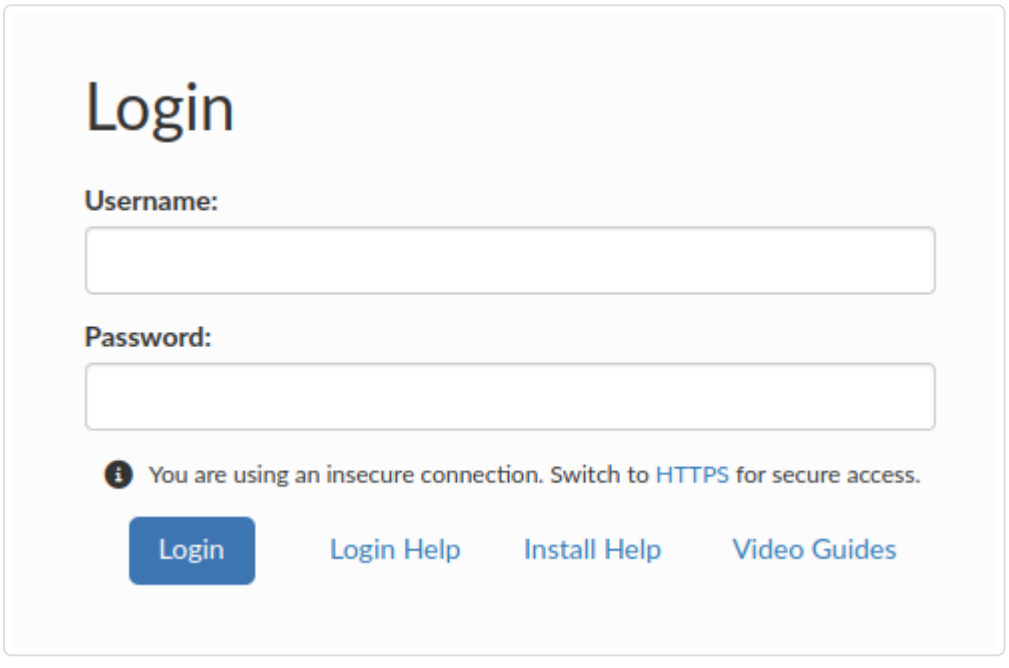
```
jq 'del(.addresses[])' < /opt/heimdall/config/heimdall.conf > /opt/heimdall/config/heimdall.conf.out
mv -f /opt/heimdall/config/heimdall.conf.out /opt/heimdall/config/heimdall.conf
service heimdall restart
```

Note: newer installs should include jq by default, but older will not, and will need to use the distribution's package installer to install jq.

File Base User Reset

The default username and password for a server can be reset by creating a file `/etc/heimdall_reset_password`. This operation will only change the credentials for an admin user, the default admin account will be re-initialized once the Heimdall server is restarted. When this operation succeed the file will be deleted.

Insecure connection



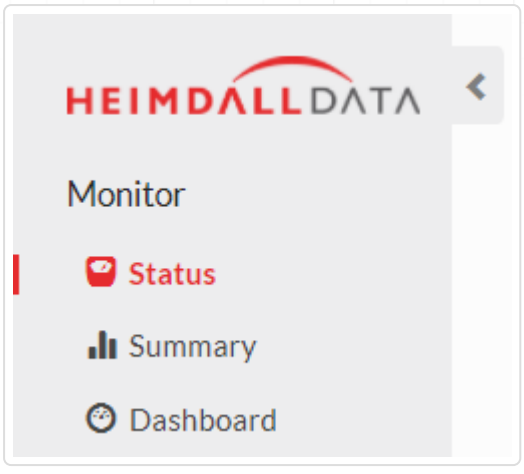
In case of an insecure connection, a warning 'You are using an insecure connection. Switch to HTTPS for secure access' appears, with a clickable 'HTTPS' link, which enables users to easily switch to a secure HTTPS protocol.

Navigation Overview

All windows have two main sets of controls for navigation, the sidebar to the left and the settings bar on the upper right. Additionally, just below the settings bar, if present, current alerts can be displayed.

Sidebar

The sidebar allows navigation between all the various tabs in the application. The current tab is highlighted in red. Via the < and > arrow beside the logo, the sidebar can be collapsed to reduce the amount of space taken:

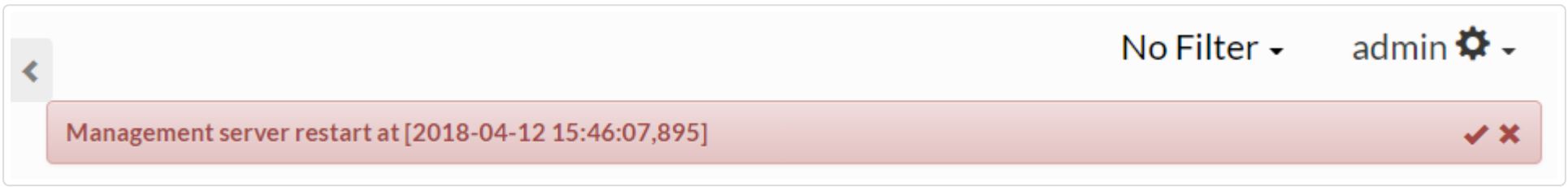


collapses to:

Settings Bar

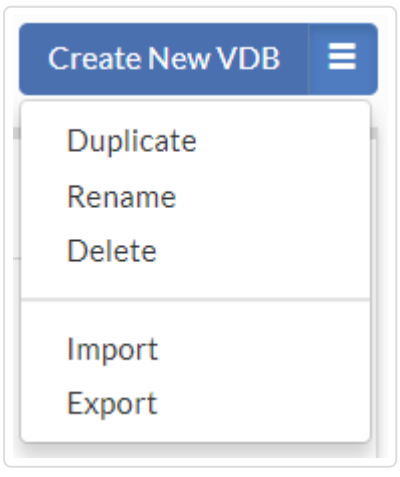
The settings bar currently provides the currently logged in user, and a gear. The gear provides options to sign our or change the password. Future settings options may also be presented here.

Alerts



In the alert section, up to three alerts may be displayed. In order to hide (but not delete) an alert, use the check. In order to permanently delete an alert, use the X. All alerts can be managed in the admin->alerts section as well.

Object Management



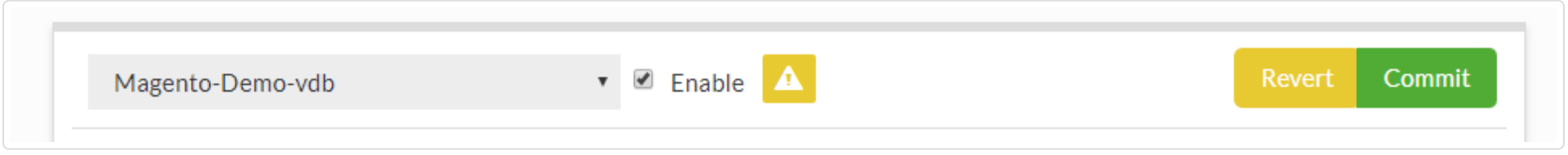
For each section that manages a configuration object, there is a menu in the upper right-hand of the configuration section. Options will include:

- **Create New** : To create a new object of this type

- **Duplicate**: In order to create an identical copy, but under another name
- **Delete**: To delete the currently selected object
- **Import**: To import a previously exported configuration object
- **Export**: Export the currently selected object

When using the export function, all dependencies are exported with the object, i.e. when exporting a Virtual Database object, it will also export the associated rules and data sources. The exception for this is that the driver objects will not include the actual driver file associated with it, but only the configuration name and settings. If on loading a driver config however, there is already an existing configuration in place, then the data source will point to the existing driver configuration, without loading the settings in the configuration file. This could, for example, result in the loss of customized template URL's for AWS environments. Please refer to the json file for any custom settings and verify they are correct on the restoration of a VDB.

Change Control



When interacting with configuration objects, any time a change is made to the object, it is flagged as changed using a yellow alert icon at the top, and the option to revert or commit the configuration. In the sidebar, any section with pending changes will be colored yellow. By selecting revert, the change is prevented, and the configuration restored to the initial state, while a commit will push the configuration to the server and the proxies/drivers.

Context Help

Nearly every label in the system is configured to provide a tool-tip for additional help when the mouse hovers over it. Additionally, many icons used will provide additional information in this way, such as the icons on the rules tab.

Post Install Operations

Health-Checks & Monitoring

Heimdall is divided into two components, the management server and the proxy or driver (it is the same code). This corresponds to the control plane and the data plane. The management server can go offline without impacting the proxy, although when offline, configuration changes can not be orchestrated. To monitor the accessibility of the management server, a monitoring system can use the default port of 8087 (see below in security for variations).

In order to add monitoring to the proxy nodes, say for load balancing purposes, an HTTP port can be opened up that provides the status of the system, based on the ability to completely pass a health check request through the proxy to the back-end database. The options to configure this are in the VDB documentation, under vdb properties, specifically the "healthcheck-port" and "healthcheck-interval". Once configured and the proxy restarted, a query against the url "/status" will report either "OK" with a "200 OK" status or "FAILED" with a "500 Server Error" status.

Additional monitoring can be enabled by selecting "AWS CloudWatch" when in AWS, and each proxy node will report to cloudwatch various statistics about cache hit rate, cpu load and performance. Please note, a proper IAM role is necessary for this option to be active (please see the security section for details).

Backup

In order to backup a complete configuration of Heimdall, two directories need to be archived. First, /opt/heimdall/config, which contains the json configuration files, and possibly /opt/heimdall/static/drivers, where any modified driver files are stored. If only the stock drivers are used, then this second directory does not need to be archive.

Additionally, individual VDB's can be exported via the menu icon in the upper right hand corner of the settings. This will save the vdb, data source, and rules settings that apply to the VDB. This configuration can then be imported into another instance.

Restore

To restore the configuration, install Heimdall again (if needed) and replace the content of /opt/heimdall/config with the archived versions. If the system is currently online, the new configurations can be read and activated by calling an api call to port 8087 with "/api/config/reload". This is also useful if using a configuration management system to synchronize configurations from an offline source or staging environment.

Maintenance

Heimdall is designed to operate with low maintenance. Some maintenance options that may be desired however:

- Control how logs roll to manage disk space: Log files will roll when the log filesystem reaches 90% by default. This is configurable via the admin->properties->Reserved disk space setting OR the Max Log Age setting.
- New version updates will be e-mailed to the configured notification contact if access to the Internet is provided. The settings can be configured to route the mail through a local mail server as desired.
- Heimdall advertises version updates for four release trains--LTS, which is limited to major bugfixes only on the LTS train, the "release" train for normal updates and a test build. To configure which release build update notifications will be sent as, in the admin->software management tab, simply select the tracking option. Most customers should select the "release" train. The test train may be updated at times with code that a specific customer has been asked to download, and may not be as well tested as the release build, but will likely have new features as well.
- To update to a new version of software, simply select the option in the software management tab. This will require external access to download the new build. The old build files will be saved in the /opt/heimdall directory labeled with a .old suffix in case a manual fallback is needed.

Securing Heimdall

Best Practices

Please follow the following best practices:

- Lock down access by creating an ACL or Security Group to block access from the Internet;
- Create a read-only user for use by developers who need to view logs for monitoring;
- Set the admin password to a secure password;
- If you wish to have logs strip the actual SQL queries, in the VDB, you can set the vdb setting of "paranoia" to true;
- With proxy authentication disabled for Postgres and SQL Server, password passthrough is enabled, so the Database takes care of password authentication. This is not supported for MySQL, as the password is required to be hashed. If authentication is disabled, then for MySQL, it will simply use the data source's configured username and password for authentication.;
- Lock down the interface with HTTPS;
- Set a limited-acccess user for the data source configuration, with access to the "heimdall" schema/database for replication lag detection;
- Enable TLS as desired on the back-end for database access.

Password Storage

Heimdall, in some cases needs to store passwords in a variety of formats for different purposes. Each case is documented here:

1. All configuration information is kept in /opt/heimdall/config, accessible only to root, including password information;
2. By default, for Postgres and SQL Server, VDB authentication is set to "passthrough" which means that individual user's credentials are not stored on Heimdall at all;
3. For MySQL, due to the nature of authentication, individual user's credentials must be configured in effectively plaintext. These are stored in the vdb configuration file, and used to both authenticate the user into the proxy, and then to authenticate the user into the database;
4. For all configurations, an alternative to storing a user's vdb password is to configure SQL based authentication, in which case the passwords will be stored in a database table instead;
5. For GUI authentication, passwords are stored in Salted SHA format, compatible with OpenLDAP, i.e. automation could be configured to copy the same credential format between the two systems.
6. One additional cleartext password is stored in the Data source configuration tab, which is used to authenticate health checks and for performing replication lag detection. No access to actual data is necessary for this user's login.

NOTE: In AWS, it is possible to configure Heimdall to store the entire configuration in an AWS Secret. Please see the AWS Environment page for details on the hdSecretKey option. Using this ensures all passwords in the configuration are also stored encrypted and not on the EBS volume of the server.

Password Rotation

For the GUI passwords, rotation can simply be done via the user's tab. For Database password authentication, there are several categories: 1. In the case of Active directory authenticated users, credentials will be cached on a success for up to five minutes. On a failure however, the credentials will be re-queried against Active directory, and will be updated if success occurs. This ensures that as soon as AD is updated, the proxy will also be updated; 2. In the case of passthrough authentication, connections are established with the user's credentials on connect, so again, as soon as the password is changed, the credentials will be reflected; 3. If the proxy is using "proxy authentication" then the credentials are stored against the VDB configuration itself. To perform a rotation, you can add a new user/password combination **without** removing the old one. During this period, both will be acceptable. Then, once the application and database is updated to use the new credentials, then original password can be removed from the proxy. This allows the issue of timing the password changes betweeen the application and database to ignore the proxy, since it will accept both at the same time.

Supported Encryption

Heimdall allows TLS security to be enabled on each VDB, and to be made required as well. In the case of Postgres and MySQL, TLS 1.3 will be used by default if the client supports it, while TLS 1.3 will be used for SQL Server, as SQL Server itself does not use TLS 1.3 yet, and all driver libraries will need to be updated for this to work.

For back-end connectivity, as we use the vendor JDBC drivers to connect to the database, we support all connectivity options that these drivers support for encryption. Please see the vendor JDBC driver help pages for specific details.

Enabling HTTPS for Management Server

Self-signed certificates

By default, the Tomcat certificate is created automatically with a blank password when the application is started if one doesn't exist. The heimdall.conf file contains a field called "tomcatPassword", it refers to the private key password in keystore.p12 file and can be freely changed. If the field has null value, a new one is created with a blank value. To create own Tomcat certificate, on the Linux command line, do the following (this will generate a self-signed certificate):

```
keytool -genkey -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore /opt/heimdall/keystore.p12 -validity 3650

Enter keystore password: <use "heimdall">
 Re-enter new password:
 What is your first and last name?
 [Unknown]:
 What is the name of your organizational unit?
 [Unknown]:
 What is the name of your organization?
 [Unknown]:
 What is the name of your City or Locality?
 [Unknown]:
 What is the name of your State or Province?
 [Unknown]:
 What is the two-letter country code for this unit?
 [Unknown]:
 Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct?
 [no]: yes
```

Once done, restart the Heimdall server, and port 8443 will be used for HTTPS.

Please see the man page for keytool for importing in an existing SSL certificate if desired.

Certificate Import

In case you wanted to import your own SSL certificate, a keystore with the keypair with alias "tomcat" is required. Then, we can use keytool command to import our custom keystore to the keystore used by Heimdall (new_keystore.p12 is the keystore user wants to import): When prompted, use the destination password of 'heimdall'.

```
keytool -importkeystore -srckeystore new_keystore.p12 -destkeystore /opt/heimdall/keystore.p12 -srcalias tomcat -srckeypass <source_

 Importing keystore new_keystore.p12 to keystore.p12...
 Enter destination keystore password: ******
 Enter source keystore password: ******
 Existing entry alias tomcat exists, overwrite? [no]:  yes
 Enter new alias name   (RETURN to cancel import for this entry):  tomcat
```

VDB Proxy TLS Support

TLS is a way of providing a safe connection between Heimdall server and client application while using proxy. By default Heimdall server enables use of TLS and decision about activating it depends on client side.

The core of TLS in Heimdall server are private keys with certificates stored in a keystore. Changes made in keystore can cause change of behavior of TLS in proxy.

Certificate binding

Heimdall allows binding of certificates in two modes:

- per-vdb: this type of private key with certificate is used only by particular proxies defined in Heimdall server. To use this mode an appropriate configuration is required, which will be shown later.
- global: this type of private key with certificate is used by all proxies which don't have defined particular use certificate in configuration. In keystore can exist only one private key with certificate used as global use certificate. Global use certificate alias in keystore is "global_use_certificate". If global use certificate doesn't exist in keystore, it is going to be autogenerated by Heimdall server.

Default Configuration

By default, when TLS use is called from client side, user doesn't have to configure anything for TLS - Heimdall server will create a keystore and generate a global use certificate.

Created by Heimdall server keystore will have below properties:

- keystore type will be JKS
- keystore file name will be `keystore.p12`
- password to keystore will be `heimdall`
- keystore file will be placed in heimdall server/proxy directory

Generated by Heimdall server global use certificate will have below properties:

- generated will be private key with certificate
- generated pair will have alias `tomcat` (note--this matches the HTTPS certificate for the central manager)
- generated pair password will be blank
- algorithm used to generate key pair will be SHA-256 with RSA
- key size will be 2048
- global use certificate will have self-signed certificate with `CN=Heimdall`
- certificate of global use certificate will be valid from day before the day of generating
- certificate of global use certificate will expire after 10 years minus 1 day

User Configuration

Heimdall server accepts configuring keystore by user, but doesn't provide a functionality for doing this. In this situation suggested is using third-party applications (for example: KeyStore Explorer) to make changes in keystore used by Heimdall server.

Creating keystore

For the situation when user want to create or replace a keystore by himself for Heimdall server, then below restrictions should be met:

- keystore type is JKS (may work with PKCS#12 type)
- keystore file is named `keystore.p12`
- password to keystore is `heimdall`
- keystore file is stored in the same directory as Heimdall server

Important: If a keystore is created with JDK 16 or higher, then you will not be able to use it with JDK lower than 16. The reason for that is JDK 16 uses more secure algorithms by default which are not supported by lower versions.

Reading keystore

For reading keystore we recommend using keytool utility.

Do not use gcr-viewer, cause currently it has a problem with reading certificates based on bouncycastle algorithms.

Managing certificates

The first thing to mention before managing certificates is that Heimdall server is using private keys with certificates (key pairs). Single keys or single certificates won't work with Heimdall Server.

For each vdb configuration it is possible to add data concerning certificates. The following fields must be added:

- certificate - the certificate in PEM format, it may or may not have a header and a footer
- certificateAlias - the alias refers to a certificate, preferably it should also refer to a specific vdb name
- privateKey - the private key in PEM format, it may or may not have a header and a footer

If any of the above fields are left with null value, the certificate configuration will be overwritten by default by global_use_certificate.

Below is an example showing how to add certificate data to vdb configuration:

```
{
  "certificate": "MIICpjCCAY6gAwIBAgIIMtkhw8lTf9gwDQYJKoZIhvcNAQELBQAwEzERMA8GA1UEAxMISGVpbWRhbGwwHhcNMjEwOTE5MDgxOTQ1WhcNMzEwOTE5MD
  "certificateAlias": "global_use_certificate",
  "privateKey": "MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQCK0kAYVKiuWs2zMj4YKBaNqCbz9uUazt8tCCmYhNRC4yAAJ0fNCkTkgOJF+naFA7V
}
```

All changes in key pairs should meet below restrictions:

- alias of key pair should match the restrictions defined for particular and global use certificates, other defined aliases won't be used by Heimdall Server
- key pair password is blank by default using global_use_certificate
- recommended is using key pair with algorithm SHAx with RSA/DSA (not guaranteed that all algorithms is going to work, because it depends on client application too)

Changes made in keystore may require reconnecting from client application to proxy, after this changes should be applied for TLS.

Example scenario of configuration

Simple scenario of creating keystore, and adding global use certificate to keystore:

1. Run application to manage keystore (for example: KeyStore Explorer).
2. Choose option to create new keystore.
3. As type of keystore select type `JKS`.
4. Save created keystore.
5. As password to keystore set `heimdall`.
6. As directory to save choose directory where Heimdall server is placed.
7. After saving, choose option for generating new key pair.
8. Set size of key size as 2048 and as algorithm choose SHA-512 with RSA.
9. Set name of key pair by configuring CN to value `Heimdall`.
10. Go further to find option to configure alias, as alias set value `global_use_certificate`.
11. As password to key pair set blank value.
12. Key pair should be fully configured, save changes in keystore.

Extra informations

This section provides informations which didn't found place in other sections about TLS but may be helpful: - format for key pairs in Heimdall server is X509

Iptables

If you are using Iptables, which blocks ports by default, Heimdall provides a solution. When starting the proxy, the ports for both the proxy and the server are automatically unblocked in iptables. If you cannot connect to the server, try restarting the proxy. It's possible that the management server was not ready yet, and the ports were not unblocked. Moreover, the server also autonomously unblocks its port in iptables during startup. Importantly, the iptables service doesn't exist on Windows systems, so the server and proxy ports won't be unblocked there.

Testing Heimdall

Many customers who are in interested in testing Heimdall will following these steps:

1. Install Heimdall;
2. Use the wizard to configure access to their database;
3. Use a tool such as DBeaver, PGAdmin, or other tool to generate traffic;
4. Observe that despite caching and read/write split being enabled, neither work;
5. Assume Heimdall doesn't work, delete and move on.

The critical flaw here is #3--using tool designed for easy user access to the database to try to test these features. They will not work (in general), for a variety of reasons:

1. Often, they will use mechanisms such as cursors to execute a query, and a cursor is inherently allowed to modify the result on the database. As such, this breaks caching and read/write split;
2. Even if they don't use cursors, they generally apply "pagination" techniques that also break caching;
3. They may use prepared queries, which can break other features as well, such as query multiplexing.

The issues also apply to some benchmark tools, such as PGBench, which is designed specifically for cache busting and leverages cursors as well.

The best way to test Heimdall is with your actual application, not with database user access tools.

Inline SQL Commands

When connecting to the database, Heimdall exposes a few commands that are handled directly by Heimdall vs. by the database. Note: These interceptions ONLY occur when using the simple query protocol via a tool such as PSQL. If your tool uses a prepared interface, as many GUI tools will, these command will not be intercepted.

Show commands

The key use cases for Heimdall database proxy include:

(Postgres only)

- show databases
- show schemas
- show tables
- show users

All databases:

- show pools: Displays a list of user pools and relevant metrics.
- show connections: Displays a list of connections to Heimdall VDBs, along with the following columns and their descriptions:
  - Con ID: Unique identifier for each connection.
  - Conn Age(s): Duration the connection with Heimdall has been active, measured in seconds.
  - Idle(s): Indicates whether the connection is idle (0) or unavailable ("NA") due to an ongoing query execution.
  - Start(s): Elapsed time in seconds since the start of the current query execution, or "NA" if no query is currently running.
  - User: Username associated with the connection.
  - Catalog: Name of the current catalog.
  - Autocommit: Current state of the auto-commit mode for this connection object.
  - sourceName: Name of the datasource to which the connection is established ("none") if there is no underlying datasource connection.
  - Server URL: URL used to connect to the datasource ("none") if there is no underlying datasource connection.
  - Con properties: Properties of the VDB connection ("none") if there is no underlying datasource connection.
  - Tables: Tables involved in the currently executed query.
- show querytracker <tablename> <count>: Displays relevant tracking information for queries tracked in Heimdall, table and count are optional. Only used for cache auto-repopulation (rarely used)
- show queryinfo: Displays attributes associated with the previous query.
- show maxrows: Shows a connection limit on the number of rows any given result-set will try to pull. The default value is 0, meaning the full result-set will be pulled.
- show cache <key>: Displays cache attributes associated with the given key. Doesn't matter whether key starts with "hdkey" prefix.

Control Commands

- set querytimeout: Sets the number of seconds the driver will wait for a Statement object to execute to the given number of seconds. If the limit is exceeded, an SQLException is thrown.
- set cachepool: Adds a value to the cache key to provide unique cache "pools" depending on external factors, unobserved by Heimdall
- set maxrows: Sets a connection limit on the number of rows any given result-set will try to pull, as implemented via the jdbc "setMaxRows()" api.

Other Commands

- drop connection: When in proxy mode, force the connection to drop--useful for using test tools that don't close connections on their own.
- clear pool: Clear server-side connections for current user.
- clear pool <userpool>: Clear server-side connections for a particular user pool, <poolname> must be the username:databasename as displayed in the show pools command.
- reset query cache: Reset all query cache for a virtual database.
- reset query cache <tablename1>,<tablename2>: Reset query cache for all listed table names in CSV format.

Note: To issue clear pool, the user must clear their own pool or be authenticated as one of the users: "postgres", "gpadmin", "root", "sa" or "admin". This will be changed to be configurable in the future.

Compliance

GDPR

The Heimdall Proxy and Manager do not store customer information beyond possibly SQL logs, and this is optional. Further, the "Paranoia" log option will strip SQL query information that may be considered sensitive from any logs.

PCI/HIPAA & Query Caching

When caching, Heimdall stores data in-memory only, and as such is considered in-flight. When connected to an external cache via Redis, an option is provided to use TLS for over the wire encryption.
The Redis configuration should be such that it does not store data to disk (this is the default for Elasticache and most other Redis deployments).

To avoid storing actual cached data to the l2 cache (Redis or Hazelcast) but to only use them for invalidation, you can set the "PCI" option in the cache settings.

SOC2 Compliance

Heimdall does not operate software as a service, so usage of the Heimdall Proxy and associated management components does not fall under SOC2 compliance.

Security Best Practices

By default, Heimdall negotiates the highest level of TLS supported by the database in question on a VDB when TLS is enabled, and will negotiate to the highest level of security common between the client and the version of Java used by Heimdall (default is Java 11). This typically results in TLS 1.2 connections except for SQL Server, as many drivers do not support negotiating with TLS 1.2, so 1.1 is used. If deprecated versions of TLS are needed for old clients, they must be enabled by using the "enable legacy TLS support", but this is not recommended unless needed and appropriate physical security over the connections is assumed.

Heimdall provides a UI interface for uploading of TLS certificates for use with the GUI and VDBs.

When in the AWS cloud environment, configurations (including all passwords) can be stored in an AWS secret.

Export Control Classification Number

Heimdall falls under ECCN 5D992.c. It does not implement itself any cryptographic functions, but instead uses the functions available from Java for the implementation of TLS.
We package by default with the open source OpenJDK, which also falls under ECCN 5D992.c per https://www.oracle.com/us/products/export/eccn-matrix-software-412042.pdf.
Our object code is available for public download to further compliance with 5D992.c.

Help and FAQ

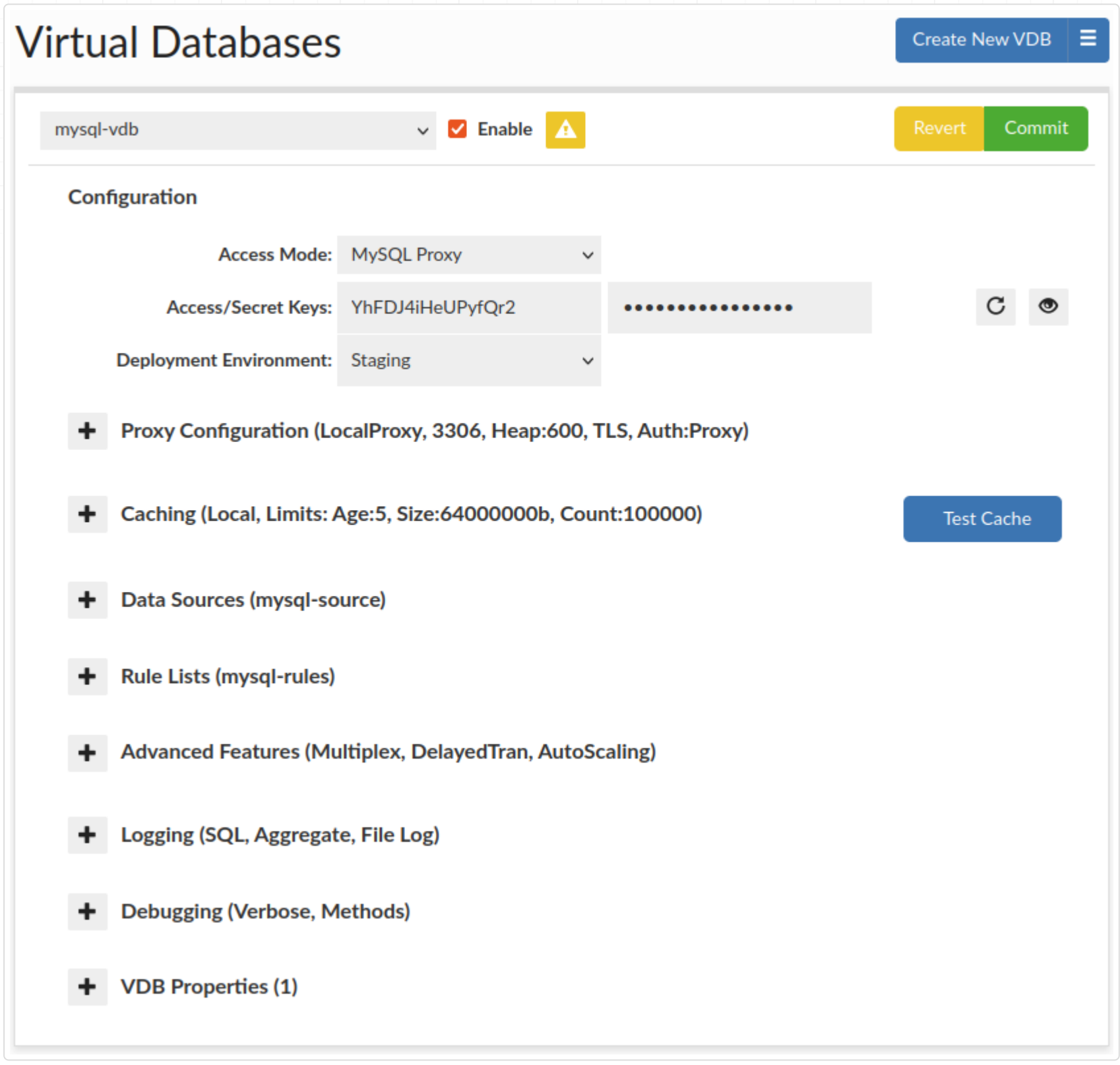
Possible runaway thread

When this alert is generated, it is simply meaning that the named thread has taken nearly 100% cpu on a single core. This can be for many reasons:

1. A client application is reusing the same connection with little to no idle time, and is normal;
2. A query with a very large result-set is streaming the result for more than a minute;
3. A backup is being made via a connection, and is saturating a single core;
4. Related to #2, but possibly a malicious user has gained access to the system, and is downloading lots of data;
5. Also related, but if the proxy is open to the internet, someone may have hacked an account, and is downloading all the data from the database;
6. There is a bug in Heimdall, and a thread is in an infinite loop.

If the thread name contains an IP address, this is a thread being used to service a connection, and likely fits one of the first five cases. If it doesn't contain an IP, please provide logs and a screen dump of the alert for analysis by Heimdall support.

VDB Overview



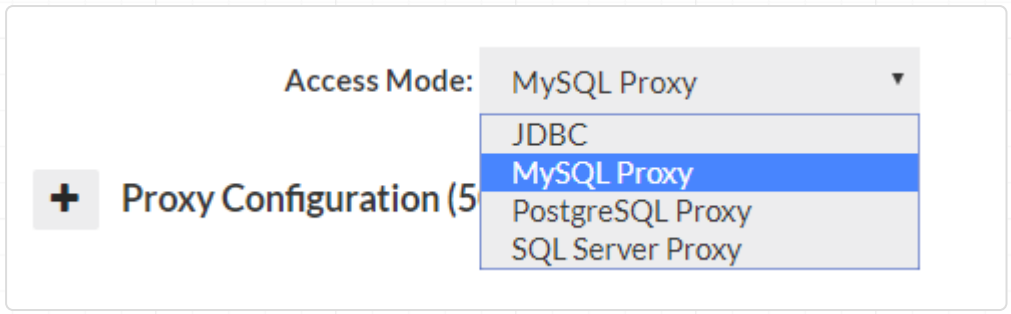
Virtual Databases (VDB) are the basic container for configurations, and aggregate all the settings that will apply to any traffic passed through the VDB. There are two ways to access a VDB, first via JDBC, which supports any JDBC compliant data source, or via a protocol level proxy. In proxy mode, a separate process will be created, and database traffic would be routed through this proxy using a specific port

Enabled: if unchecked, this will result in a proxy disabling access via the port specified. In JDBC mode, this will result in an SQL exception being thrown on a new connection attempt.

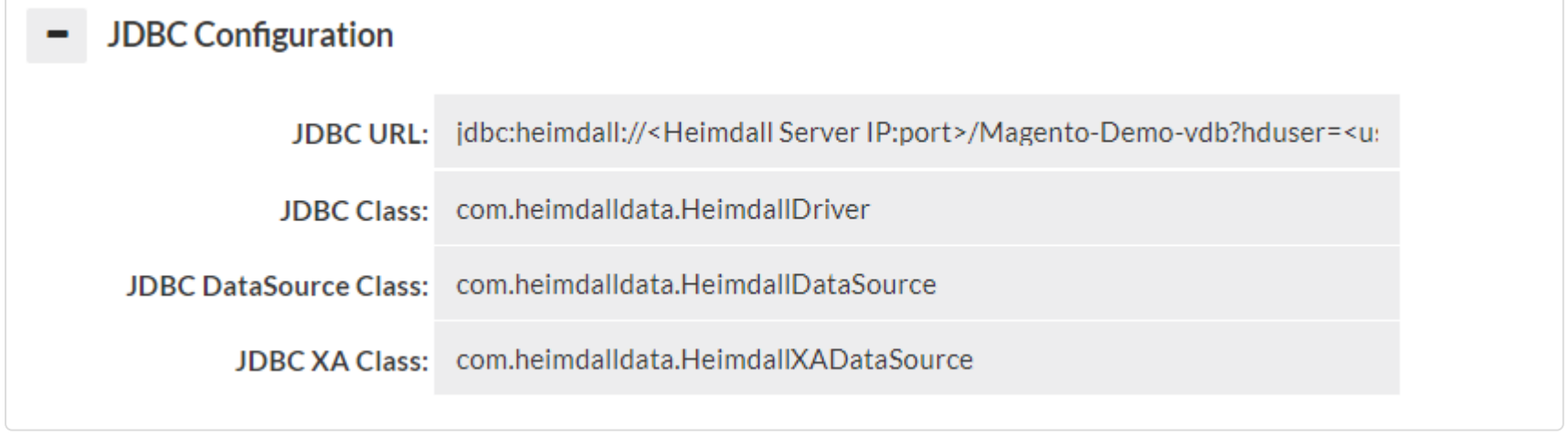
Access/Secret Keys: These are generally used in distributed proxy or JDBC mode to provide a shared user/password to allow the remote proxies to pull the complete vdb configuration.

Access Mode: The first step in configuring a VDB is to select the access mode, which will adjust what options are presented, to simplify the configuration:

Deployment Environment: Records the deployment environment of a VDB, supported values: {Development, Staging, User Acceptance Testing, Production}. Groups VDBs by environment within Status tab.



JDBC Configuration



When in JDBC mode, a JDBC URL is provided in the GUI, which is used to configure the application to access the Heimdall Data Access Layer. The JDBC URL is of the format:

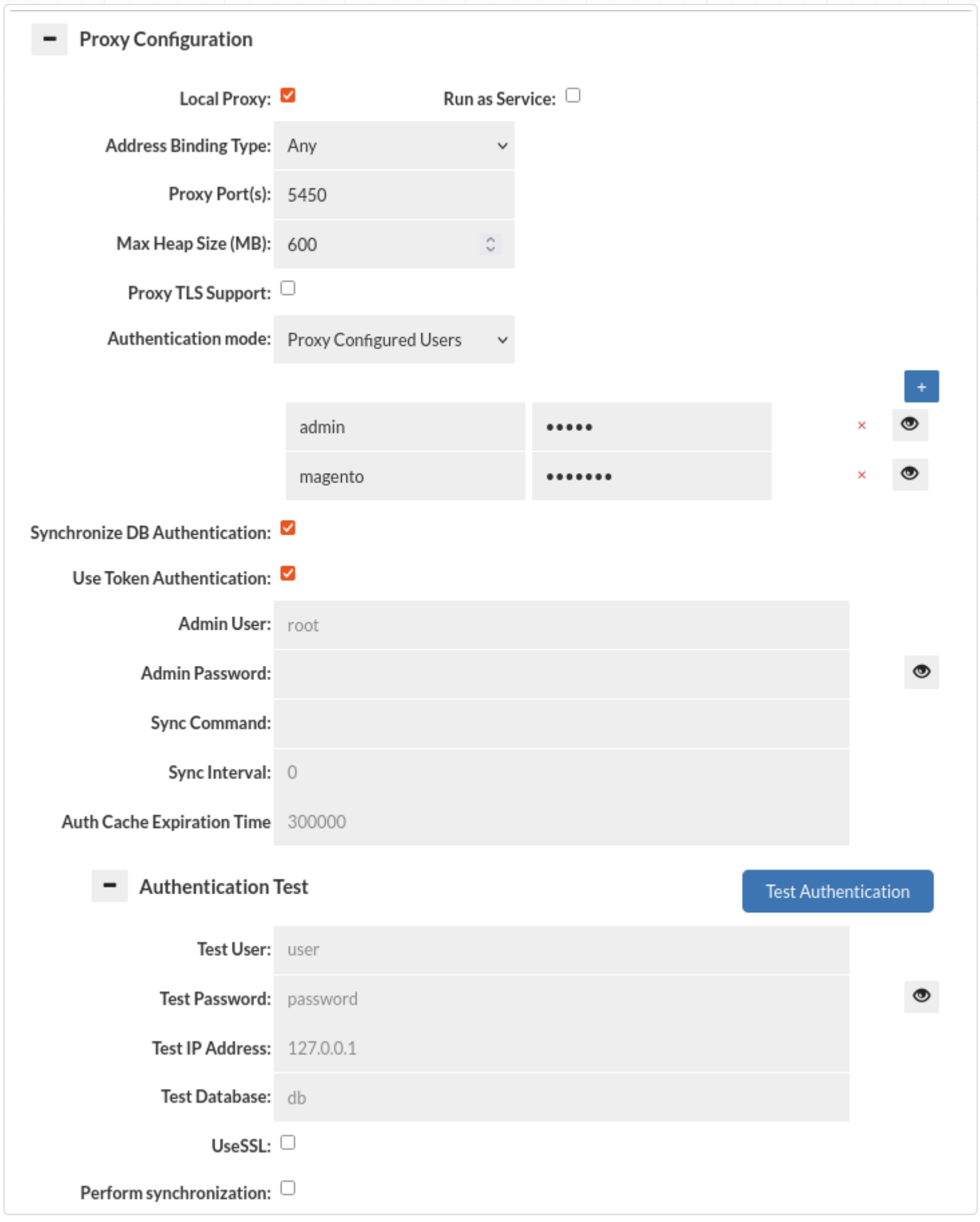
JDBC:heimdall://IP:port/{vdb name}?

Optionally, several parameters can be used in conjunction with the Heimdall Connect string:

- **hduser**={username}: The username to authenticate against the Heimdall server with.
- **hdpassword**={password}: The password for the Heimdall server, again overriding the parameter in the data source.
- **user**={username}: The username to use to authenticate against the database with. If the event the hduser is not specified, this will ALSO be used as the default hduser value, in cases where the users on both match.
- **password**={password}: The password for the database server. In the case where the hdpassword is not specified, this will also be used as the default hdpassword. Note: When using the user and password, the data source credentials will still be used for health checking the application and should have appropriate access for this task.

Note: Typically, only the hduser and hdpassword is used in the Heimdall JDBC URL, as the data source can provide the user and password options. The ability to specify this here is to allow applications to specify their own username and password, in general to allow many users to access a data source at once.

Proxy Configuration



Proxy mode operates by having the management server either start a process on its own, or having a proxy on a remote system connect. Once started, the proxy will open a port that a client can connect to as if the proxy were the database itself, and access data via that process.

- **Local Proxy**: This option allows the management server to start and manage restarts of a proxy instance on the same server as the management server itself. This option is a great way to simplify the testing of Heimdall in limited environments.
- **Run as Service**: This option allows the management server to run a proxy as a linux service. It allows proxy to run even while management server is off.

When in proxy mode, several options will be available, two options being required: the address binding type, and the proxy port. The address binding type specifies the behavior of the proxy:

- **Any**: In this binding mode, all local IP addresses will be bound to, or more specifically, it binds to "0.0.0.0";
- **Localhost Only**: When this option is set, the binding will be to the 127.0.0.1 IP only;
- **Specific IP**: This allows a specific IP address to be bound for use. An example of this is to use 172.17.0.1 in a docker container when bridge mode is enabled, so that the proxy can bind for use by any other containers on the same host, without impacting any other proxies that may reside on another host.

Proxy port: The second required option. This specifies the value of the TCP port that the proxy will be listened to. If there is an error binding to a given port or IP, then a GUI alert will be issued when the proxy attempts to start. It is important that the ports do not conflict with other proxies being run on the same host or binding, as only one process can bind to a specific IP:port combination at once. This applies if installed on the same server as a database as well--if the database is on port 3306 for example, than the proxy can not use the same port. You can set:

- single port by typing value of port (for example: 5433).
- multiple ports by separate them by comma (for example: 5433, 5434, 5435).
- range of ports by separate them by minus (for example: 5433-5435)
- mix (for example: 5433, 5435-5436, 5450)

There are several other options available in the proxy configuration as well:

- **Max Heap Size(MB)**: (only visible if the "Local proxy option is set) The setting for the Proxy java heap memory limit. Default is '600M' which with overhead, will generally consume up to about 1GB of total RAM. This will also result in the setting "Xms" to set the smallest heap size, so as to try to provide more consistency in the "free memory" graph on the dashboard and to ensure that the total memory is always available. Please note, that in distributed proxy mode, this setting has no effect--the java options need to be configured via the user data or heimdall.conf file on the remote instance. For more details of a file `/etc/heimdall.conf`, please see heimdall.conf configuration.

- **Proxy TLS Support**: When enabled, and the client requests it, this option will enable TLS negotiation. Initially, a self-signed certificate will be generated for the proxy, which can then be customized in the Java keystore file in the proxy install directory.

- **Proxy TLS Required**: When TLS is enabled, this option will force all connections to connect only with TLS. Any attempt to connect without TLS will be rejected. For Postgres and MySQL, warnings will be provided to the client on the disconnect, with SQL Server, only an alert on the GUI will be presented.

- **Certificate**: Certificate assigned to Virtual Database for TLS connections.

- **Use Token Authentication**:

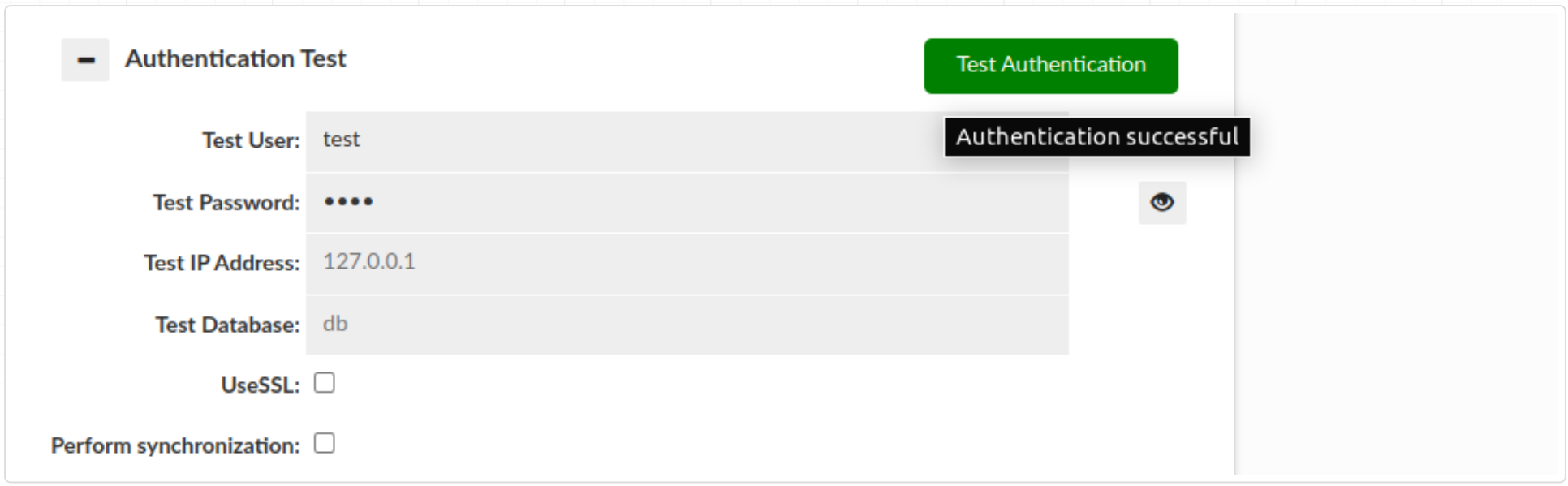
- **Authentication Mode**: This dropdown allows selecting of the proxy authentication mechanisms. Please see the Theory->Proxy Authentication page for more information on authentication.

- **Synchronize DB Authentication**: Specifies if synchronization of users and groups should be performed. Moreover, it allows configuring how often the synchronization is allowed to perform after reconnection (Sync Interval) and how long that data will stay in Authentication Cache (Auth Cache Expiration Time).

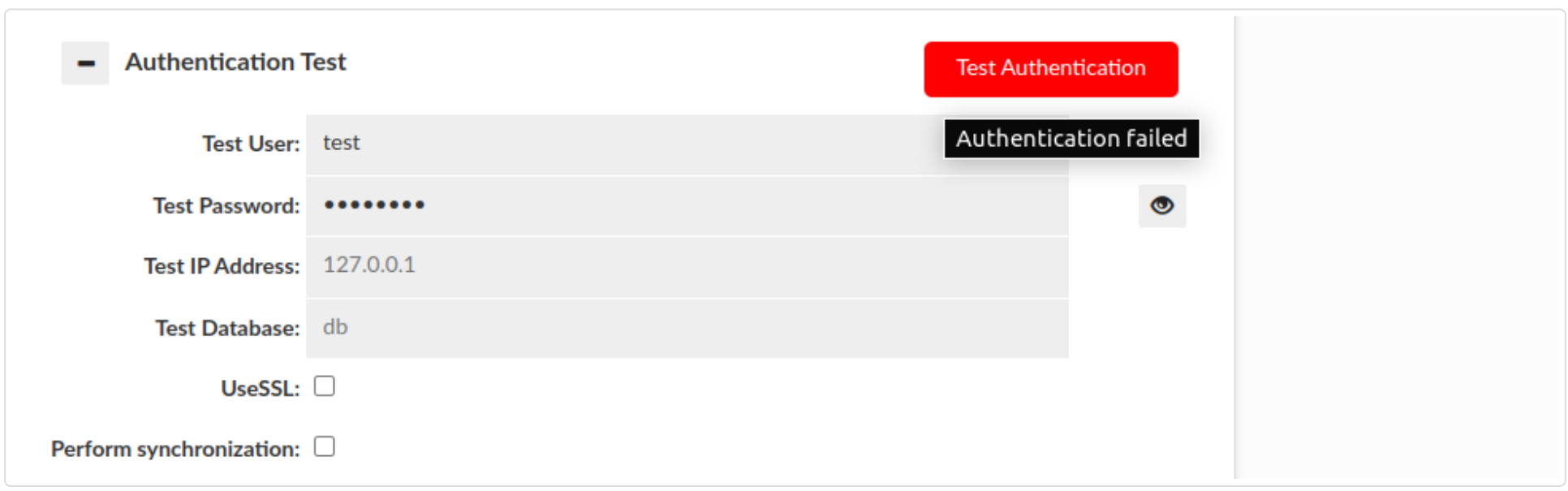
- **Authentication Test**: This section allows testing of authentication at the proxy layer. As full SQL based authentication can provide options such as limiting the scope by source IP, what database (these are taken by default from Data Source JDBC URL), or even if you are connecting via SSL or not, this gives a convient UI to determine if an authentication request would pass or fail the proxy layer. Please note--this still requires authentication at the database level, which is not tested via this interface.

There are two basic results that this test can end with:

Authentication successful:



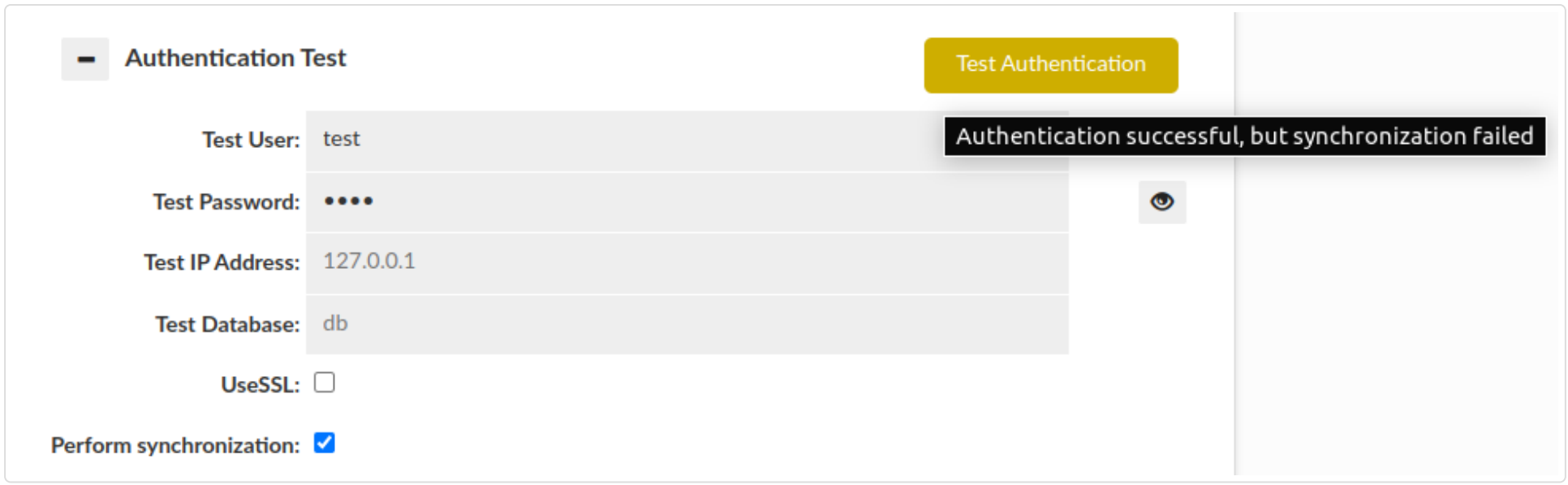
Authentication failed:



- **Perform Synchronization**:

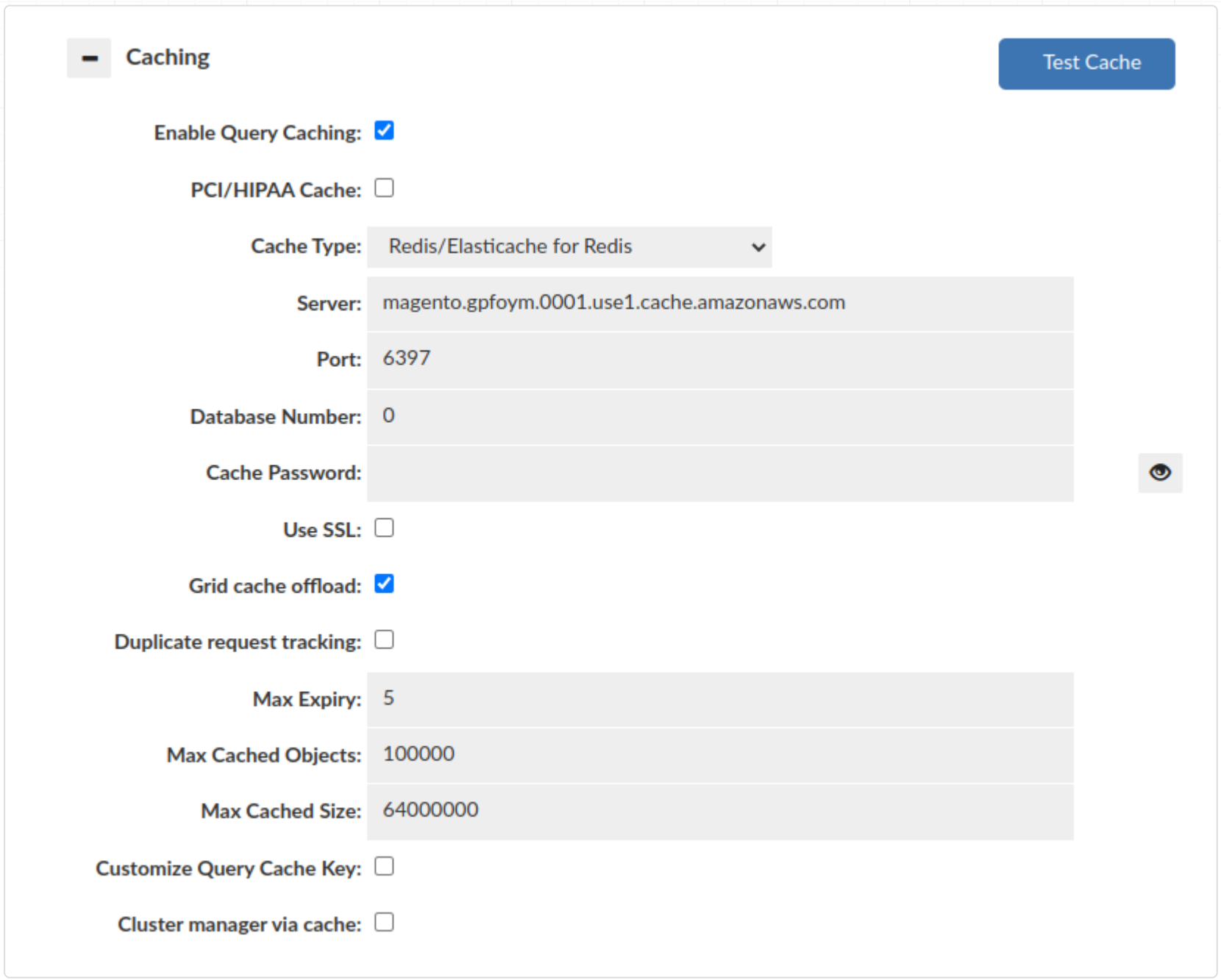
This section also allows us to perform synchronization of user and groups on test by enabling "Perform synchronization" (Synchronization query is retrieved from "Synchronize DB Authentication").

Then some more results can be observed:



Please be aware that if JDBC URL given in the Data Source contains either ${host} or ${database} respectively Test IP Address or Test Database has to be provided. Anyway, in this case if anything is wrong with the configuration, the additional information will be provided as a hover to the button.

Caching



This set of options configures the base cache used by a given VDB. There can be only one cache per VDB at any time, and if the cache settings for a given type are to be changed at runtime, it is required that the cache be disabled first, then enabled again with the new settings. This will allow the cache to be completely torn down, and reinitialized, allowing the entire type of cache to be changed without application restart.

Options (what is visible will vary depending on what type of cache is selected):

- **Enable query caching**: Check to enable caching–if not checked, no cache rules will trigger caching, although rule processing will be done as normal.
- **PCI/HIPAA Cache**: When enabled, no cached objects will actually be transmitted to the grid-cache, but the cache will be used to message invalidation requests. This is to avoid transmitting possible sensitive data across unencrypted connections, while maintaining cache synchronization integrity. This option is mutually exclusive with disabling the grid cache offload function, and will remove the benefit of the cache in cold-start scenarios.
- **Auto-tune Cache**: Enabling this option will enable the cache logic to disable caching in situations where it doesn’t make sense from a performance perspective. This includes if a query pattern isn’t providing any cache hits, or the benefit of the hits that do occur isn’t sufficient to justify caching.
- **Cache Type**: Select the desired cache type. Options will adjust based on the cache type selected.
- **Server**: For external grid cache interfaces, specifies the server name or IP to connect to.
- **Port**: When using an external grid cache, specifies the TCP port to connect to.
- **Database ID Number**: For Redis, which database "number" is used, not supported when the Redis instance is in cluster protocol mode.
- **Cache Password**: For interfaces with a password authenticated interface, the password to connect to the cache with.
- **Use SSL**: When connecting to the cache, should SSL be used for the connection (Redis)
- **Verify Peer**: When using SSL, verify the peer TLS certificate
- **API Cache Name**: For interfaces that utilize a “named” cache interface, this is the name of the cache to use, i.e. Hazelcast.

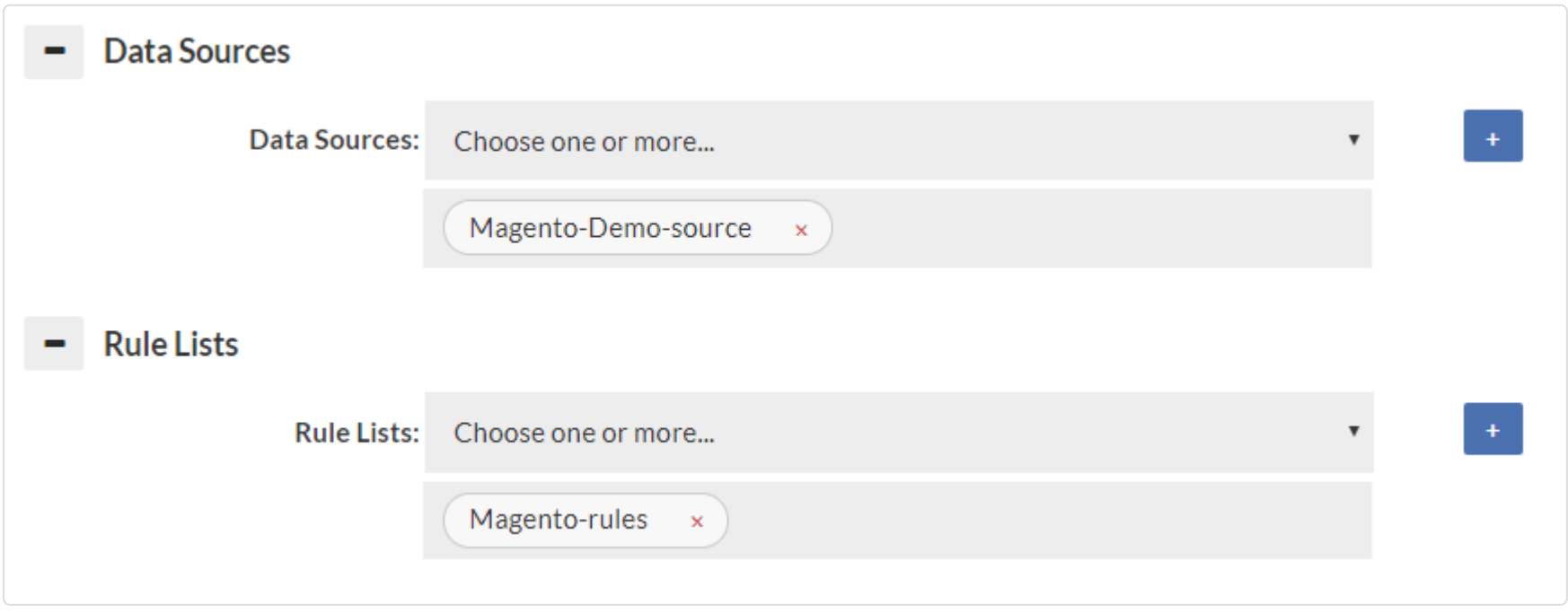
- **Cache Configuration File**: When supported, this is an external configuration file to configure the cache, i.e. a Hazelcast client or server XML configuration file. The location of the file should be relative to the current working directory of the JDBC driver or proxy, which will be printed on startup. On a proxy install, when this file is not present, it will be pulled from the Heimdall Server.
- **Grid Cache Offload**: This option, if enabled, enables a first-tier local cache, which allows hot content to be served out of local heap. In the local cache, the objects will be stored in the performance optimal format (binary streaming format for a proxy, and Java objects when not a proxy).
- **Duplicate request tracking**: Track the query hash in a temporary list with the same TTL as a cache itself, and only cache if we have a "hit" in this temporary list. This prevents caching of unique queries that will never return from cache.
- **Max Expiry**: If set to a value above zero, this value (in minutes) is the maximum TTL the L1 cache will cache an object for. This overrides the TTL in the cache rules if it is lower than the TTL set there FOR L1 cache queries. This value does not impact the TTL in the L2 cache such as Redis.
- **Max Cached Objects**: If set to a value other than 0, this limits the total number of objects in the L1 cache layer to a fixed maximum. This can assist in reducing memory garbage collection time under heavy load, and is advised under very high request volume.
- **Max Cached Size**: In proxy mode, this setting controls how much of a result-set's transmit buffer is saved before caching is aborted. Any result-set larger than this will not be cached.
- **Customize Query Cache Key**: This allows the key used to access cached objects to be customized in order to expand when the result is considered the same. By default, all queries will include a hash of the complete SQL query, but also includes the VDB name the request was made through, the database user, catalog and schema. These fields can be removed to allow, for example, multiple VDBs to share the same data in the cache.
- **Cluster manager via cache**: This option allows a VDB to be managed by multiple management nodes at once, via the cache interface. For this to work, all management nodes must be populated with the vdb, and the cache settings configured to be identical. Once done, configuration changes for that VDB will be synchronized between nodes, and the cache will be used to propagate metrics as well, allowing the dashboard to be used at the same time on both nodes. Further, if one management node goes offline, the access layer nodes will remain unaffected.
- **AWS Access Key, Secret Key, Tag, Tag Value**: When using Hazelcast with AWS Auto-discovery, these options may be necessary for proper discovery of other Hazelcast nodes.

Note: With Amazon’s AWS Elasticache service, the Redis parameter group option of “notify-keyspace-events” should be set to the value of “AE” in order to optimize cache behavior. This will also be instructed in the log output. For non-Elasticache Redis servers, this option will be configured automatically.

Note 2: Memory allocation for cache is dynamically controlled based on free space allocated to heap. Use the VDB setting of "xmx" to adjust this (in vdb properties). It defaults to target about 1GB of total memory used by the proxy process.

Test cache: The "Test Cache" button allows the user to verify whether a cache connection can be established. Additionally, it checks if key tracking is enabled. If key tracking is not enabled, an alert message will be sent to inform the user about it. After reloading the configuration, make sure to wait for a minimum of 10 seconds as the cache will need to restart. Failing to wait can lead to incorrect test results.

Data Source & Rule List

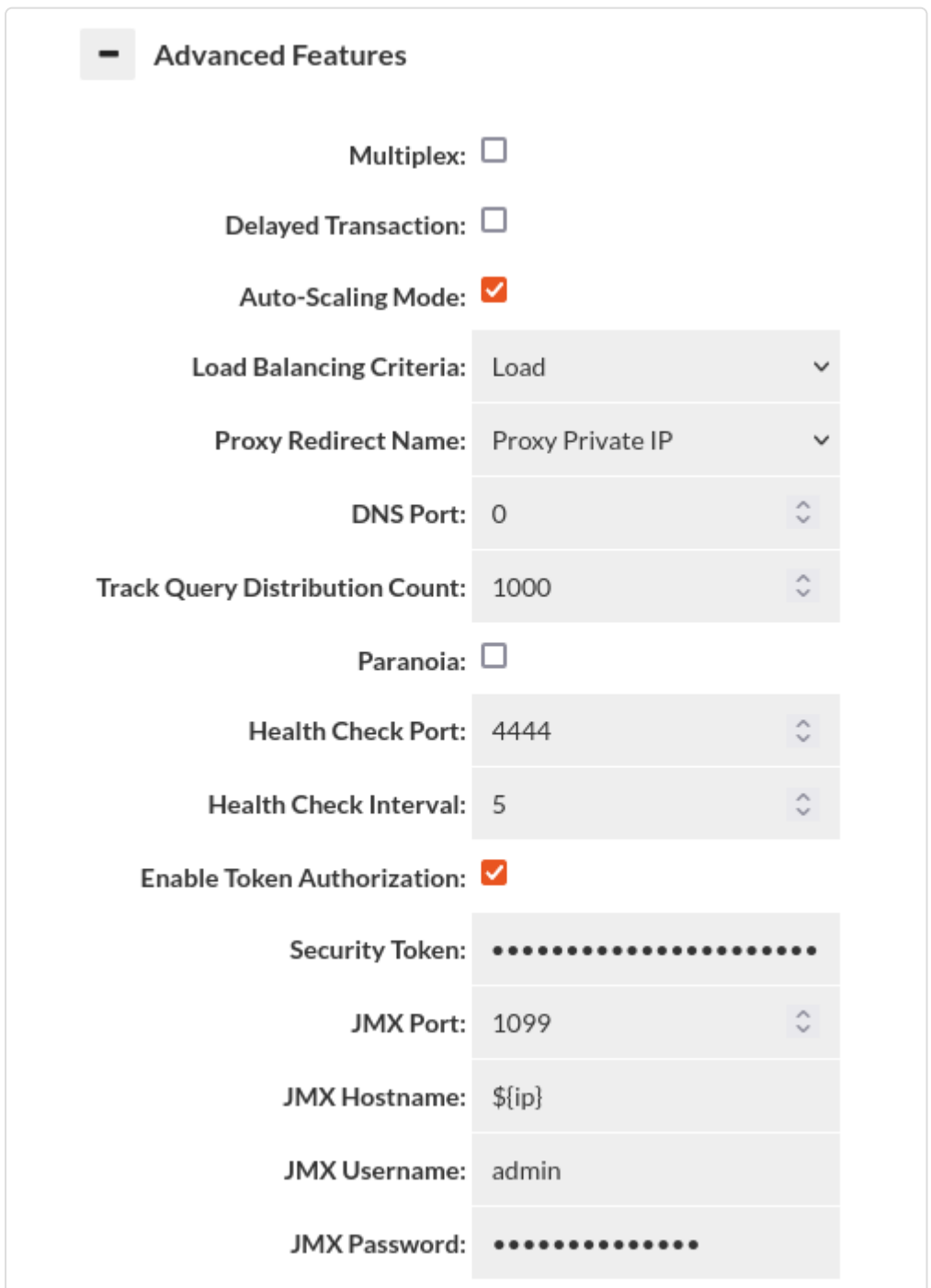


Specify at least one data source, as a default for data to be retrieved from. If a forward policy is specified in the rules for a vdb, it also must be selected here to insure proper connectivity is established to that data source for the forward function to work properly. A connection to the data source will only be established when used if not the primary data source. A reasonable attempt will be made to insure that the data sources for forwarding and read/write split are automatically populated here, but in some rare corner cases (with dynamically generated properties), all data sources will need to be specified here.

Like the data source, the rule list selector configures what rules should be attached to the vdb. If empty, no rule list will be used. All rule lists that are used by the initial (default) rule-list must be specified or they will not be executed. A reasonable attempt will be made to insure that the rules used in "call" actions are automatically populated here, but in some rare corner cases (with dynamically generated properties), all rules will need to be specified here. The log may generate warnings if this is not setup properly.

Note: Only the first rule-list specified will be executed by default. In order for additional rule-lists to be executed, they must be referenced from a rule with an action of call.

Advanced Features



- **multiplex**: Do load balancing to the server at the transaction level vs. the connection level. This can drastically reduce the number of connections being established to the server. Requires connection pooling to avoid a dramatic performance drop, and may have side-effects that break certain apps. Test carefully before using in production. DelayedTransactions and multiplexing can be used together to improve the query distribution for apps that support them. Examples of features that if used can cause application breakage with multiplexing include Postgres's search_path (then user must execute a query with the full path to the table name), and MySQL session variables. Any "state" will effectively be lost between queries that are not in a transaction when this feature is used. With this option, intelligence is applied to pool in the most effective mode, similar to how PGBouncer provides session, transaction and statement level pooling. Unlike PGBouncer however, you don't need to choose one mode, as it will intelligently switch between modes depending on the situation. As an example, if we

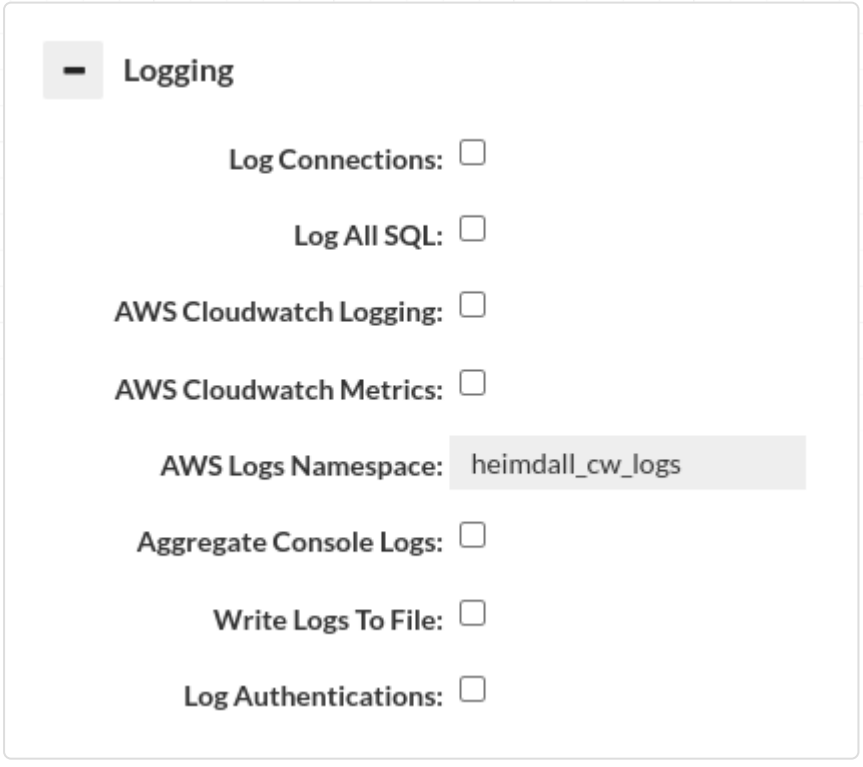
determine a temporary table is created, it will switch to session level pooling until the table is released, then switch back to statement level pooling. If a prepared statement is created, this will also happen automatically. This provides the best pooling behavior without additional configuration on the user's side.
- **multiplex timeout**: When multiplexing how many milliseconds should occur before releasing the connection on the back-end. In general, it is safe to leave this at 0.
- **delayedTransaction**: (proxy only) Delay when a transaction is started until a DML is detected. This is useful when using a framework that does everything in transactions, and the "in-trans" option in a rule isn't useful. Rules set to not match in a transaction will operate until a DML is encountered, then will not match until a commit/rollback. There may be other performance benefits on the server side as well depending on server. Test carefully before production use.
- **Auto-Scaling Mode**: (proxy only) Enable to activate auto load-balancing features of Heimdall proxies, for use in multi-proxy deployments. This includes DNS and HTTP based redirects. This option requires external cache (Redis or Hazelcast) configured for VDB in order to work. See below for the options relevant for this feature.
- **trackQueryDistributionCount**: a count (Long) of the number of query entries to track for auto cache refreshing (default of 1000, requires autoRefresh cache option). Rarely used.
- **paranoia**=Enable paranoid logging, don't reveal the SQL patterns or queries, only their hashes in debug logs
- **healthcheck-port**: If the proxy should do a self-check on itself, and expose the status via the designated port. With an HTTP connection, request /status to determine the status of the proxy. It can be used with auto-scaling set to active/standby mode to redirect all /status requests to active node allowing external load balancers to route traffic only to one proxy and keep other as backup.
- **healthcheck-interval**: When using the healtcheck-port option, specify how often the proxy should issue a health-check on itself.
- **Enable token authorization**: If enabled user that wants to perform api base invalidation will have to provide correct authorization token in request params.
- **security-token**: Security token should be added to request as parameter for example token="exampleToken".
- **JMX Port**: If JMX monitoring is desired, set the port here. Typically 1099 if used.
- **JMX Hostname**: The hostname that JMX uses to refer to itself--the monitoring clients must resolve this name to the Heimdall instance. The value of ${hostname} can be used to map to the internally detected hostname, and can be combined with other string values, i.e. "${hostname}-proxy" could be used to provide proper resolution on a remote client that doesn't have access to the actual hostname via dns.
- **JMX Username**: The username the JMX client will use to connect to the proxy
- **JMX Password**: The password for JMX connectivity

Auto-scaling Mode

When enabled, the following Advanced options become available:

- **Load Balancing Criteria**: For redirection multiple criteria can be used, the most common of which will be by load. This will select the lowest loaded (by cpu) proxy to receive traffic. Other options include random and active/standby. DNS based load balancing supports all of these options while health check (HTTP) works only with active/standby mode, which redirects all traffic to single node and keeps other ones as backup in case primary one fails.
- **Proxy Redirect Name**: When redirecting, we can choose to either redirect by the proxy's IP address or by hostname. In the case of hostname, this must resolve to all clients connecting as-is, which often won't be the case. In general, the IP address should be used.
- **DNS Port**: If set to anything other than 0, each proxy will listen for DNS queries on the specified port, and will answer **ANY** DNS query with a list of proxies to connect to (one in case of active/standby mode), sorted in order based on selected mode (random by default). This can be used in conjunction with the client redirect to remove NLB completely from the database protocol path, yet still provide reliable load balancing. In general, this should be set to 54 to ensure that there is no conflict with any Systemd listener on the host.

Logging

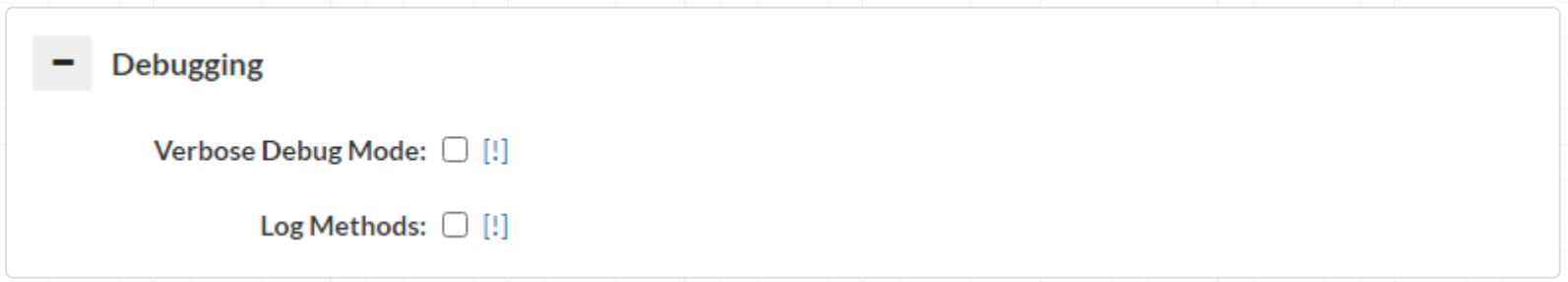


The following log options are available:

- **Log Connections**: to log when a JDBC or TCP connection is established by the calling application
- **Log All SQL**: equivalent to enabling a LOG policy with a .* wildcard. If this is enabled, it will override any log rate control configured on a log policy.
- **AWS CloudWatch Logging**: If in AWS, and selected, then logs generated by VDB instance will be periodically sent to the CloudWatch as a log stream.
- **AWS CloudWatch Metrics**: If in AWS, and selected, then multiple metric points will be generated under the Heimdall category for the VDB and AWS instance to allow monitoring of the access layer performance. Note, these metrics are generated on a per-minute basis, so may add to the cost of CloudWatch. Additionally, console logs will be shipped to CloudWatch as well when selected.
- **AWS Logs Namespace**: If in AWS, a name provided as that field will be used as a log group name AND metrics namespace in CloudWatch.
- **Aggregate Console Logs**: In order to simplify debugging, this option allows console logs from the access layers to be sent to the central manager for logging. When using the "Local proxy" option, this option will be implicitly configured for the management server managed proxy instance, even if not set for other proxies.
- **Write Logs to Files**: When performing SQL logging, the logs are by default written to an internal database for optimal Analytics speed. If the logs should also be written to detailed CSV logs for each query, please select this option. This may result in a large amount of data written to the log directory, although logs will automatically be cleaned up when that filesystem reached 90% full.
- **Log Authentications**: Each authentication attempt will be logged. We will get information about whether authentication succeeded for a given user with the currently selected authentication mode. You can read more about authentication here Theory->Proxy Authentication.

- **Important**: When logging SQL, a large amount of data may be generated, and logging can impact the overall performance. To avoid this, you can disable sql logging here, and instead use a log policy with parameters that limit how much data is written. Please see the rules section for details on parameters and the logging section for more details on logging overall.

Debugging



- **Pass-through Enabled**: When in JDBC mode, the system can be set to pass any **NEW** connections through to the underlying driver, bypassing all processing logic. This should only be used in rare situations to debug if the system is causing a problem directly, or the existence of the system is causing a problem.
- **Verbose Debug Mode**: In order to diagnose issues with rule processing or other behavior anomalies, this option can be set in order to track the processing of a query through the access layer. This option can cause significantly performance penalties and should be used with caution.
- **Lite Debug Mode**: Works similar to verbose debug mode, doesn't log data all the time, instead keeps it in a buffer for up to a second and dumps all records on an exception.
- **Log Methods**: log all JDBC method calls made by an application, excluding those relating to resultSets. Warning!
- **Log ResultSet Methods**: Include ResultSet operations when logging other methods. Warning!

Important: When adding in method logging, for every query generated, it may result in a dozen records for JDBC methods, and with resultset methods, every row will likely have one or more log records. Caution should be observed when using these two log options are used, and should in general only be used in low-volume lab conditions.

VDB Properties

Allows a list of name-value pairs to be used to configure various options, in general per direction from customer support, and for use with test releases to enable a fix. Example:

- **suppressNoResult** - With Postgres, if an update query is executed via the executeQuery result, it will generate an exception on return saying "No results were returned by the query". In some frameworks, this is detected and suppressed when using the native Postgres driver, but not with the Heimdall driver. In order to work around this behavior, this option can trigger us returning a null instead of a resultset, which appears to allow the calling code to work fine. (true/false)

- **dnsCacheTTL** - The cache time (in s) to use for the dns resolution cache, defaults to 5s to conform with AWS requirements. Note: This will impact the global JVM setting for this if used in JDBC mode.

- **dnsNegativeTTL** - The time to cache negative DNS (non-existent) queries, defaults to 10s per Java standard. Note: This will impact the global JVM setting for this if used in JDBC mode.

- **delayCacheInit** - The time in seconds to delay initializing the cache, normally not needed.

- **reparsePrepared** - In cases where prepared statements include dynamic variables, this triggers the reparsing of the expanded query for the purposes of isolating it's base query pattern. This can resolve issues when patterns overflow the proxy or driver memory, or analytics show the pattern with variables included. (true/false)

- **rejectPrepared** - As many features end up disabled with prepared statements, it may be needed to reject prepared statements in a QA environment to ensure they don't creep into production code. This will result in an SQL exception if they are attempted. (true/false)

- **syncInterval** - Synchronization interval (in ms) which determine after what time the synchronization query will be privileged to be executed after reconnection. Default value = 300000ms.

Proxy only

These properties are applicable only to VDB working in Proxy mode.

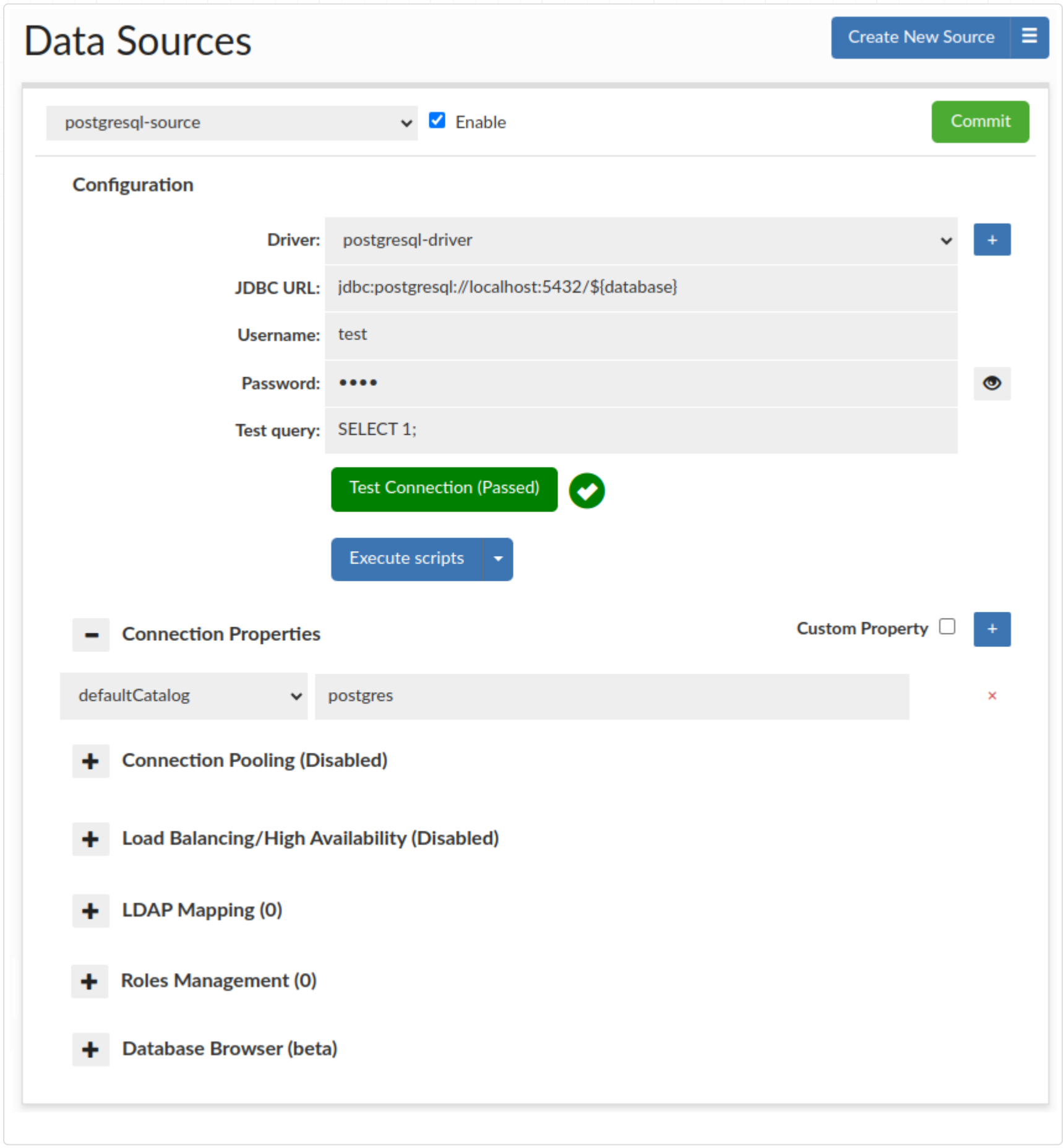
- **connectionIdleTimeout** - Connection timeout (in ms) for an idle client-side connection, will terminate connection's thread (requires a healthcheck port and interval to function).

- **queryTimeout** - Timeout (in ms) for a connection executing a query, will terminate connection's thread (requires a healthcheck port and interval to function).

- **maxConnectionThreads** - Max amount of connection threads allowed. Exceeding the limit will terminate the connections that were idle for the longest time (requires connectionsCleanupPercentage property to function).
- **connectionsCleanupPercentage** - Percentage of connection threads that will be terminated after maxConnectionThreads limit is exceeded. The excess connection is not included in calculation, for example when limit of 10 connections is exceeded (we get 11th connection), the amount to be deleted is calculated from 10. Setting this property to 10 means 10%.
- **desiredCipherSuite** - Forces defined cipherSuite to be used, for TLS connections. All other than defined ciphers will be disabled, use it with caution. Example values: TLS_AES_256_GCM_SHA384, TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- **extractTlsSecrets** - Extract the shared secrets from secure TLS connections for use with Wireshark. Secrets are stored in log folder. To apply this property, Proxy restart is required and Verbose Debug Mode on. Boolean value, default value = false. It is also recommended to enable "Close connections" checkbox in packets capture configuration to close connections so that every connection has secrets extracted correctly for packet decryption. (true/false)
- **enableBlacklist** - Enable adding address to blacklist after many failed database authentication attempts. (true/false)
- **connectionIdleTimeout** - Connection timeout (in ms) for an idle client-side connection, will terminate connection's thread (requires a healthcheck port and interval to function).
- **queryTimeout** - Timeout (in ms) for an connection executing a query, will terminate connection's thread (requires a healthcheck port and interval to function).
- **defaultTimeZone** - Default Time Zone for proxy. Example values: Europe/Warsaw, America/New_York. (Only PostgreSQL)

Note When using the healthcheck-port option, an additional feature is enabled, that of API based invalidation. Please see the Cache theory section for more details.

Data Source Overview



A data source represents access via JDBC to a back-end database server. Each VDB definition must have at least one default data source associated with it, which will be used if no policy dictates what database should receive a query. Source configuration includes the connection properties, connection pooling, and Load balancing and cluster management characteristics.

The **driver** field allows either a new or existing JDBC driver to be specified. Heimdall comes with a small library of JDBC drivers that it has been tested with, but other drivers can be configured in the driver tab as needed. When using the proxy functionality of a VDB, the tested drivers should in general be used, as some internal and version specific features of a driver are often used by the proxy to provide compatibility with the binary layer translation that occurs.

The **JDBC URL** configuration will normally be build through the configuration wizard, or can be copied from an existing JDBC application. Properties on the JDBC URL can be moved to the name-value list in the connection properties, OR can remain in the JDBC URL–either works, although using the name-value list provides an easier to view list vs. a long URL. When internally doing database type detection, the JDBC url is also used to determine what type of database is in place, for per-database behavior adjustments. When using load balancing, this URL is not in general used, with one

major exception--when using the hdUsePgNotify option, this will be the target used for notifications. This is necessary as Postgres notifications will not be shared between nodes, and as such this should point to a single name that resolves to a single target database.

The data source **username** and **password** are used for several purposes:

1. For health checking the data source;
2. When performing lag detection, to write to and read from the Heimdall schema;
3. When doing cluster detection, reading cluster variables as appropriate, or when Heimdall needs to extract metadata for other purposes;
4. When no user or password is specified in JDBC mode, this user will be used to connect to the database;
5. In proxy mode, if proxy authentication is not provided. In this case, any user can connect, but the DB connection will be made with these credentials.

In some cases, it is necessary to specify various parameters in a dynamic way, i.e. with the Odoo application, it changes the active database at runtime, and connects to the desired database as desired. To support this, fill in the following values as necessary:

- ${catalog} or ${database} to specify the name of the database (either will work)
- ${schema} for the specified schema
- ${user} for the connection user
- ${password} for the password specified on connect (only for Postgres and SQL Server)

The **test query** option is used in a variety of situations, including health checks, and verifying that a connection is still viable after an exception. It is also used in a variety of conditions when pooling is configured, in order to validate the connection while idle, etc.

To validate that the options appear valid, please use the **Test Connection** button. This will initiate a connection through the vendor driver by the Heimdall management server, and acts as a general guide if the configuration appears valid. It is not a guarantee that under all conditions it is valid, however.

To automatically execute a set of SQL scripts on the data source, use the **Execute scripts** button. This will attempt to execute selected scripts located in config/scripts in the Heimdall installation directory. Scripts are grouped by database type into directories. You can modify existing ones or add your own to these directories to be able to execute them via GUI. Predefined files are put into place on Heimdall startup, these use the "heimdall" schema by default, and you can modify them, but if you choose to use a different schema, it's existence must be ensured.

Connection properties

Connection properties are database and driver specific. When in proxy mode, a default set of properties will be inherited by the source in order to provide compatibility with the proxy mode used. In JDBC cases, no default properties will be set by Heimdall. When using a known driver, all supported connection properties will be provided in a drop-down list, and tool-tip help provided for each one when selected. When using an unknown driver, the property name field will allow an arbitrary name to be specified and value provided, as per your JDBC driver's documentation.

In order to add a custom property (in the event it isn't in the drop-down list), you can use the custom property checkbox next to the blue plus, to provide a free-form field.

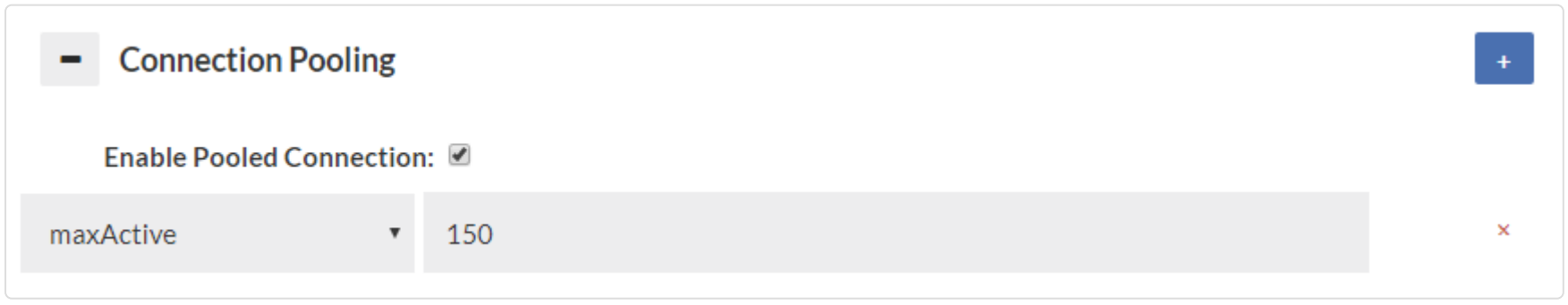
Special properties:

- **dbTriggerTracking, dbTimeMs and dbChanges**: Please see the Cache Theory section.
- **initSQLAsync**: Please see the Async Theory section.
- **hdConnectRetryCount**: Specifies the number of times a connection attempt will be made, defaults to 3.

- **hdConnectRetryDelay**: The length of time between retry attempts. This value doubles on every retry for a given connection. Defaults to 0 (ms).
- **hdConnectTimeout**: How long each attempt to connect will be made in ms. This option sets database driver specific values based on what is supported. Defaults to 1000(ms).
- **hdUsePgNotify**: Instead of using a cache's pub/sub interface, when using Postgres, use the postgres notify interface for cache invalidation and read/write split last write time notifications. Please note, this still requires a cache to be enabled, although it can be the "local only" cache option.
- **hdNotifyDBs**: When using hdUsePgNotify, by default only the default catalog is listened to. This option allows one or more catalogs to be listened to, for trigger based invalidation. A value of * means "all catalogs", with comma separated values starting with "-" removing from the complete list. Example: *,-template0,-template1 would be a generally good starting point.
- **azureDbHost**: Please see the [Azure] (../environment/azure.md) documentation for more details.
- **dbTriggerTracking, dbChanges, dbTimeMs**: Options to enable query based table change tracking, leveraging triggers. See the caching page for more details.

Please see the UI for a more complete list, including vendor options, as the drop-down list provides context sensitive help on each option provided.

Connection Pooling



The pooling section controls if connection pooling (the reuse of back-end connections for many front-side connections) is used internally to Heimdall, and if so, what the connection pooling properties are. Heimdall implements a variant of the Tomcat connection pool, and where appropriate uses the same properties for configuration.

The primary difference between the Heimdall and Tomcat connection pool is that the Heimdall pool implements it as a "pool of pools" where each user/database (catalog) combination becomes a pool within the connection pool, but overall resources are constrained by the overall maxActive option, which dictates the overall number of busy AND idle connections that can be established to the data source. Additional properties of maxUserActive and maxUserIdle dictate how many active (busy+idle) and idle connections are allowed on a per-catalog+user combination. This provides much more granular control over the behavior of the connection pool when many databases and users are connecting to the database server.

If the database has configured a type of idleTimeout (currently supported: idleSessionTimeout for postgres(version 14+), wait_timeout for mysql) if the value is detected sessions in idle pool will be deleted with an increasing chance in time based on idleTimeout, it is done to prevent having closed connections in the idle pool. If connections are not deleted fast enough consider changing the timeBetweenEvictionRunsMillis pool property.

The properties section provides a drop-down list of the available properties, and tool-tip help for them. The available properties are:

Important queries--in general these should be set in all cases:

- **maxAge**: The chances of a connection being closed when being returned to the pool (1 out of X) or during an eviction run if idle. Old behavior was the Time in milliseconds to expire the connection on return or idle (Long, 20). connections less than the timeBetweenEvictionRunsMillis will not be culled at random, so if there is activity on the connection at least that often, it will remain connected.
- **maxActive**: The maximum number of open connections (Integer, no default, however an attempt is made to detect the value from the server). This is equivalent to the connection limit in PGBouncer's max_client_conn option. If not set, the proxy will attempt to determine this from the database itself, but if more than proxy is active, this should be manually set to account for the maximum number of proxies, as it is applied on a per-proxy basis.
- **maxUserActive**: The maximum number of open connections for a particular user, can be overridden with a rule (Integer, 0)

- **maxUserIdle**: The max number of idle connections for a particular user, can be overridden with a rule (Integer, 0)
- **maxWait**: The time to wait in milliseconds before a SQLException is thrown when a connection is requested, but can't be honored due to the number of connections (Integer, 30000)
- **resetQuery**: The query to run after a connection is returned to the pool (String, no default)
- **testOnReturn**: True if validation happens when a connection is returned, implied with a reset query (Boolean, false)
- **validationQuery**: The query to run during validation (String, select 1)
- **rateLimit**: Total connections/ms that can be opened, (double, 2)
- **maxBurst**: Number of ms of burst that can be allowed before clamping occurs (double, 50)

Other properties:

- **abandonWhenPercentageFull**: Connections that have been abandoned isn't closed unless connections in use are above this percentage (Integer)
- **commitOnReturn**: If autoCommit!=null and autocommit is not set then the pool can complete the transaction by calling commit on the connection as it is returned to the pool (Boolean, false)
- **rollbackOnReturn**: If autoCommit!=null and autocommit is not set then the pool can terminate the transaction by calling rollback on the connection as it is returned to the pool (Boolean, true)
- **connectionProperties**: The connection properties that will be set for new connections. Format of the string will be [propertyName=property;]* (String)
- **defaultAutoCommit**: The default auto-commit state of connections created by this pool. If not set, default is JDBC driver default (If set to "null" then the setAutoCommit method will not be called.) (boolean, true). When the connection is returned to the pool, this state will be reset if necessary.
- **defaultCatalog**: (String) The default catalog of connections created by this pool.
- **defaultReadOnly**: (boolean) The default read-only state of connections created by this pool. If not set then the setReadOnly method will not be called. (Some drivers don't support read only mode, ex: Informix)
- **defaultTransactionIsolation**: (String) The default TransactionIsolation state of connections created by this pool. One of the following: NONE,READ_COMMITTED,READ_UNCOMMITTED,REPEATABLE_READ,SERIALIZABLE. If not set, the method will not be called and it defaults to the JDBC driver.
- **initSQL**: A SQL executed once per connection, when it is established (String)
- **logAbandoned**: If true, stack trace will be recorded and printed out for timed out connection (Boolean)
- **logValidationErrors**: Log errors during the validation phase to the log file (Boolean, false)
- **minEvictableIdleTimeMillis**: Minimum amount of time a connection stays idle before it is evicted (Integer, 60000)
- **removeAbandoned**: True if connection in use can be timed out (Boolean)
- **removeAbandonedTimeout**: Timeout in seconds for connections in use (Integer, 60)
- **suspectTimeout**: Timeout in seconds for connection that suspected to have been abandoned (Integer)
- **testOnBorrow**: True if validation happens when a connection is requested (Boolean)
- **testOnConnect**: Validate connection after connection has been established (Boolean)
- **testWhileIdle**: True if validation happens when a connection is not in use (idle) (Boolean)
- **timeBetweenEvictionRunsMillis**: Sleep time for background thread in between pool checks (Integer, 5000)
- **validationInterval**: If larger than zero than validation will only occur after the interval milliseconds has passed (Long, 3000)
- **validationQueryTimeout**: The timeout in seconds before a connection validation/reset queries fail (Integer)

For additional information about each pool parameter, please see the Tomcat pool attributes page (https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html#Attributes).

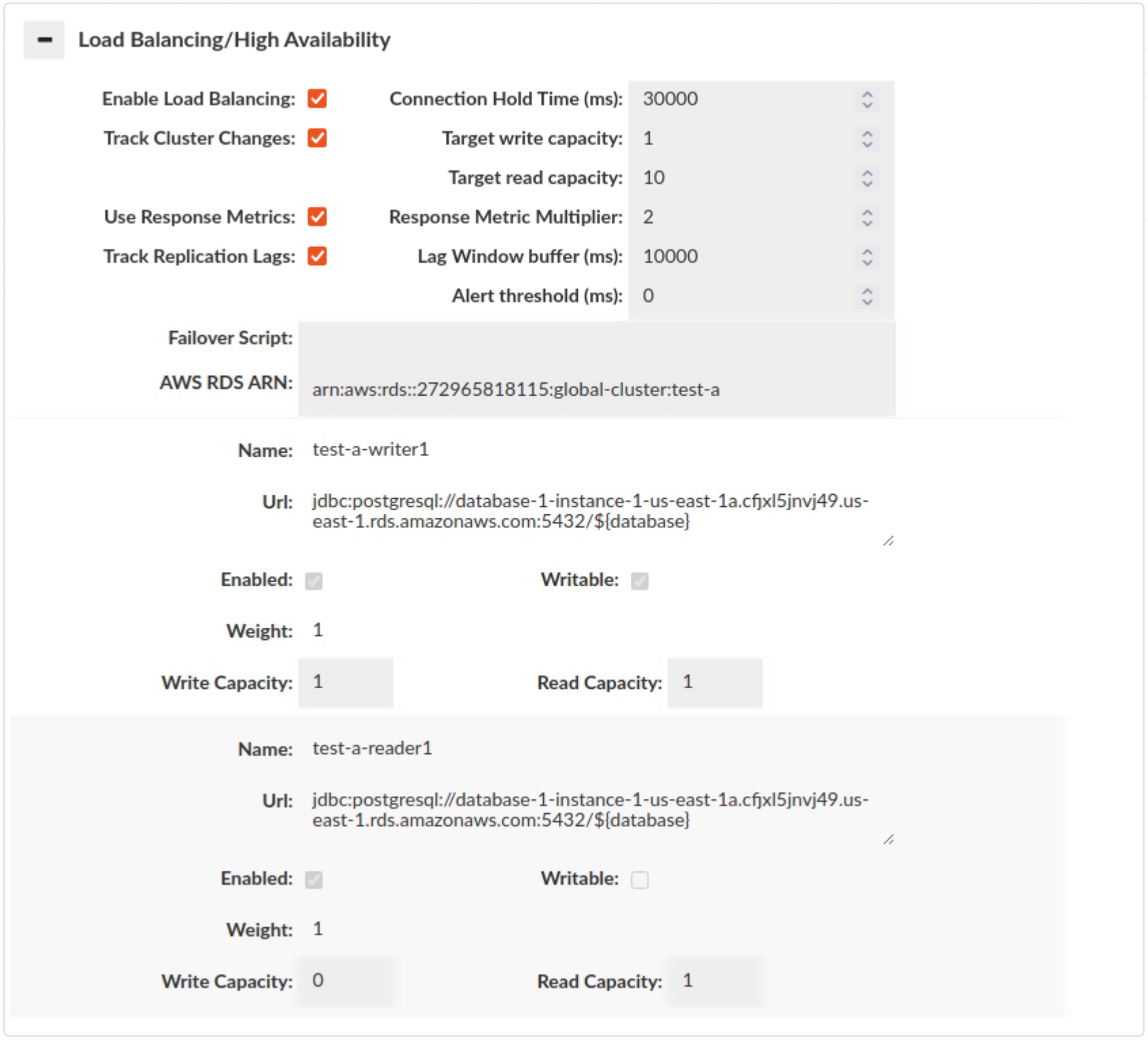
Warning: Not all applications will be compatible with connection pooling. If issues are observed in initial application testing, it is recommended that disabling connection pooling be performed in order to isolate if this feature is triggering undesirable behaviors in the application. If using the connection pooling, the default number of connections is set to 1000. To adjust this, change the “maxActive” setting to the desired number of connections.

Connection Pooling options

There are commands specific for connection pooling such as **show pools** and **clear pool** described in the Inline commands section.

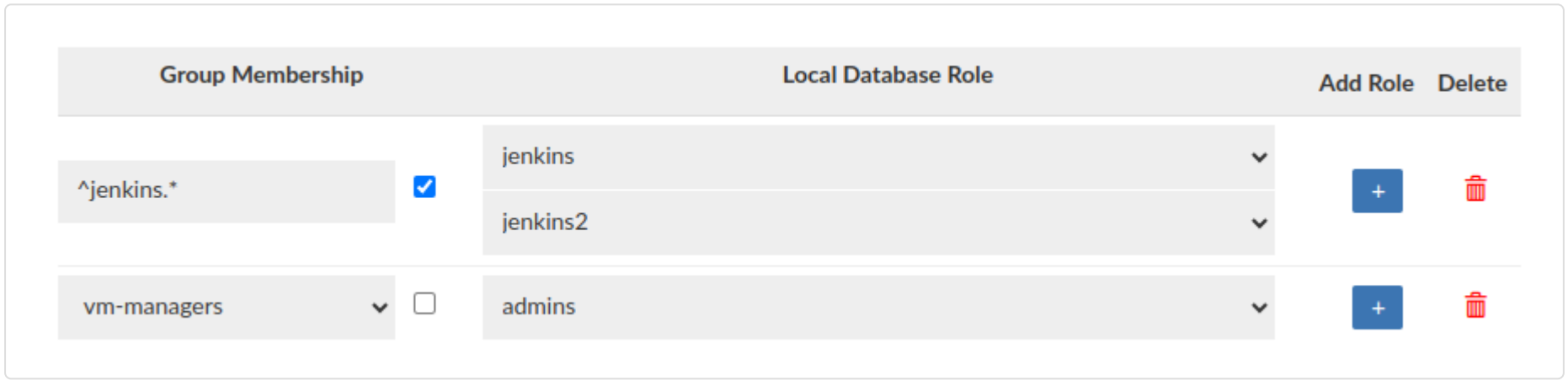
Postgres only: There is an option to clear a pool when making a connection to it is impossible due to the pool already being at maximum capacity. Please see Clear pool option.

Load balancing & High Availability



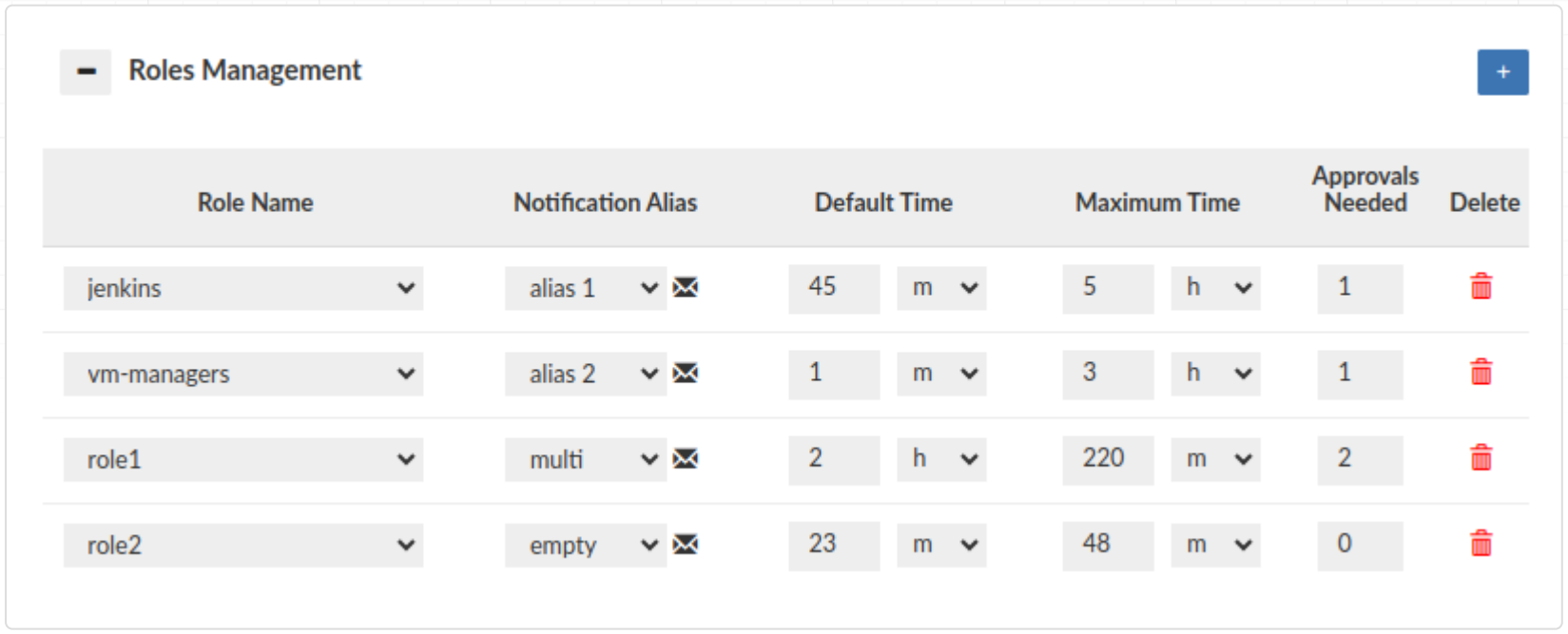
Due to the complexity of this topic and how the options interact with other subsystems, please see the help section dedicated to Load Balancing for more details.

Group Mapping



The Group mapping feature provides a flexible approach to associate a group (e.g. Ldap) with specific database roles. In the absence of group mapping, Heimdall defaults to mapping groups to roles with identical names. This option applies when the heimdall sync_user script is used with ldap authentication. Group mapping will also be triggered when user session is created via portal. Users can configure group extraction based on the group regex when the "Use Regex" checkbox is enabled, if checkbox is disabled, noone of the regexes will be applied to the mapping. When no mapping is configured, this option can be used as "deleting" group.

Roles Management



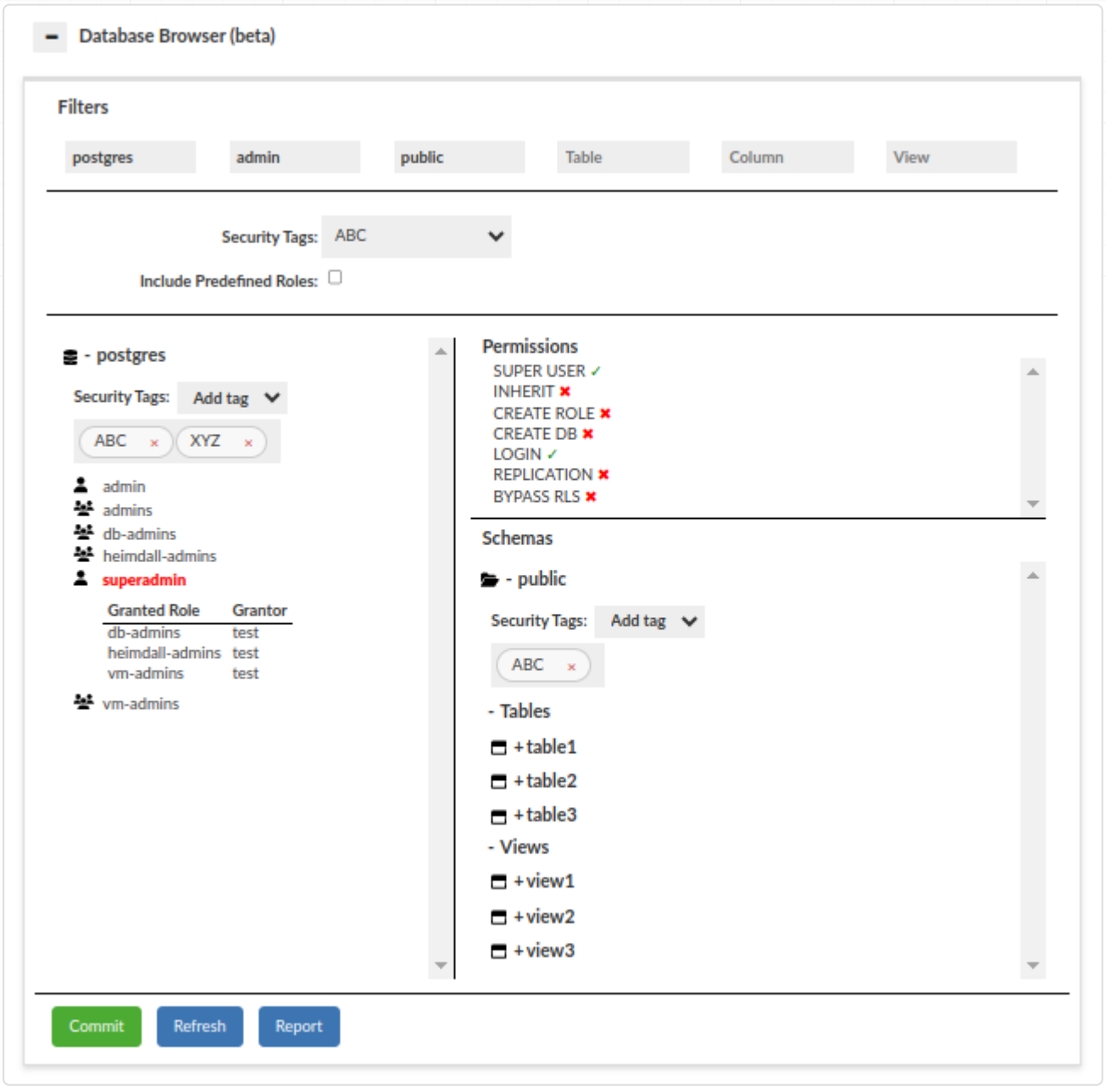
This section is used to manage roles in the database. In the configuration for a single entry with a role, you can specify a notification alias associated with an email group. This group will be informed when, for example, a session is requested for a particular role. Additionally, there are other options like the one to set the number of approvals needed to approve a specific role.

The entire configuration is described as follows:

- **Role Name:** Used to select a role from the database for which we want to create a configuration.
- **Notification Alias:** Here, you indicate the notification entry created in the Admin tab, which will be used to notify users associated with it when an action concerns the specified role. After clicking on the envelope icon next to it, you can view the list of emails associated with the specified notification.
- **Default Time:** Specifies the default session duration for the given role.
- **Maximum Time:** Determines the maximum time for which a particular role can be requested.
- **Approvals Needed:** Used to specify the number of approvals required for the requested role to be approved. The default value is 1. You cannot set a value higher than the number of emails associated with the specified notification alias. If no emails are associated with the notification, the value is set to 0, and it cannot be changed.

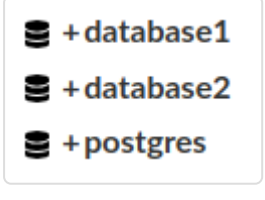
Warning: This feature is **NOT** functional when using MySQL with version < 8.

Database Browser (beta)

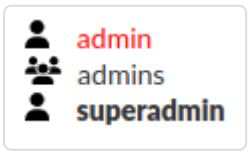


The Database Browser is a versatile tool designed for viewing database structures and modifying permissions. It operates on multiple levels to provide detailed access and control. We can distinguish the following sections:

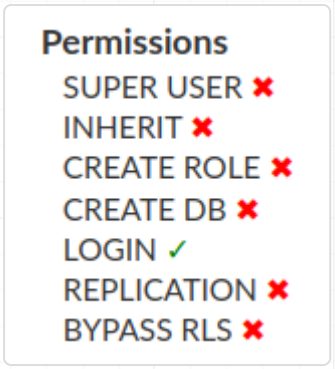
- **Databases:** The entry point where you select the database to work with.



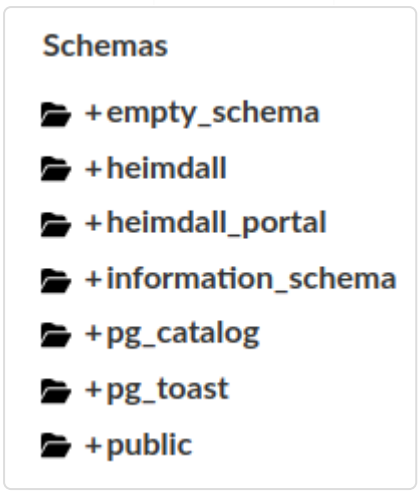
- **Roles:** You can select a specific role in the database to open the below sections. A distinction has been made between groups (roles that cannot log in), regular users (roles that can log in), and superusers (roles that can log in and have super permissions). When a role is selected, it will be highlighted in red.



- **Permissions:** You can view the permissions of a role, such as the ability to log in, superuser, etc.



- **Schemas:** Explore various schemas assigned to roles in the database.



- **Tables:** Access and manage tables, you can view and modify table and column permissions.

- **Views:** Access and manage views, a subset of tables, for tailored data representation. Similarly to tables, you can modify view and column permissions.

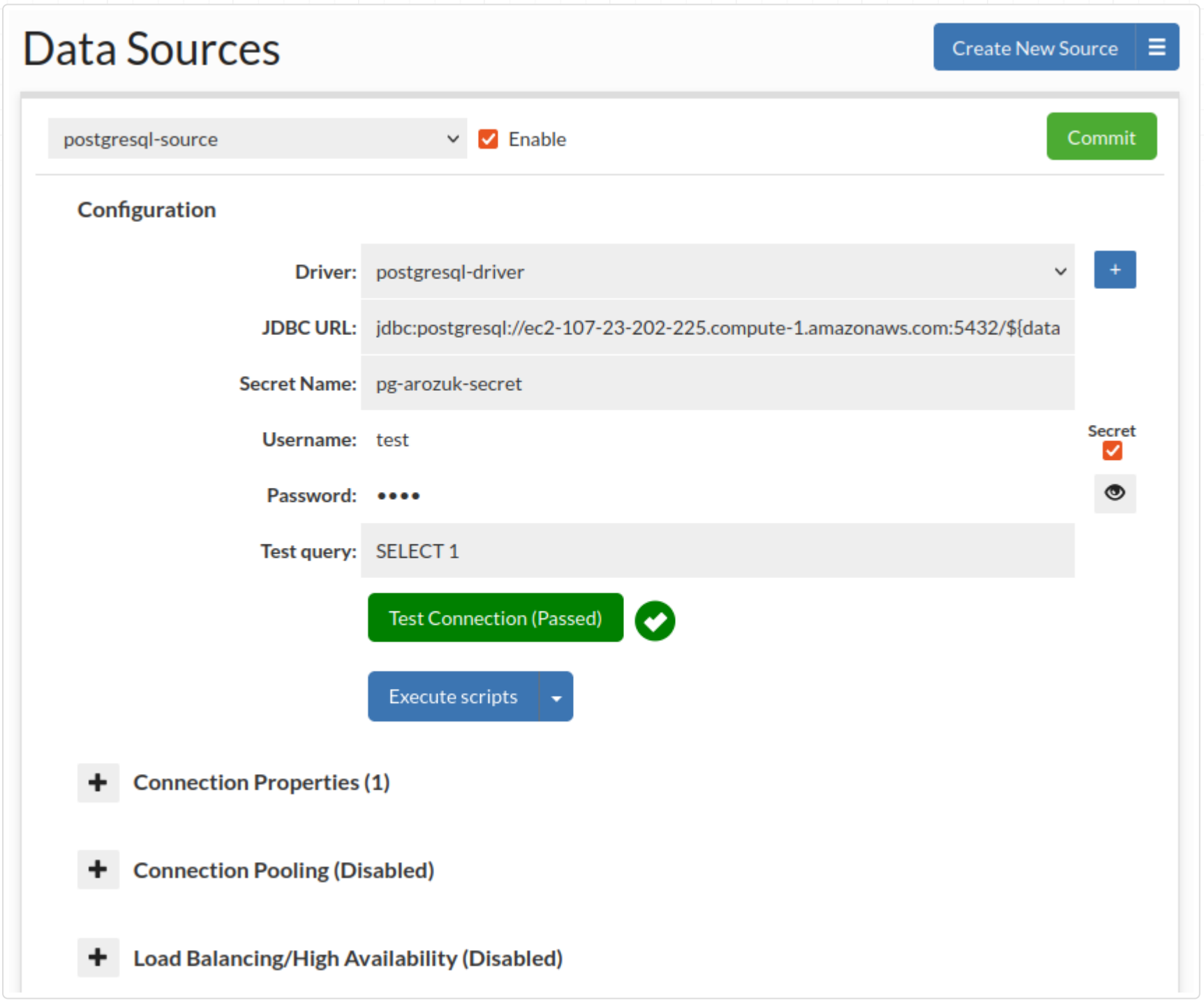
Depending on the type of database, the above description may vary slightly. For example, in MySQL there is no schema level, in SQLServer we have a distinction between database and server users. The above description represents the full functionality available for PostgreSQL databases, however, the idea remains the same.

By clicking the ✓ or ✖ near the permission and then click the **Commit** button you can save the updated permission.

By clicking the **Refresh** button you can fetch the newest data.

By clicking the **Report** button you can generate and download a report of security tags for the currently selected "Security Tags" filter.

AWS Essentials



When operating within the AWS environment, users will encounter an enhanced configuration option: the AWS Secrets Manager checkbox located adjacent to the username field. Enabling this option reveals an additional field for the Secret Name. Upon inputting the AWS Secret Name and subsequently clicking the Commit button, the system will automatically retrieve credentials such as the username and password from AWS Secrets Manager.

Furthermore, in instances where a health check is configured and fails, such as during password rotation, the system will attempt to retrieve updated credentials from AWS Secrets Manager. In the event of a proxy-induced change to the username and password, users are advised to refresh the DataSource page to view the newly uploaded credentials.

For those utilizing a custom secret manager, it is imperative to adhere to the structure outlined in AWS RDS secret manager. For instance, the JSON format should resemble the following: {"username":"test","password":"test"}.

Configuring Trigger Based Invalidation

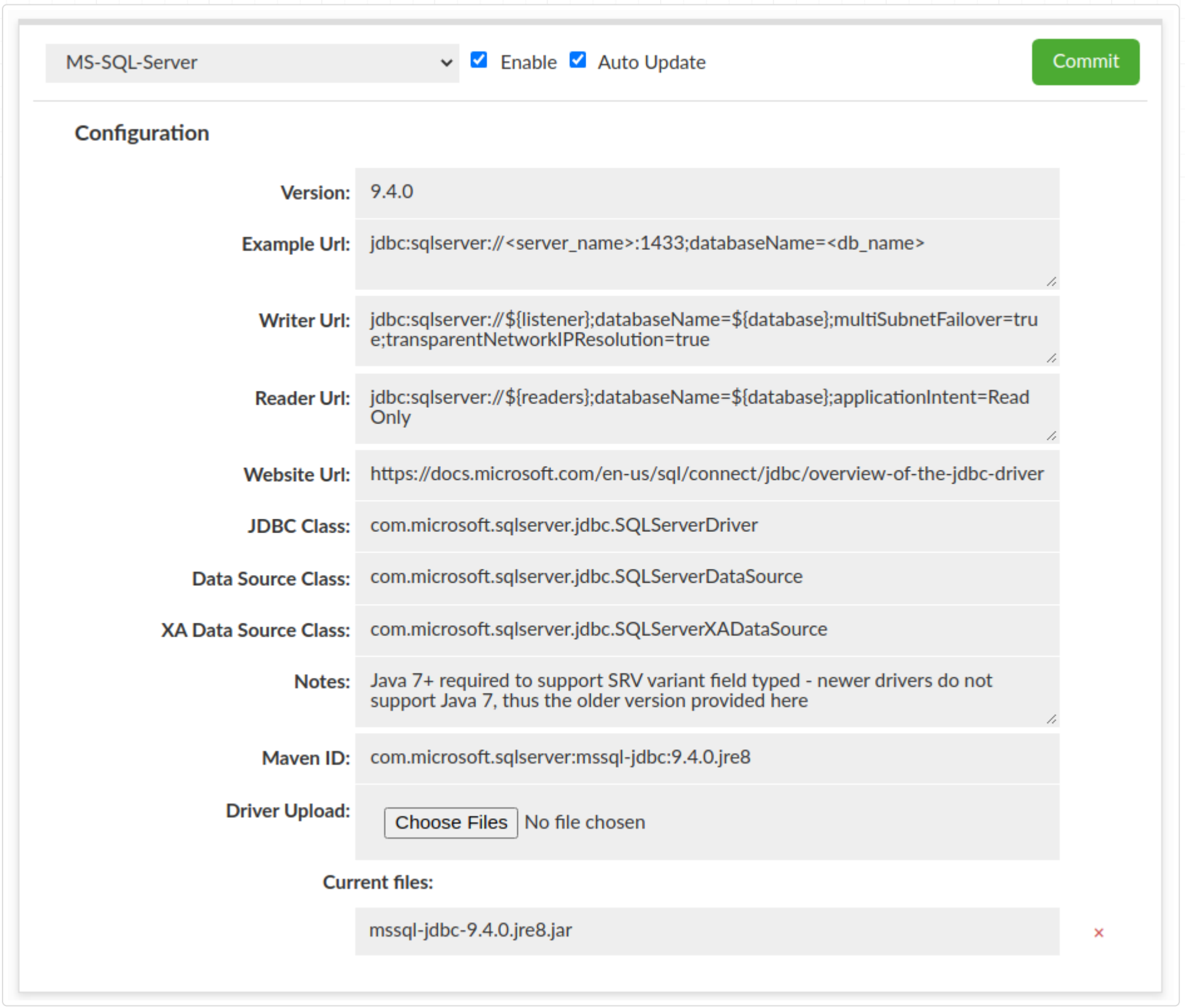
In order to support out-of-band data ingestion into the database, a data source can be configured for "Trigger based invalidation". There are several options available on the data source for this configuration:

- dbTriggerTracking: set to true if this feature is to be enabled
- dbTimeMs: A query to retrieve the current time in MS on the database, to insure timestamps are synchronized
- dbChanges: A query to retrieve a timestamp (as a long) and table name (as a string), for the last invalidation time of that table.

The dbTimeMs call for example defaults to the value of "SELECT heimdall.get_time_ms()", usable on MySQL without any changes. The dbChanges call defaults to {CALL heimdall.changes(@ts)}. This can be a stored procedure. The string "@ts" will be replaced automatically with the last modification (per the DB time) that has been observed, so can

be used to pull only the tables that have been modified since the last invalidation observed. The table name returned needs to be fully qualified, as observed with the "printTables" option for the rules, as this is the table name we will be comparing for invalidation purposes.

Driver Overview



The drivers tab allows you to manage the jdbc jar file that will be provided to the application or used by the proxy.

Fields

- **Version**: The version ID of the driver, for reference only
- **Example URL**: Used as a reference, not required
- **Writer URL**: Used in handling AWS RDS cluster tracking (see below)
- **Reader URL**: Used in handling AWS RDS cluster tracking (see below)
- **Website URL**: Used as a reference, not required
- **JDBC Class**: Required: provided by jdbc driver vendor
- **Data Source Class**: Optional: used in some situations, provided by JDBC driver vendor
- **XA Data Source Class**: Optional: used in some situations when using XA transactions, provided by JDBC driver vendor
- **Notes**: Optional text field for notes on the driver
- **Maven ID**: Optional field. Used to provide automatic download of jdbc jar file from maven central. Text should match a pattern: "groupId:artifactId:version" e.g. for mysql: mysql:mysql-connector-java:5.1.49. Jar file will be downloaded on driver creation and during update if not exists.
- **Driver Upload**: Upload your driver file(s) here. If a driver happens to be incompatible with our remote download scheme, this driver file can be left blank, as long as the driver is available on the server using the Heimdall Driver itself. This is a known issue with the AWS Redshift driver.

AWS Cluster Tracking

In order to account for various configurations that may be desired, if load balancing is enabled, and cluster tracking is enabled, the reader and writer URL patterns will be used to generate the actual JDBC URL's to represent the various nodes of the cluster. The reader and write URL's can contain the following variables for substitution, which should be used based on the information extracted from the AWS configuration:

- **${listener}**: For SQL Server, this will represent the listener URL, which points to the IP that tracks between the primary writer node during a failover. Empty for non-SQL Server cluster types
- **${writeEndpoint}**: For Aurora clusters, this will point to the primary endpoint hostname, which uses DNS to fail between primary nodes.
- **${readerEndpoint}**: For Aurora clusters, this will point to the reader endpoint hostname, which uses DNS over time to spray load across reader nodes. For a single node reader setup, this is appropriate, but in multi-node reader configurations, this can result in uneven distribution.
- **${writer}**: For any cluster type, this will point to the actual active writer. If more than one writer is detected, then multiple nodes will be created, one with each writer.
- **${writers}**: Like with writer, but will fill in all writer hostnames in a comma separated list. Useful for URL types that allow multiple hosts to be included in a single URL
- **${reader}**: Generates one node per reader, including a node for the writer, for the purpose of reading. The read weight of readers will be set to 10, but the read weight of the writer will be set to 1.
- **${readers}**: Generate one reader node, with all readers in a comma seperated list. The writer(s) will be first in the list.
- **${replica}**: Include all the readers as distinct servers, not including writer nodes
- **${replicas}**: Include all the readers in one URL, not including writer nodes

If the writer URL includes the string ":replication:" AND there is only one node in the cluster, then the value for the writer will be duplicated twice in the reader list. This is to avoid a degenerate case for the MySQL driver where a "replication" url requires two hosts to be defined, or it will fail to connect to any host. By duplicating the host twice, it passes the check, and will work as expected.

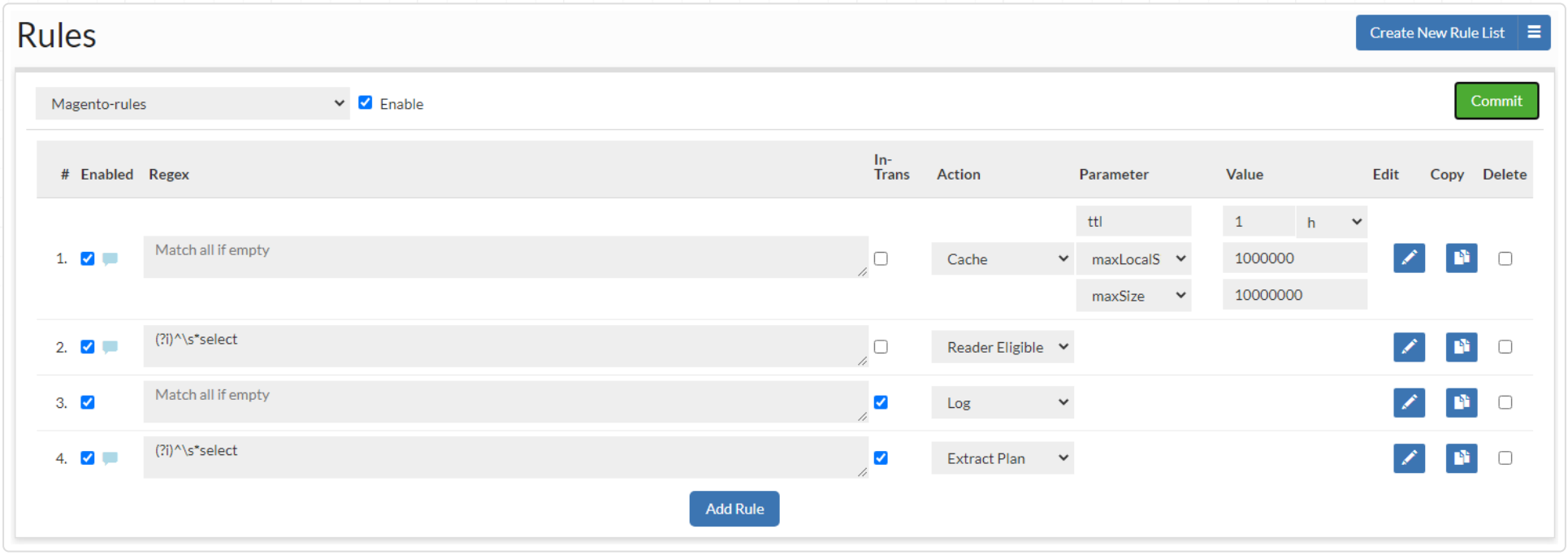
Advanced

When a vendor driver is used by the Heimdall driver, we will first attempt to load the driver remotely, i.e. by downloading the driver from the central manager. This provides a single source for the drivers, simplifying the management of the drivers. If this fails, it will attempt to use the local class loader in order to find the driver. In an OSGI Java environment, it may be necessary to package the database driver with the Heimdall driver as a dependency in order for all behaviors to work as expected if the remote load feature does not work.

Proxy vs. JDBC Driver Behavior

When using Heimdall in proxy mode, it is highly recommended that the drivers provided in the Heimdall install package be used. This is as a result of the tight integration work that is done with Heimdall to operate with these drivers, and may be broken between release builds of the vendor driver. In JDBC mode, this requirement can be ignored, and the application recommended version of the drivers should be used.

Rule Overview



A rule list is a set of rules that is processed in order on queries, and dictates the processing done on a particular query and its response. A rule is composed of several components:

- **#:** A current row indicator (for moving rows) and the index of the rule.
- **Enabled:** If the rule is enabled. Non-enabled rules will not be processed.
- **Note Indicator:** When a note is attached to a rule, an icon will be visible, and the note can be viewed by hovering the mouse over the icon
- **Regular Expression:** See below for regular expression behavior
- **In-Trans Flag:** This specifies if a rule should operate on a query that is in the middle of a transaction, i.e. it is not in auto-commit mode in an explicit transactional context (between "Start Transaction" and "commit" or "rollback")
- **Action:** What action should be taken on a regex match
- **Parameter:** A parameter that modifies the behavior of the action
- **Value:** The value of the parameter, e.g. to specify the maximum time to live for a cache rule
- **Edit:** An icon to provide an expanded view of the rule, and to allow editing of comments
- **Delete:** Toggle deletion of a rule on the next commit

To re-order a rule, it can simply be dragged into place in the desired order.

Hint: To better edit a rule, make sure to use the pencil icon to open up the expanded window, which provides more editing options.

Regular Expression Field Behavior

Each rule has a field that can contain either a re2j 1.1 (https://github.com/google/re2j) regular expression, or an extended specifier (below). The re2j regular expression language is nearly identical to normal Java regular expressions, except that it can operate in linear time, while Java regular expressions may end up being unbounded in time. As a tradeoff, certain features dependent on backtracking are removed.

Examples of some simple regular expressions:

- "(?i)^select" Case insensitive match all queries that start with "select" at the start of the line
- "(?i)somestring" Case insensitive match queries with the string "sometable" anywhere in the query

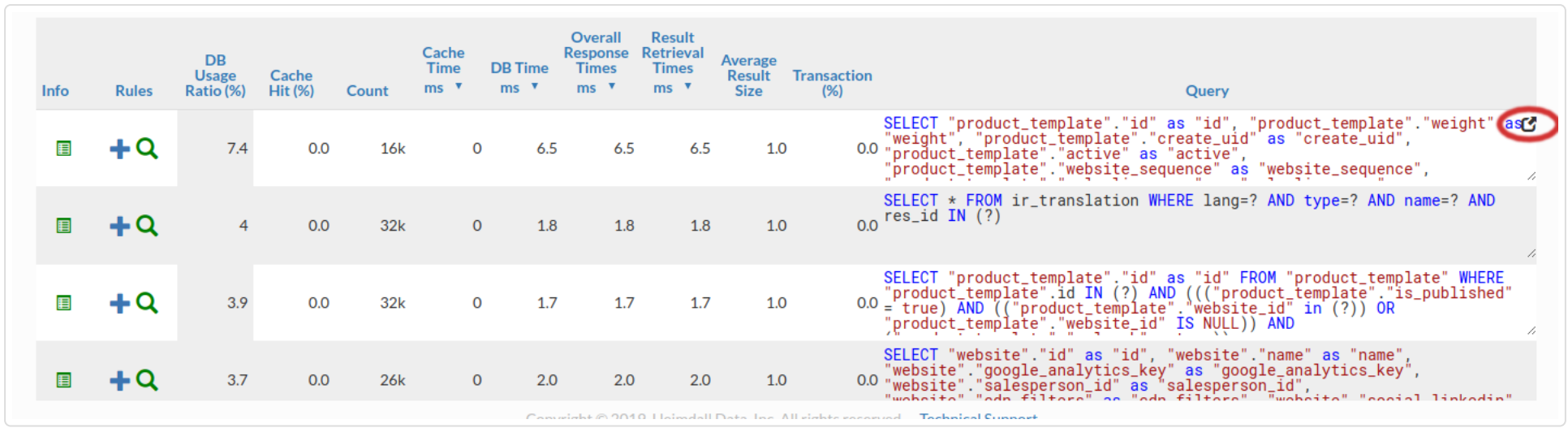
In addition to regular expressions, the following extended specifiers can be used instead:

- **literals**:{string1},{string22}..., to match any one string literal in a list.
- **tables**:{table1},{table2}..., to match any one table in the named list.
- **tablesall**:{table1},{table2}..., match only if all the tables in the list are present in the query tables.
- **tablesonly**:{table1},{table2}..., match only if all the tables in the query tables are in the list present (but all do not need to be).
- **tablesexactly**:{table1},{table2}..., match only if every table in the list is present in the query tables, and no other tables are present, i.e. an exact match between table lists.
- **users**:{user1},{user2}..., match if the query's uses matches one of the users in the list (as regex).

- **ldapgroups**:{group1},{group1}..., Match if any of the named groups matches a group the connected user is a member of (as regex)
- **catalog**:{catalog1},{catalog2}..., match if the query catalog matches any catalog in the list (as regex)
- **catalogprefix**:{prefix1},{prefix2}..., match if the query catalog starts with any string in the list
- **appname**:{application_name1},{application_name2}..., match if the postgres application name matches (as regex)
- **ports**:{port1},{port2}..., match the port the query was received on, via the vdb. This allows multiple ports to be used on the vdb, and have a different behavior based on which port was used.
- **ips**:{ip1},{ip2}..., match the IP the query was received from, via the vdb. Subnet matching is not currently supported. If complex IP based rules are desired, it is suggested that an additional port be added to the proxy, the ports qualifier be used, and firewall rules can be used to control what IP ranges have access to this new port with the desired configured behaviors.

Note for extended specifiers: The fully qualified catalog.table name is expected, so if using the "magento" database with a table of "users", you would match against "magento.users". Please note the dynamic parameters below for substitutions that are supported here as well. To verify that the table name is as expected, use the "printtable" option, view the query in the expanded analytics tab, or use the debug option to view the fully qualified table name as used internally. Alternatively, use the "expanded query view" in the analytics, and then scroll down to the table names:

To open the expanded query view:



Info	Rules	DB Usage Ratio (%)	Cache Hit (%)	Count	Cache Time ms	DB Time ms	Overall Response Times ms	Result Retrieval Times ms	Average Result Size	Transaction (%)	Query
		7.4	0.0	16k	0	6.5	6.5	6.5	1.0	0.0	SELECT "product_template"."id" as "id", "product_template"."weight" as "weight", "product_template"."create_uid" as "create_uid", "product_template"."active" as "active", "product_template"."website_sequence" as "website_sequence",
		4	0.0	32k	0	1.8	1.8	1.8	1.0	0.0	SELECT * FROM ir_translation WHERE lang=? AND type=? AND name=? AND res_id IN (?)
		3.9	0.0	32k	0	1.7	1.7	1.7	1.0	0.0	SELECT "product_template"."id" as "id" FROM "product_template" WHERE "product_template".id IN (?) AND ((("product_template"."is_published" = true) AND (("product_template"."website_id" in (?)) OR "product_template"."website_id" IS NULL)) AND
		3.7	0.0	26k	0	2.0	2.0	2.0	1.0	0.0	SELECT "website"."id" as "id", "website"."name" as "name", "website"."google_analytics_key" as "google_analytics_key", "website"."salesperson_id" as "salesperson_id",

Copyright © 2019 Heimdall Data, Inc. All rights reserved. Technical Support

And to view the fully-qualified table names, in the expanded view, scroll down to the tables section before any query plans:

```
    (
      ("product_template"."is_published" = true)
      AND (
        ("product_template"."website_id" in (?))
        OR "product_template"."website_id" IS NULL
      )
    )
    AND ("product_template"."sale_ok" = true)
  )
```

Tables

- odoo.public.product_template
- odoo.public.ir_translation

Execution Plans

Odoo-vdb

2 times, last at 09/29/2019 11:35:53:997

Additionally, you can use the string "${catalog}" to match the current connections catalog in processing a rule, so as to allow matching on multiple catalogs as appropriate.

Examples of some extended regular expressions:

- "tables:magento.core_store" to match on the table magento.core_store only, but any other table may exist as well

- "tables:${catalog}.core_store" to match on the table core_store only, but in any catalog, not just the magento catalog
- "tablesexactly:magento.core_store" to match on the table magento.core_store only, and no other tables are in the query

When using the expanded rule view, it is possible to add more than one regex field to a rule, and then specify if AND or OR logic should be used to connect them. This allows for example, one to specify a tables: specifier, then to filter say INSERT, UPDATES or DELETES only against that table.

Rule Processing Behavior

All rules are processed in order of evaluation. This means that a rule that matches earlier in processing can be overridden later in processing by the same rule type.

In order to process a rule, the following steps are taken:

- Check if rule list is enabled
- Iterate through rules
- Check if a rule is enabled
- Check if the in-transaction flag is set when in a transaction
- Check if the rule type should be excluded for processing, due to a previous ignore rule
- Check if the regular expression matches (see below)
- Process the capture parameter, if set, for dynamic parameters
- Check if the rule rate matching limits are allowed
- Process the update, printmatch, printtables, tables, and notify flags (although no action is taken at this point)
- Process the rate limiting parameters (except delay) this is done only once, after caching, async and forwarding are executed)
- Per-action rule processing

Note: In the case of multiple Allow rules in consecutive order, the rules internally will self-organize based on the frequency of hits observed by the driver. The more often a rule has been hit, the further up the list it will be positioned to optimize the regex lookup performance. Any other type of rule will be processed in the order specified, and no Allow rule will optimize around another other rule type.

Special case: The nocache rule behaves as both a "cache with a ttl of 0" and "ignore all other cache rules following", i.e. it prevents all past and future cache rules from impacting the query. Most rules can be overridden, but for safety, the nocache rule can not be overridden once it matches a particular pattern.

Rule Flow Control & Dynamic Parameters

Several actions and properties can be used to control the processing of rules for more advanced applications:

- **Ignore Action**: When an ignore action is specified, the name of an action type can be specified. Any rule matching that action type will be ignored in any further processing of the matching query. This can be used to create exclusions where a particular action shouldn't be taken, but creating a regular expression to account for this would be difficult.
- **Call Action**: A call can be used to call by name a nested rule-list, to allow one rule-list to be called from another rule-list. This can be leveraged to provide a common set of rules for an application, but custom behaviors for particular instances. Another use case is to use a single regular expression match to apply multiple actions to a given set of content, as in the called rule, an empty regex can be used.
- **Stop Parameter**: When specified on a matching rule, no further processing will occur in the current rule-list for the given query. If in a nested rule-list due to a Call action, the calling rule-list will continue to be processed.
- **Capture Parameter**: This parameter triggers dynamic property evaluation. When set, regular expression capture groups are used to edit the parameter values for the query, in order to dynamically adjust behavior. A common use would be to include a comment at the front of a query, and then use the value in the comment part of or a parameter's value, for example, to dynamically set the TTL of a query based on the comment's included value: Example:

  Regex: /* (.*) */

  Action: Cache, ttl=${1}

  Query: / ttl=30000 / SELECT * from Table

Result: The query would be cached for 30 seconds

Similar to the capture parameter based on regular expression capture groups, some built-in strings have special meaning in properties, and will be replaced with the internal values during rule processing:

- **${catalog}** or **${database}**: the current catalog/(database in MySQL term) being used.
- **${vdb}**: The VDB the traffic was received on
- **${user}**: The JDBC user on a connection
- **${connid}**: The connection ID of the front-end connection the query was issued on

Note: These variables can be used when using the extended table matching syntax, such as "tables:${catalog}.tablename" to avoid having to specify the actual catalog to match against. There is no need to specify the "capture" parameter for these to operate in this context.

Rate Limiting

Often, such as in the case of a "Log" rule action, it is desirable to limit the rate that a particular rule is matched to an absolute number. Each rule has several options that can be used for this:

- **matchlimit**: Enforces a rate of X pattern matches, with up to Y seconds of burst. To specify the burst value, use the parameter "maxburst". Useful to limit the amount of log data generated in a similar manner to using the sample parameter.
- **maxburst**: Specify the number of seconds of bursting that is allowed before clamping the rate for the ratelimit or matchlimit parameters. Defaults to 10.
- **ratelimitname**: Specify a "rate limit" control token name for limiting. This can be a dynamic value for say ${catalog} or ${user} to limit rate by various metadata.
- **concurrency**: Specify the number of parallel queries at a time that can be executed. The lock will be released after the the "execution" is complete, i.e. after the initial data is received, but not necessarily after all the data has been transmitted by the database;
- **concurrencyname**: Name the token for tracking of concurrency.

Another case is that a rule should be evaluated only a certain percentage of the time, say 1 out of 10. The following parameters can be used for this:

- **exclude**: A ratio of queries should NOT match the policy, as expressed as 1/X. Useful to allow a small number of objects that would otherwise be a cache hit to be a cache miss instead. Example: A value of 10 would exclude 1 out of every 10 matches from actually matching the rule.
- **sample**: A ratio of queries should match the policy, as expressed as 1/X. Useful to limit logging to a small percentage of traffic. Example: A value of 10 would mean that 1 out of 10 matches would be treated as a match.

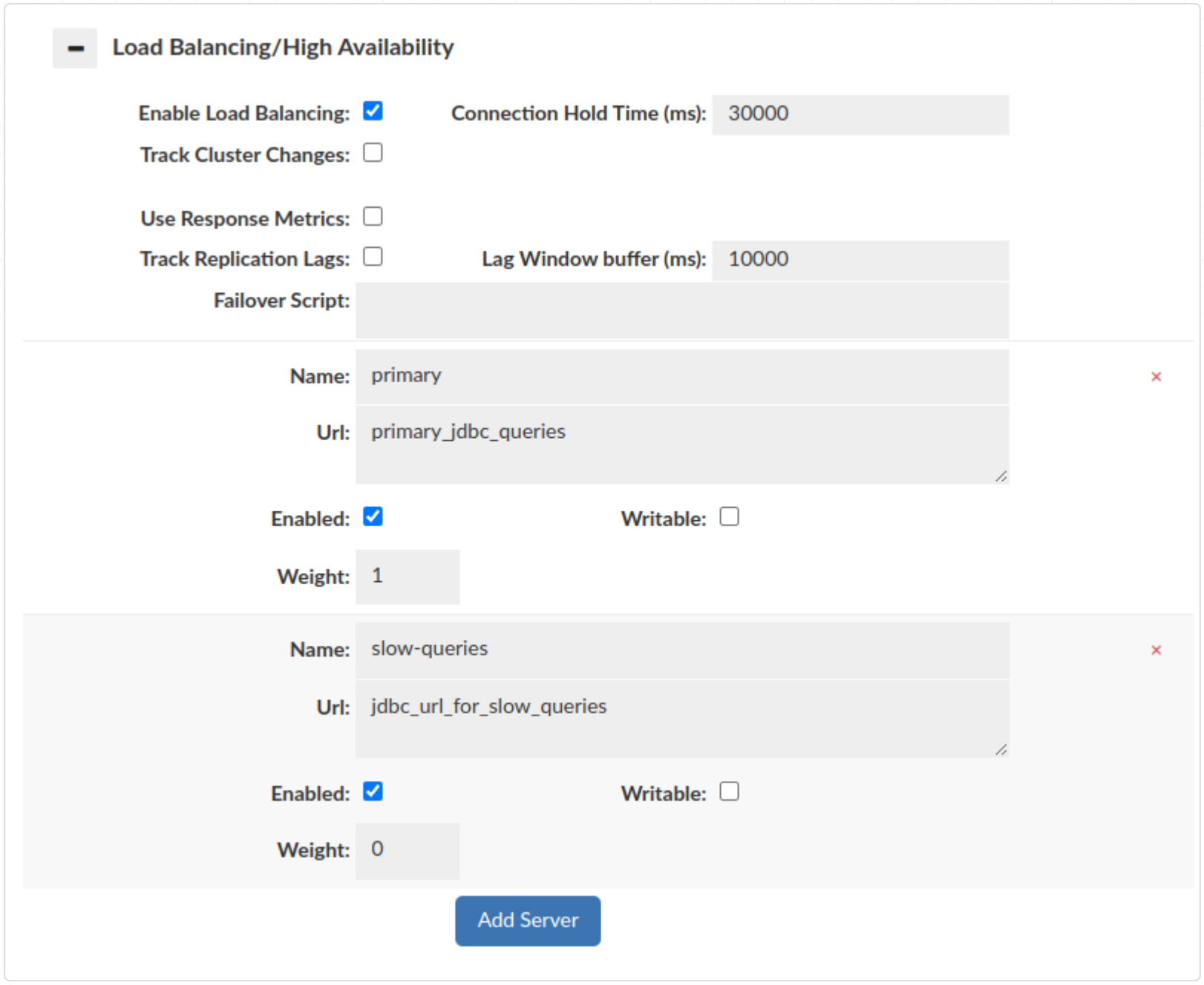
A final case is where queries matching a rule should themselves be slowed down, say to help prevent an outage of all traffic due to a DoS attack. The following parameters can be used in such cases:

- **delay**: When a rule with a delay flag matches, induce a delay in processing. Useful to determine if a rule can have an overall impact on performance, i.e. if a delay doesn't make a noticeable difference, then caching won't either. Also useful in inducing delays to determine if moving the database further away will impact performance significantly. This delay will occur only if the query is requested from the database, i.e. it will not impact a cached query.
- **ratelimit**: Enforces a rate of X queries per second, with up to Y seconds of burst. To specify the burst value, use the parameter 'maxburst'. This will actually SLOW DOWN queries matching this rule to at most this value.

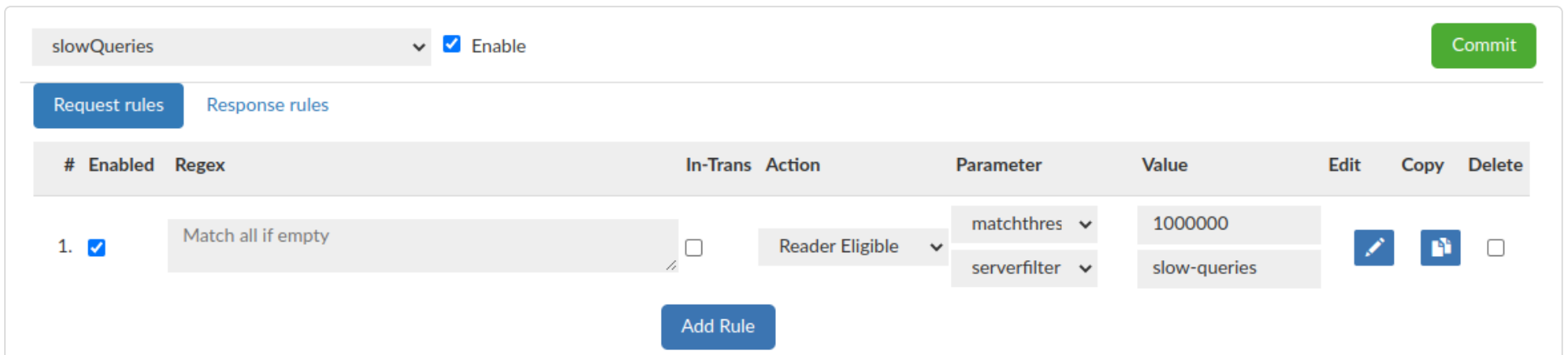
Queries Forwarding

Configuring data source and rules in a proper way allows us to forward slow queries to another data source to avoid database performance drop. If average runtime will be longer than given 'matchthreshold', then query will be forwarded to data source defined by 'serverfilter'.

Data Source configuration:



Rules configuration:



Rule Details

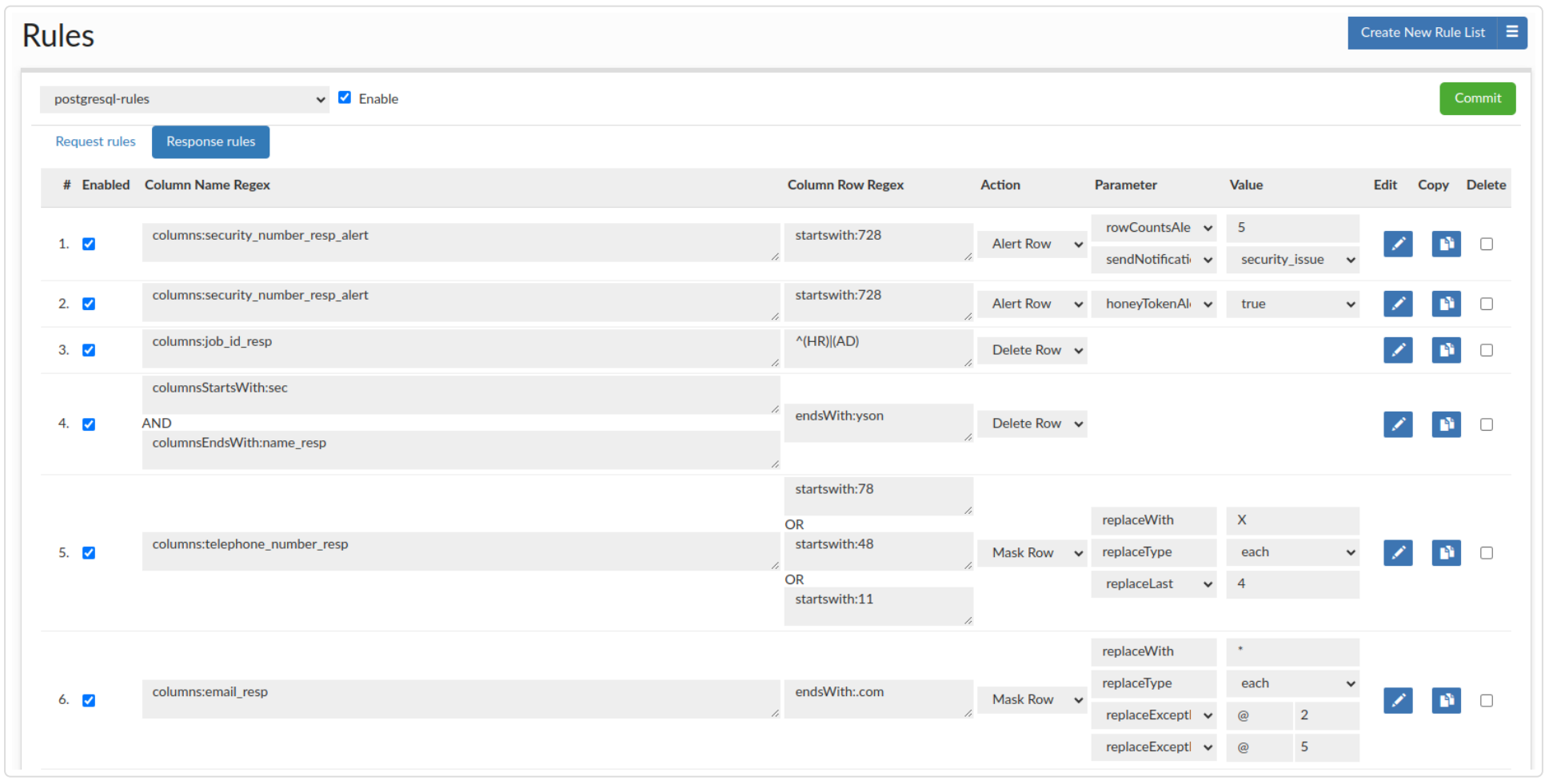
- **For Any Rule:**
  - **delay:** Adds X microseconds to each query, to help model the impact of latency between the db client and server, for hybrid cloud impact testing
  - **evicttables:** Tables that should be flagged as evicted when a query matches the rule
  - **exclude:** Exclude matching the rule 1/X times, to allow a small sample of queries to match
  - **invalidate:** Can be specified to override invalidation behavior of writes for matching queries, default is to allow invalidations to occur.
  - **log:** When specified, behave as if a log rule had matched, used to consolidate rules
  - **matchdelta:** Specifies the minimum time between matches on a per-proxy basis, overrides matchlimit
  - **matchlimit:** Specifies the frequency (in X per second) that a rule should match
  - **matchthreshold:** Only match to unknown queries or ones that are known to have an average runtime longer than X microseconds
  - **maxburst:** When using matchlimit or ratelimit, specify the number of seconds worth of bursting to allow before limiting the rate

    - **notify:** If specified as true, flag that the notification configuration should be used to notify on a match
    - **notrack:** When specified, don't track the performance of this query for statistics records. Useful to "hide" overhead queries that aren't impacted by performance
    - **olderthan:** specify the time window in ms to allow content to not be evicted on eviction. Negative prevents caching into the future
    - **preparedonly: ** Match only if a query is a prepared query. This can be used to trigger exceptions to change settings for users to optimize pooling behavior
    - **printcapture:** When a rule matches, print capture groups of the rule match
    - **printmatch:** When a rule matches, print the rule that matched
    - **printmeta:** Print the resultset metadata for results returned
    - **printresults:** Prints to logs the result of matching queries
    - **printtables:** If specified, print the table attributes to be used for caching & invalidation
    - **printtiming:** Prints to logs the execution time of this rule, useful for profiling called rulelists, such as for the SQL Firewall
    - **ratelimit:** Limits matching queries to X per second. This is tracked on a per-rule basis, so ratelimit and matchlimit can interact on the same rule
    - **sample:** Match the rule only 1/X times, often to reduce frequency of logging, but remain proportionate to load
    - **serverfilter:** A filter (regex) to determine what read-only servers should be used to handle a request. If none match, this is ignored. Only used with forwarding & read/write split
    - **stop:** If specified on a matching rule, no further rule processing in the current rule list will occur
    - **tables:** For any matching rule, specifies the tables that should be added to a query's table list. Use "none" to allow caching without tables. Newlines and whitespace are acceptable between table names, but not in table names.
    - **update:** Specifies if the matching queries should be processed as updates by the cache engine
    - **queryTimeout:** Query execution timeout
- **Async Execute:** Execute insert/update/delete asynchronously, not supported by MySQL proxy
    - **asyncsize:** The maximum length of a query (in characters) that can be made asynchronous
    - **batchSize:** The maximum batch size of queries to be executed--queries will not be delayed to fill the queue, but this sets the maxiumim size of
    - **maxPollInterval:** The maximum time (in ms) to poll after an async query is received before a batch is processed, defaults to 0 for no delay the batch.
    - **source:** The name of the data source to forward the query to for async execute. This source must be specified in the VDB definition.
    - **queueName:** The per-source named async queue to process the query from. Defaults to the name of the data source the query is sent to. Set to ${table} to allow dynamic replacement of the table name (if only one).
    - **holdUntil:** When to hold the thread until
    - **spoofedResult:** The result the DML will return if the result is spoofed (only for immediate holdUntil values).
    - **lockTable:** Lock tables of async DML while the DML is pending, to serialize access to the table (default to TRUE)
- **Nocache:** Disable caching for a query, overrides previous cache rules as well as prevents later cache rules from being evaluated
- **Cache:** Use the cache configuration for the VDB to cache matching queries, if possible
    - **ttl:** Specifies the TTL in ms of objects cached, if not evicted due to a write first
    - **cachesystem:** True if Heimdall should cache system tables, and not just user tables, can break things!
    - **unconditional:** if true, bypass many internal checks to insure a query is cacheable
    - **maxLocalSize:** Specify the largest size of a cachable object in local memory, defaults to Integer.MAX_VALUE
    - **maxSize:** Specify the largest size of a cachable object, defaults to Integer.MAX_VALUE
    - **autoRefresh:** Specify that queries may be automatically refreshed in cache if invalidated (but not on TTL expiration). Only a limit number of queries are tracked for this purpose, configurable on the vdb properties
    - **revalidateOnly:** Don't actually serve from cache, but validate that the result in the cache is the same as what the server serves
    - **shared:** Share a cache object across users. Note, this may bypass security, so should be used with caution. To globally trigger this, use the custom cache key option in the vdb cache settings.
- **Drop:** If not allowed with an allow rule, drop the request, and generate SQL Exception
- **Result:** Return an integer value as a result of a SQL call, i.e. for rows updated
- **Retry:** Retry execution if exception occurs
    - **maxRetries:** Maximum number of attempts that will be made if none of them succeeds

    - **exception:** String that exception should contain
    - **retryDelay:** Delay between retry attempts
- **Forward:** Allows queries to be sent to alternate data sources for execution
    - **source:** The name of the data source to forward the query to. This source must be specified in the VDB configuration.
    - **setcatalog:** If in forwarding, should we inherit/set the catalog from the parent connection on the forwarded connection.
    - **setschema:** If in forwarding, should we inherit/set the schema from the parent connection on the forwarded connection.
    - **readonly:** When forwarding, should the connection be allowed to read-only servers.
- **Call:** Execute (call) a named rule list, must also be included in the VDB config to work
    - **rule:** The rule name to call/execute in place of the call rule itself, if matched.
- **Ignore:** Ignore rules of the specified type in the current rule-list that occur after this rule
    - **action:** What type of action to ignore after matching
- **Log:** Log the SQL observed in the query
    - **logcolumns:** List of comma separated column names/aliases to save in log records, based on the resultset metadata. Use '*' or 'all' for all columns. Note: HIGHLY impacting on performance.
    - **slowtime:** Response time in microseconds above which logging should be triggered.
- **Extract Plan:** Automatically (periodically) retrieve query plan on queries
    - **threshold:** The response time threshold in ms after which the plan is requested, based on query pattern average response times
    - **frequency:** How many times the query plan can be requested per hour
    - **format:** What format to use when extracting an execution plan
    - **force:** By default Extract Plan works only for queries with result set from DB (for example, SELECT queries). To force explaining on any queries set the parameter to true.
    - **checkTableMismatch:** For SQL Server, check if tables in the explain match what is tracked internally by Heimdall
- **Learn Pattern:** Generate a firewall learn action to add a rule to a named rule list. A new rule will not be added if one with the same pattern and action type already exists.
    - **rule:** The name of the target rule list to add learned rules to.
    - **action:** Specifies the action of newly added rules, defaults to "allow".
- **Allow:** Explicity allow a query to bypass the learn and drop rule types
- **Trigger:** Trigger another SQL or script to execute as a result of a matching query
    - **type:** Specifies the type of trigger action to perform. If execute, then it will call the named program or script, and execute on the central manager (always). If SQL, it will execute on the proxy/driver unless in parallel, delay and queueunique is TRUE, then it will process on the central manager.
    - **command:** The command to execute, either as sql or as a script. Accepts a replacement value of ${tables} to represent the tables in the query.
    - **timing:** When to execute the command, either before, after, or in parallel to the triggering sql. With before, if an exception is encountered on the initial query, the trigger action is not performed. If an exception is encountered with after, then the exception will be reported to the calling thread. When sql is set to execute in parallel mode, it will execute on the central manager if delayed, on the proxy/driver if not, and exceptions will only be reported in the logs.
    - **delay:** When running in parallel to the triggering sql, how much time to delay processing (in ms), if any
    - **maxDelay:** When running in parallel, delayed and queueunique, how often should we run (at least) even if the trigger action keeps occurring, i.e. to insure data inserted is flushed to another db at least X times an hour
    - **queueUnique:** When performing delayed parallel queries, should the execution queue only maintain unique command executions, to prevent the same command from executing over and over later?
    - **source:** The name of the data source to execute a SQL query on. This source must be specified in the VDB configuration.
- **Stack Trace:** Generate a Java stack trace in logs on match
    - **threshold:** Response time threshold for stack traces to trigger (in microseconds)
- **Transform:** Change the query based on capture group manipulation
    - **target:** When translating a query, the target of the translation, with $1 representing the first capture group, $2 the second, etc.
- **Table Cache:** Apply a TTL to a query based on if the table specified is referenced by the query. Lower TTL values override higher ones
    - **ttl:** Specifies the TTL of objects cached, if not evicted due to a write first

    - **table:** In table cache rules, specifies the name of the table to apply settings for
- **Tag:** A no-op action, for use with universal parameters only
- **Debug:** Allows to enable and disable debug mode for a connection
    - **enabled:** If true, enables driver debug logging at that point
- **Pool:** Pool and connection multiplexing behavior overrides
    - **maxUserActive:** Sets the per-user active pool parameter (idle+busy), and overrides any default set in the source settings. Useful to allow individual users to have more connections than others, and can be set based on AD group, say to allow service accounts more connections than general users (Integer).
    - **maxUserIdle:** Overrides the per-user allowed idle connection limit (Integer).
    - **multiplex:** Override the vdb level multiplex setting to either enable or disable multiplexing (Boolean).
    - **multiplexSkip:** Triggers skipping multiplexing for X number of queries after the current matching one (Integer).
    - **reuse:** Control if connections can be reused on return by pooling, defaults to true (Boolean). Only impacts the primary (writer) connection, not read/write split connections.
- **Procedure Wrap:** Wraps SQL queries into a procedure. Only works for Microsoft SQL Server. Necessary for proper behavior when variables dependency is required e.g. with clause OPTIMIZE FOR in query.
- **Reader Eligible:** Flag that the query is allowed to be processed by a read-only server in a cluster.
- **Multiplex:** Manage multiplex behavior (works only when the multiplex is enabled in VDB config)
    - **disable:** When the capture parameter from regex is matched, then it is saved in memory and from this point multiplex will be disabled for all queries, unless it will be enabled again.
    - **enable:** When the capture parameter from regex is matched, then it is removed from the memory (mentioned in the "disable" point) and multiplex will be enabled for next executed query (unless there are any other captured parameters disabled)

Response Rules



In the Response Rule tab, we can define actions that will be processed for responses. The rule consists of several elements:

- **#:** A current row indicator (for moving rows) and the index of the rule.
- **Enabled:** If the rule is enabled. Non-enabled rules will not be processed.
- **Notes Indicator:** When a note is attached to a rule, an icon will be visible, and the note can be viewed by hovering the mouse over the icon
- **Column Name Regex:** Specify the column name regular expression to match with. See column name regexes below for more information.
- **Column Row Regex:** Specify the column row regular expression to match with. See column row regex below for more information.
- **Action:** What action should be taken on a regex match.

- **Parameter:** A parameter that modifies the behavior of the action.
- **Value:** The value of the parameter, e.g. to specify the replace string for a masking rule.
- **Edit:** An icon to provide an expanded view of the rule, and to allow editing of comments.
- **Copy:** Allows to copy chosen rule.
- **Delete:** Toggle deletion of a rule on the next commit.

To re-order a rule, it can simply be dragged into place in the desired order.

If there is a row of data where more than one rule tries to perform the same operation (e.g. two or more alarms or masking rules), only the last rule will be applied. It still allows for masking and alerting on the same field as well as applying the same rule to different columns in the same row.

Response rules are also applied to metadata retrieval.

Hint: To better edit a rule, make sure to use the pencil icon to open up the expanded window, which provides more editing options.

Column Name Regex Behavior

Defines the column name expression to match.

Examples of specifiers that can be used :

- **columns**:{column1},{column2}..., to match any column name literal in a list.
- **columnsStartsWith**:{column1},{column2}..., to match any column names that starts with any string in the list.
- **columnsEndsWith**:{column1},{column2}..., to match any column names that ends with any string in the list.

Column Row Regex Behavior

Defines the column row expression to match.

Each column row regex field can contain either a re2j 1.1 (https://github.com/google/re2j) regular expression, or an extended specifier (below). The re2j regular expression language is nearly identical to normal Java regular expressions, except that it can operate in linear time, while Java regular expressions may end up being unbounded in time. As a tradeoff, certain features dependent on backtracking are removed.

Examples of some addition to regular expressions, the following extended specifiers can be used instead:

- **literal**:{column1},{column2}..., to match any column name literal in a list.
- **startswith**:{column1},{column2}..., to match any column name that starts with any of the given string.
- **endswith**:{column1},{column2}..., to match any column name that ends with any of the given string.

Response Rule Action Details

Parameters of each rule type can be chained, editing is required to add more than one parameter. In a masking rule the order of processing is the same as the order of the printed parameter options.

- **Alert Row:** Raise an alert if the response data conditions are met (there is a match for Column Name Regex and Column Row Regex)
  - **honeyTokenAlert:** Raise alert if queried for a "honey token" (if all conditions were met)
  - **rowCountsAlert:** Raise alert if more than a certain number of rows are queried as a result of a single query. (ColumnRowRegex field should be left blank, otherwise the alert will be promoted when at least one match is made in a row and other conditions are met - so basically it's looking for "token" that will trigger alerts)
  - **sendNotification:** Sends email notification via AWS SNS service, when alert rule (such as honeyTokenAlert or rowCountsAlert) is triggered. It is recommended to utilize the 'Edit' button to associate the 'sendNotification' rule with an existing alert rule. Sent email will contain alert details, and its subject will be dynamically set as your vdb name that generated this alert. Before using this feature, client has to previously define Notification entry in the Admin -> Notification tab. More information about Notification system is available here
- **Delete Row:** Deletes the matching rows from result set (might be used on metadata as well).
- **Mask Row:** Masking row of data from result set
  - **replaceWith:** Specifies with which string, data should be replaced. (This field is mandatory)
  - **replaceType:** Specifies the type of replacement, whether we should replace Each string/character or All with value provided by replaceWith field.
  - **replaceFirst:** Specifies how many first characters to replace with the given string provided in the replaceWith field.

- **replaceExceptFirst:** Specifies how many of the first characters are left unchanged and the rest will be replaced.
- **replaceLast:** Specifies how many of the last characters should be replaced with the given string specified in the replaceWith field.
- **replaceExceptLast:** Specifies how many of the last characters are left unchanged and the rest will be replaced.
- **replaceAll:** Specifies whether whole row of data should be replaced. Should not be combined with other masking actions (as the result will be returned immediately).
- **replaceBeforeString:** Specifies delimiter and number of characters (in that order) to replace before given delimiter. Can be usefully with email masking.
- **replaceExceptFirstBeforeString:** Specifies delimiter and number of characters (in that order) that should remain unchanged (counting form the beginning of the delimiter) and the rest of the string will be masked to the specified delimiter.
- **replaceAfterString:** Specifies delimiter and number of characters (in that order) to replace after the given delimiter.
- **replaceExceptLastAfterString:** Specifies delimiter and number of characters (in that order) that should remain unchanged (counting form the end of the delimiter) and the rest of the string will be masked to the specified delimiter.
- **replaceFirstRegex:** Specifies the regex how the field should be replaced, matches only the first matching regex pattern. The replaceType field has no effect on regular expression processing.
- **replaceAllRegex:** Specifies the regex how to replace the field, matches all matching regex. The replaceType field has no effect on regular expression processing.

Additional examples of using response rules

Create two rules, one for the "honey token" alert and one for delete row that will delete that honey token, so that the final user only sees the alert but not the "honey token" data. The number of rows and their size would be computed after that operation, so from the perspective of the user (who doesn't have access to the HeimdallData Centrall Management console) they wouldn't know that any "honey token" was queried.

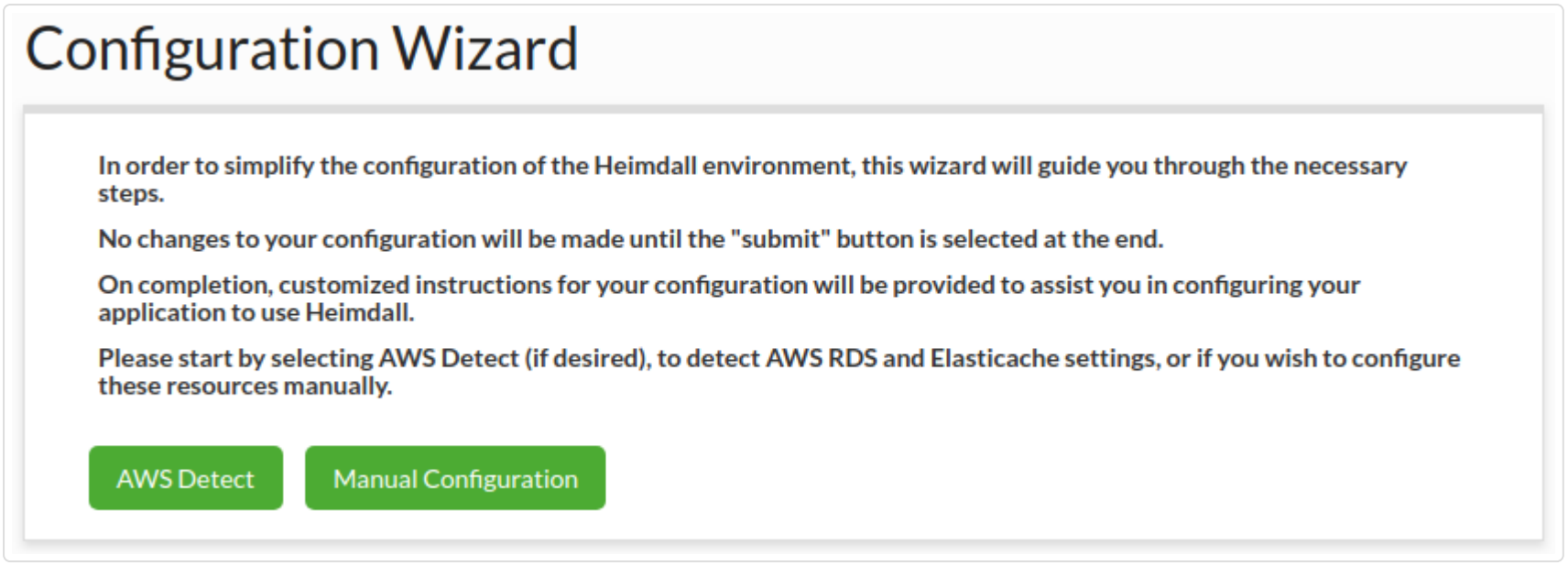
When working with application like DBeaver, we need to load a lot of data at startup, including all the tables, even if we need only a few of them. We can use Delete Row rule, to edit the metadata results from information schema so that all unnecessary tables/partitions are removed from the metadata, greatly reducing amount of data load to DBeaver, and improving boot time for users.

Wizard Overview

The Wizard is provided to provided a guided walk-through to configure a VDB, data source, and desired pre-configured rule-sets that may be desired.

Manual vs. AWS Detect

If the Heimdall management server detects that it is in AWS, it will offer the AWS Detect option in the first section of the wizard. For this to work, either an IAM secret and key need to be provided OR an appropriate IAM role needs to be associated with the ec2 instance the management server is running on. Please see the aws environment section for more details. Otherwise, simply select Manual Configuration to continue to the next section. Using the AWS detect option does NOT change what options are available, it simply allows defaults to be pre-populated to simplify the process.



If the AWS Detect option fails, one of the more common reasons is due to a locked down VPC that doesn't provide connectivity to the API endpoints. You can test the various endpoints via an SSH session to the Heimdall manager, then using the following (adjust the region name as appropriate:

```
wget https://ec2.us-east-1.amazonaws.com/
wget https://rds.us-east-1.amazonaws.com/
```

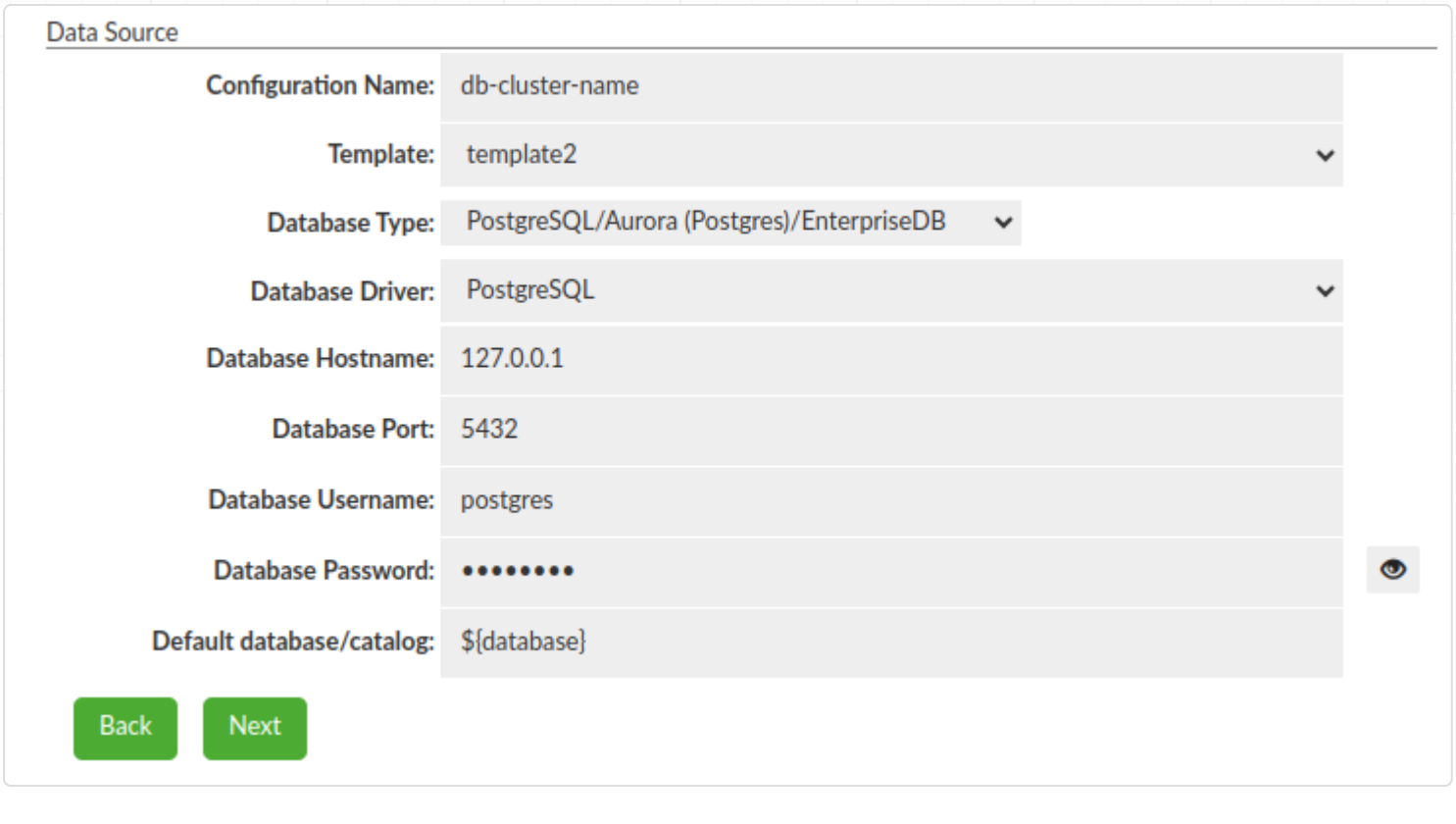
Please see the additional comment on api endpoints in the AWS IAM section.

Data Source

Here, the database settings are configured. First, name the configuration (this should be a unique name that correlates to the application or database in question). Next, select the database type and driver, which sets a variety of defaults based on the database capability. The rest of the database values should be set appropriately. In the case of Postgres, since a database instance will either implicitly or explicitly need to be selected, use ${database} as the default, unless it is known that only one database instance will be used. In the case of a proxy, Heimdall will observe what the desired database is based on the connect negotiation, and will fill in this value automatically as needed.

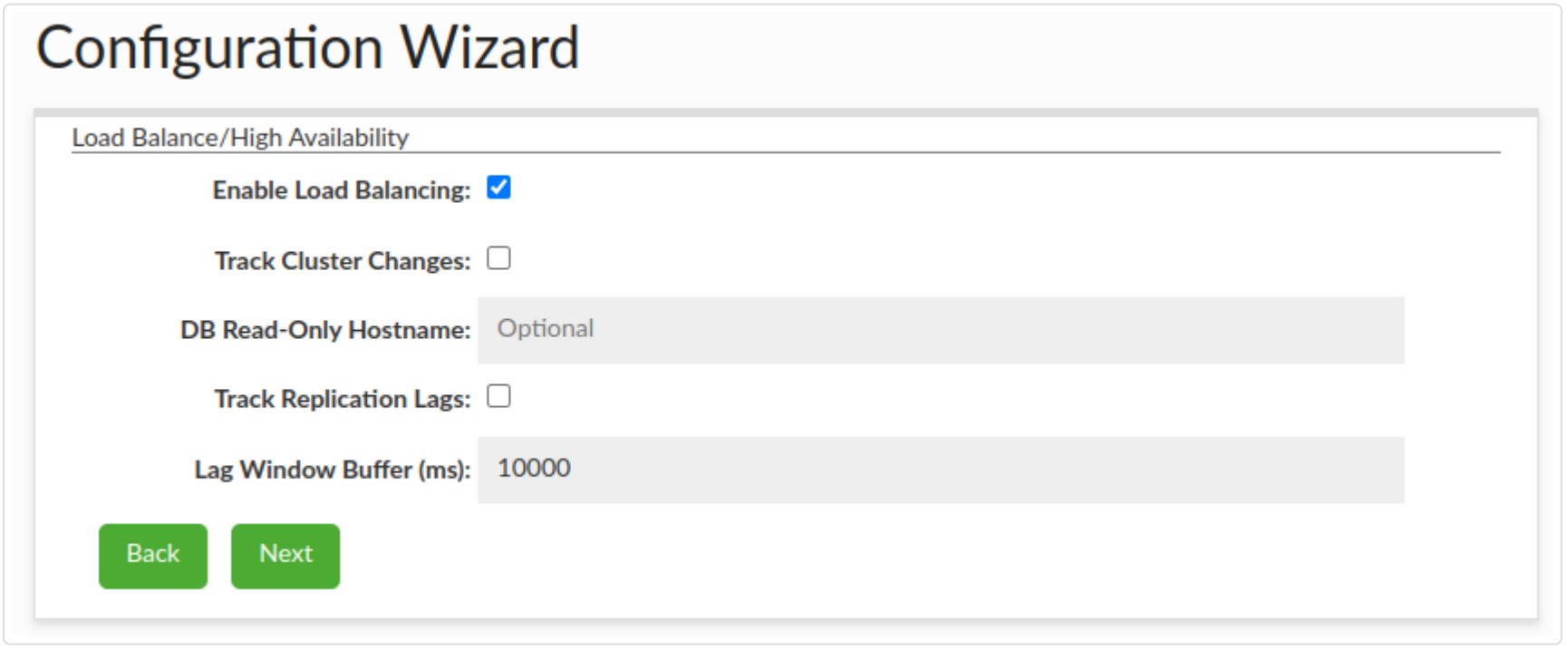
The Heimdall management server detects which cloud environment it is hosted in and provides the correct database names for the detected environment. If you don't see the type of database you are actually using, you can disable "cloudDetection" in the admin tab and then all database types will be shown in the list.

You can also choose a pre-made template from the library that will complete the configuration fields in the wizard, and you will only have to customize some of them to your needs. Remember! Not all things can be set in the wizard. Some of them you will have to set after finishing the wizard.



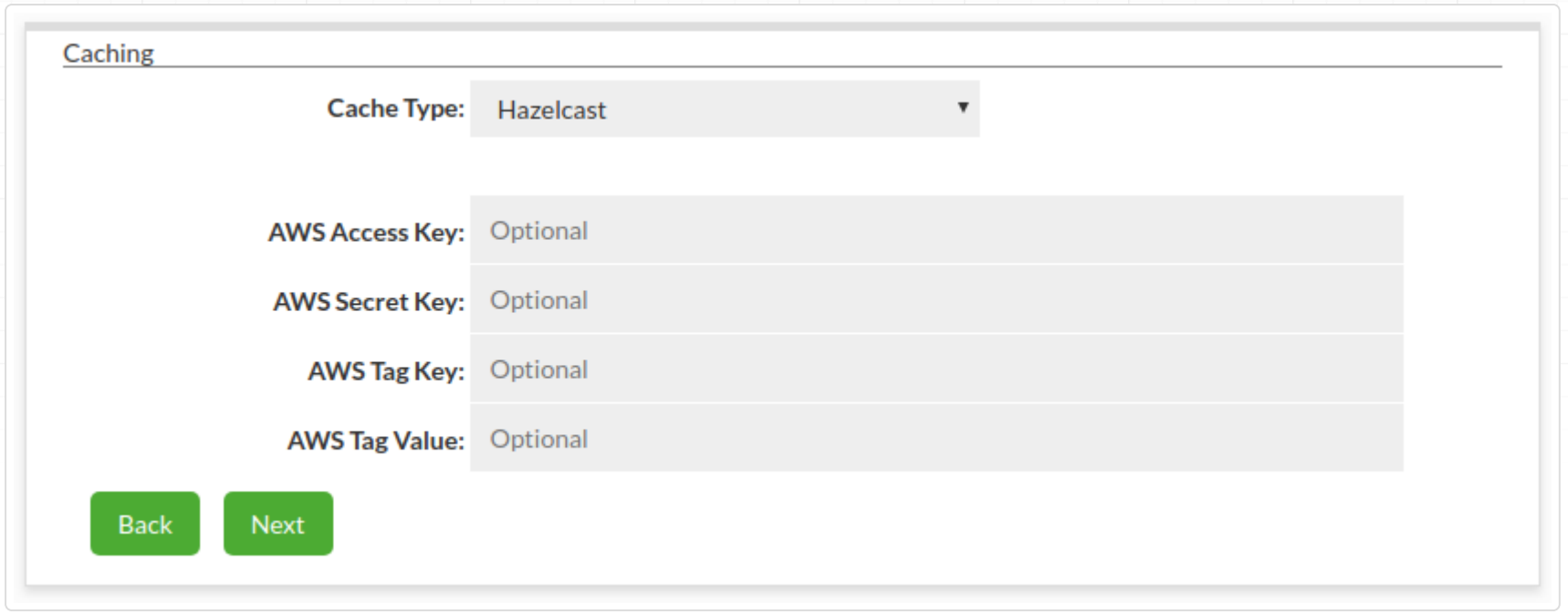
Load Balancing/High Availability

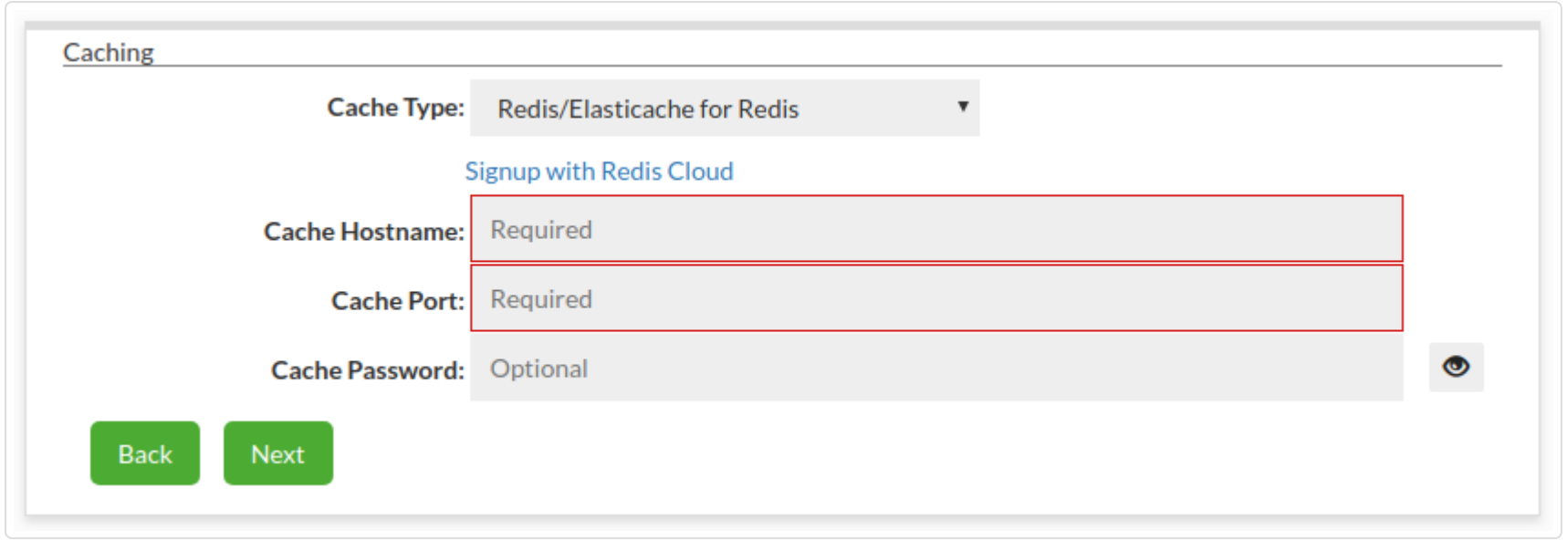
Here the various LB and High Availability options can be set. In the case there is a read-only server, enter it here. If there is more than one, they can be added in the data source tab once created. Please see the sources section for more details on each option.



Cache Configuration

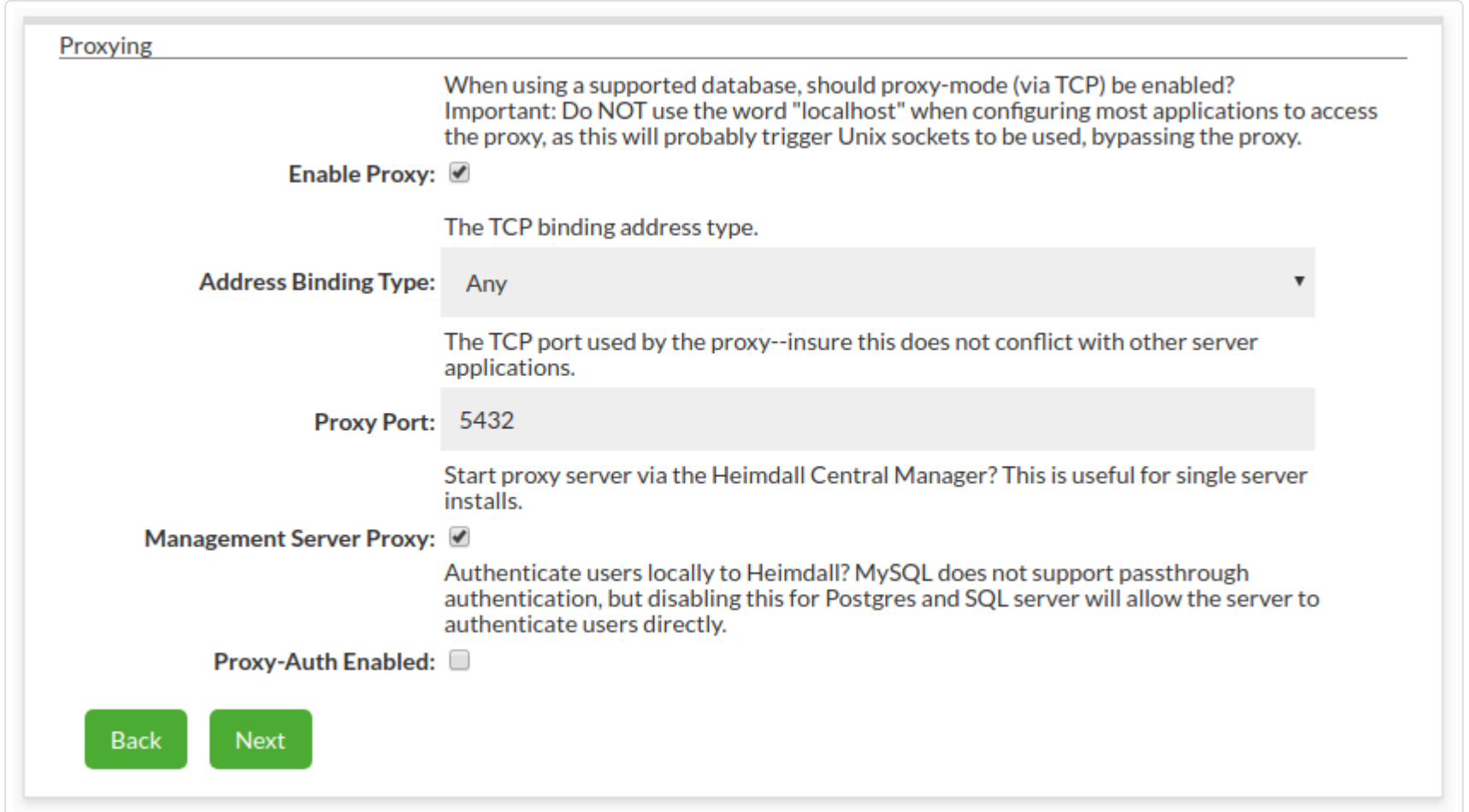
Here, the various options for each cache type (if caching is desired) can be configured, e.g.:





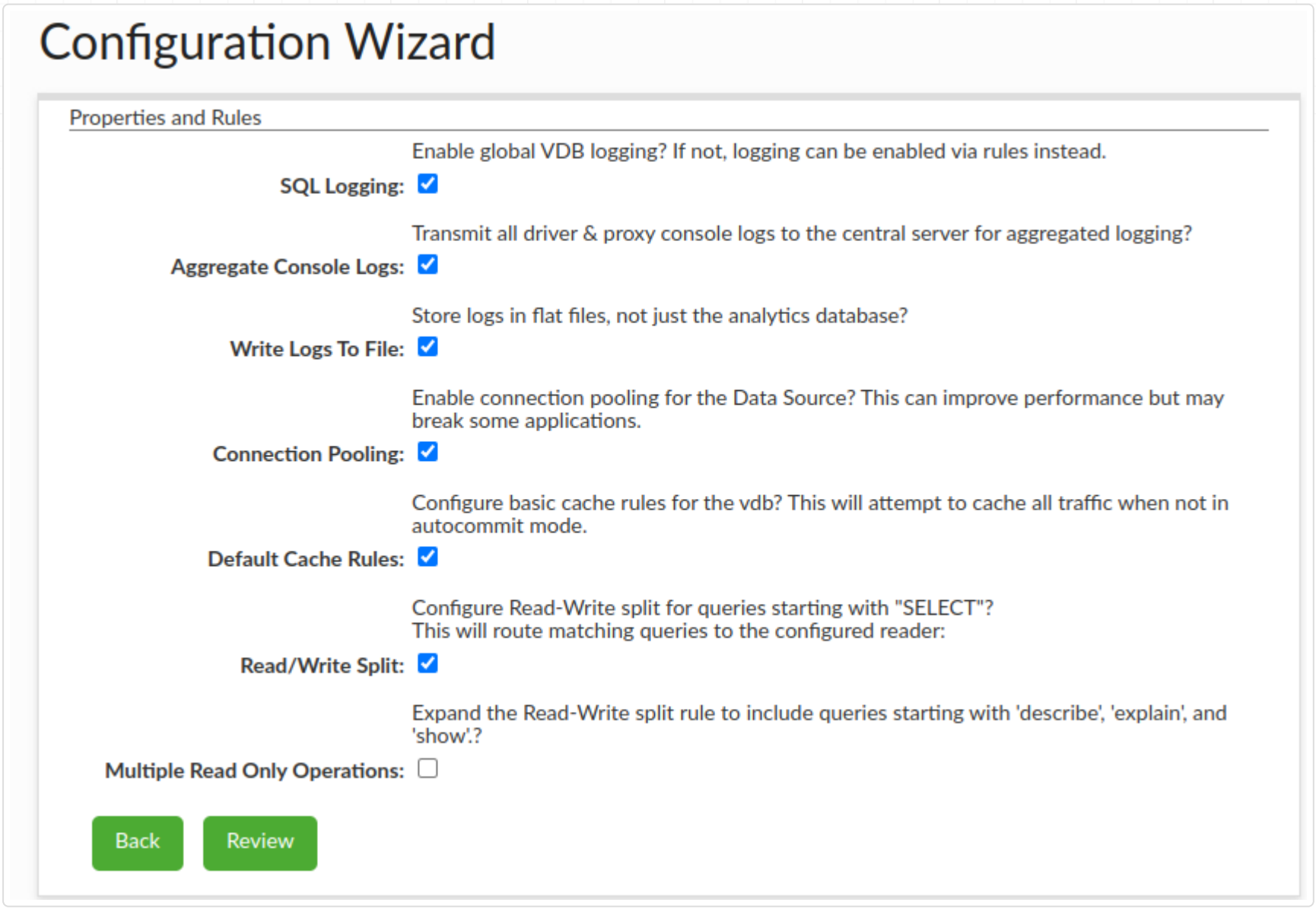
Proxy Configuration

When the data source supports a proxy (i.e. not Oracle or other JDBC only data sources for example), then this next step allows for the configuration of the proxy settings. If Active Directory support is used or more than one proxy port is desired, please configure this in the VDB->Proxy settings after the wizard is complete.



Properties and Rules

This next section allows a variety of pre-defined rules and properties to be set. Any setting can be adjusted or disabled after configuration, so it is generally recommended that they be left enabled unless the feature is known to cause problems or is not desired.



Review Summary

Next, the overall configuration can be reviewed for correctness:

Summary

Data Source

Configuration Name: db-cluster-name

Database Type: PostgreSQL/Aurora (Postgres)/Greenplum/EnterpriseDB

Database Driver: PostgreSQL

Read/Write JDBC URL: jdbc:postgresql://127.0.0.1:5432/${database}

Read-Only JDBC URL: Optional

Database Username: postgres

Database Password: ••••••••

Load Balance/High Availability

Track Replication Lags: ☐

Lag Window Buffer (ms): 10000

Caching

Cache Type: Local Only (no distributed invalidation)

Proxying

Enable Proxy: ☑

Address Binding Type: Any

Proxy Port: 5432

Management Server Proxy: ☑ **Proxy-Auth Enabled:** ☐

Properties and Rules

SQL Logging: ☑

Aggregate Console Logs: ☑

Connection Pooling: ☑

Default Cache Rules: ☑

Read/Write Split: ☑

Query Plan Extract: ☑

Back Next

Submit

Finally, once the configuration has been reviewed, a final section with general guidance will be provided based on the options selected. Select "submit" to provide the configuration to the server.

Based on the selected options, please follow these directions for best results upon submitting the configuration:

- Please test the data source via the "test" button on the sources tab.
- A default cache-all rule will be added to the rulelist db-cluster-name-rules and committed.
- Proxy authentication passthrough has been configured, any configured user on the database should be able to use the proxy.
- You have selected to run a proxy on your management server for use by other servers.
- Do not use the hostname of "localhost" when pointing to a Heimdall Proxy as this will often bypass the proxy.
- You can configure your application to point to the Heimdall instance's IP on port 5432.
- Please verify that the AWS security group or other firewall allows connections on port 5432.
- Console logs from the Heimdall Driver/Proxy will be shipped to the central manager for reporting.
- If you have any difficulty configuring your application, please refer to Application Installation.

Back | Submit | Save to PDF

Modules Overview

Starting with the first full release in July, Heimdall includes a module system for loading features at runtime. As of this release, the modules include:

- **Cache modules**: Hazelcast and Redis
- **Feature modules**: Async execute, Forwarding+read/write split, Load Balancing/HA, SQL Password Extraction and Proxy-side triggers

Note: More modules are likely to be broken out, so consider the modules included in each archive to be the definitive list.

Installing

In order to install a module that is not currently installed, locate the module desired in the install zip file (under /heimdall/static/modules) and through the admin->modules tab, load the module. That is it. In the case of a cache module, the ability to select the cache type will be made available in the VDB tab.

In the case of the default password extraction module, it will leverage a JDBC connection through the VDB itself, allowing all caching, logging, to apply to the password query. To configure the password query, set in the vdb properties a property named "passProvPasswordQuery", with a value such as "SELECT password FROM users WHERE username='?'". The ? will be replaced with the username provided, and the database should return a single value of the password that should be expected.

Updating

Modules are updated automatically along with the other Heimdall components. A module can be updated only if there is a new version with the same major version. For example, if the module is 1.0-18.7.23.1, the major version is "1.0" and the minor version is 18.7.23.1. In the case of a grid cache module, the major version will align with the library version the module is compiled against, i.e. for Hazelcast, we are currently shipping compiled against 3.6.8, but we can provide modules compiled against other versions of Hazelcast as well.

Custom Modules

In particular for the Password extraction module, Heimdall can make available the source code so that custom modules can be compiled and used for your environment. In the event of a bug-fix, we may provide a module instead of a complete release package to minimize the impact on deployment. Modules can be updated at run-time, so we can often resolve issues without requiring a reboot of the system using the modules system.

Module specifics

Please see the rules specific for each action the module provides for more information, or the page dedicated to the module as appropriate

Proxy Modules

- heimdallmysql: enables the MySQL proxy code
- heimdallpostgres: enables the Postgres/Greenplum/Redshift/AlloyDB proxy
- heimdallsqlserver: enabled the SQL Server proxy

Cache Modules

- heimdallhazelcast: Hazelcast support
- heimdallredis: Activates Redis cache support

Feature Modules

- heimdallpw: provides a means to extract proxy passwords from the default data source. Example configuration: VDB Property: "passProvPasswordQuery" value: "SELECT password FROM users WHERE username='?'"
- heimdallasync: Async Execution support (see documentation page for more detail)
- heimdallauth: Stub module to provide GUI level authentication support, no directly accessible feature, but can be used as the basis for custom authentication modules
- heimdallextractplan: allows query execution plans to be retrieved after executing queries
- heimdallforward: forwards queries from one data source to another based on rules
- heimdallloadbalancer: activates the LB and HA functionality of Heimdall (see LB documentation page for more detail)
- heimdalltrigger: allows triggering of actions based on matching rules

Management Server Overview

The Heimdall server operates using a system call Spring Boot, which provides the basic infrastructure for managing connectivity, api calls, etc. As such, settings that apply to Spring Boot are used to configure aspects of behavior such as the TCP port used as well as TLS configuration

Changing TCP Port

To change the TCP port of the server, in the install directory, create a file named "application.properties" and in it, add the following line, adjusting for the desired tcp port

server.port=8087 # Server HTTP port.

When connecting to an alternate port, the driver configuration will also need to be adjusted to account for this.

When configuring a proxy directly, append : to the hostname in the JDBC URL.

Heimdall.conf

The file is a configuration file and can be used to populate environmental variables or cloud environments. It should be generated at directory: `/etc/heimdall.conf`

It can contain different options which should be echoed out if necessary with syntax: "${optionName}=${optionValue}":

Option Name	Description
`cloudDetection`	If the manager should detect cloud services on startup
`cloudOptions`	(Available when cloud Detection is disabled) For heimdall running on premise to allow using cloud services, initializes on startup (none, aws, azure, gcp, oracle)
`hdHost`	Hostname of management server
`hdPort`	Port of the management server, generally 8087 or 8443
`hdRole`	server | proxy
`hdSecretKey`	In AWS, use this as the name of an "AWS Secret" to store the configuration, protecting included passwords from being written to disk. To use, proper permissions must be set on the IAM role. This option provides two major benefits. First is that all passwords are stored in AWS Secrets, in an encrypted format. Second is that redeployment of a management server can be done with a configuration pre-populated, so there is no need to backup and restore configurations to account for failures. Simply terminate the old instance and a new instance with the same user-data will be created with the same configuration as the original.
`hdPassword`	Login password for the management server
`hdUser`	Login username for the management server, can be admin
`javaOptions`	Any arbitrary options desired to be set
`secure`	If the proxy should use HTTPS to connect to the manager
`vdbName`	Exact name of the vdb to service

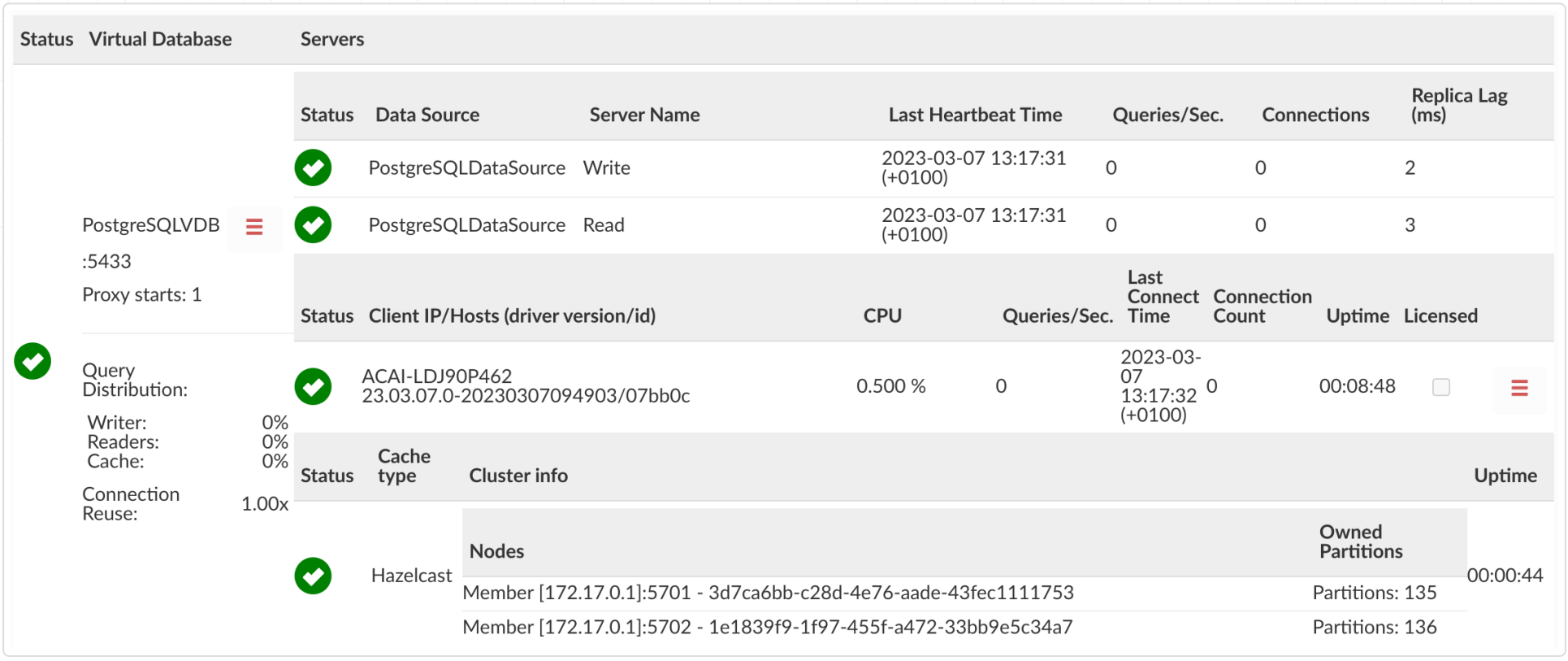
Example user data script:

```
#!/bin/bash
(
echo "cloudDetection=true"
echo "cloudDetection=aws"
echo "hdHost=HostNameOfManagementServer"
echo "hdPort=8087"
echo "hdRole=server"
echo "hdPassword=secretSafeLongPassword"
echo "hdUser=admin"
echo "secure=false"
echo "vdbName=SQL_DB"
) > /etc/heimdall.conf
```

Once initialized, this configuration can be adjusted manually if necessary. Note, if the hdRole is set, then the instance will automatically allocate 80% of instance memory for the process (server or proxy). This can be tuned in the /etc/heimdall.conf as needed as well. These settings will effectively allow auto-scaling groups of proxies to be configured.

Please note: If building an AMI for auto-scaling, which may be used by multiple scaling groups with different configurations, it is suggested that after doing initial testing, the heimdall.conf be deleted so that user-data will be re-read to build the Heimdall configuration. This will leverage the user-data on each new initialization to build the configuration at startup.

Status Overview



The status page provides a quick view of what VDBs are configured, what servers are attached to them, and their current health status, along with what clients are connected to them.

For each VDB, the data sources will be shown, the last HB time an update about that server was received, the current qps, connections and if configured, replica lag is.

For the clients, the pin option allows a given client to be saved and always viewed on the status dashboard. This is to allow the current state of clients that only periodically to be viewed. If the pinned option is not set, then clients not seen for over five minutes will disappear from the status view.

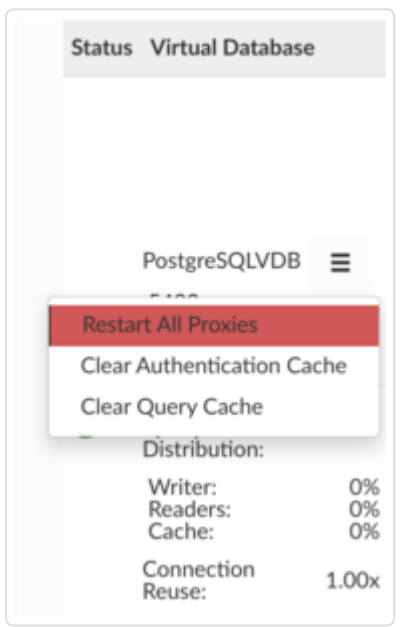
Note: A client MAY show as red if it is using connection pooling, despite traffic being active from the client.

If in the VDB configuration **Cluster manager via cache** is enabled and cluster cache has been used (i.e. Hazelcast or Redis working in cluster mode), the cluster information table will appear, showing cache type, status, list of nodes, partitions owned by each node and the cache uptime.

Note: The Hazelcast instance on the Heimdall Manager should not be responsible for owning any partitions; rather, all partitions ownership will given to the Heimdall proxy nodes.

VDB Actions

When the collapsed menu button is clicked, there should be displayed additional actions showed in the image below.



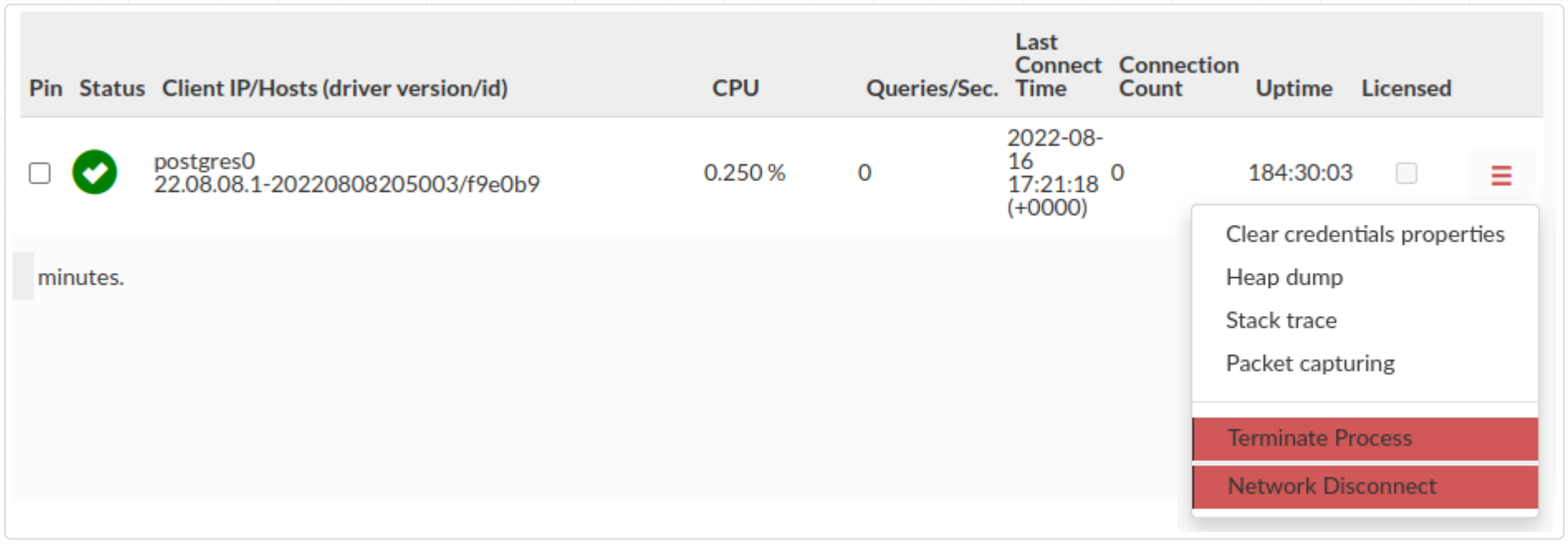
Below are some actions and details what each of the button is used for:

- **Restart All Proxies**: Restart all proxy instances for VDB, on the management server only.

- **Clear Authentication Cache**: Clear all authentication related Caches (users, passwords, AD groups etc.).
- **Clear Query Cache**: Clear Query Cache, if using a shared cache, this may clear all objects in the cache.

Proxy Actions

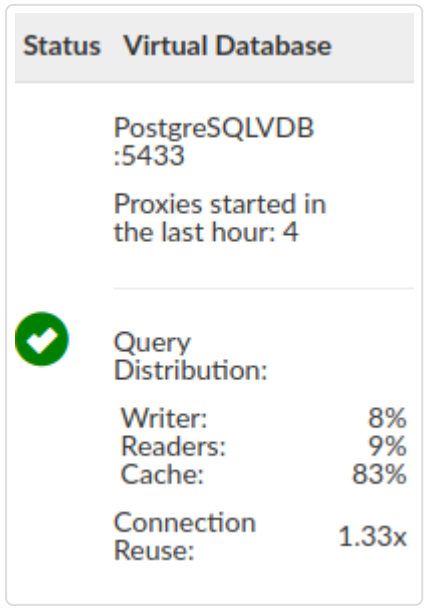
When the collapsed menu button is clicked, there should be displayed additional actions showed in the image below.



Below are some actions and details what each of the button is used for:

- **Clear Credentials Properties**: It will clear the credentials properties cache for a specific proxy. It's usefully when the SQL driven authentication is enabled.
- **Heap dump**: It will make a proxy heap dump, which will be available in Logs tab. You will need jmap to perform this action
- **Stack trace**: It will make a proxy stack trace, which will be available in Logs tab. You will need jstack to perform this action
- **Packet Capturing**: You can initiate a timed packet capture on specific Proxy nodes. This includes limiting duration, total number of files, and the size limit for the capture. Port 8087 and 8443 will be filtered out of the capture automatically.
- **Terminate Process**: It will terminate and restart proxy process - if the proxy is manged by central manager. Otherwise, if you have an independent proxy running, proxy process will be terminated (not restarted).
- **Network Disconnect**: It will disable the network interface of proxy simulating its unavailability. To enable proxy again turning the interface on manually and restarting the proxy is required. Superuser privileges required. Does not work on local proxies, as it's the same as central managers.

Query Distribution



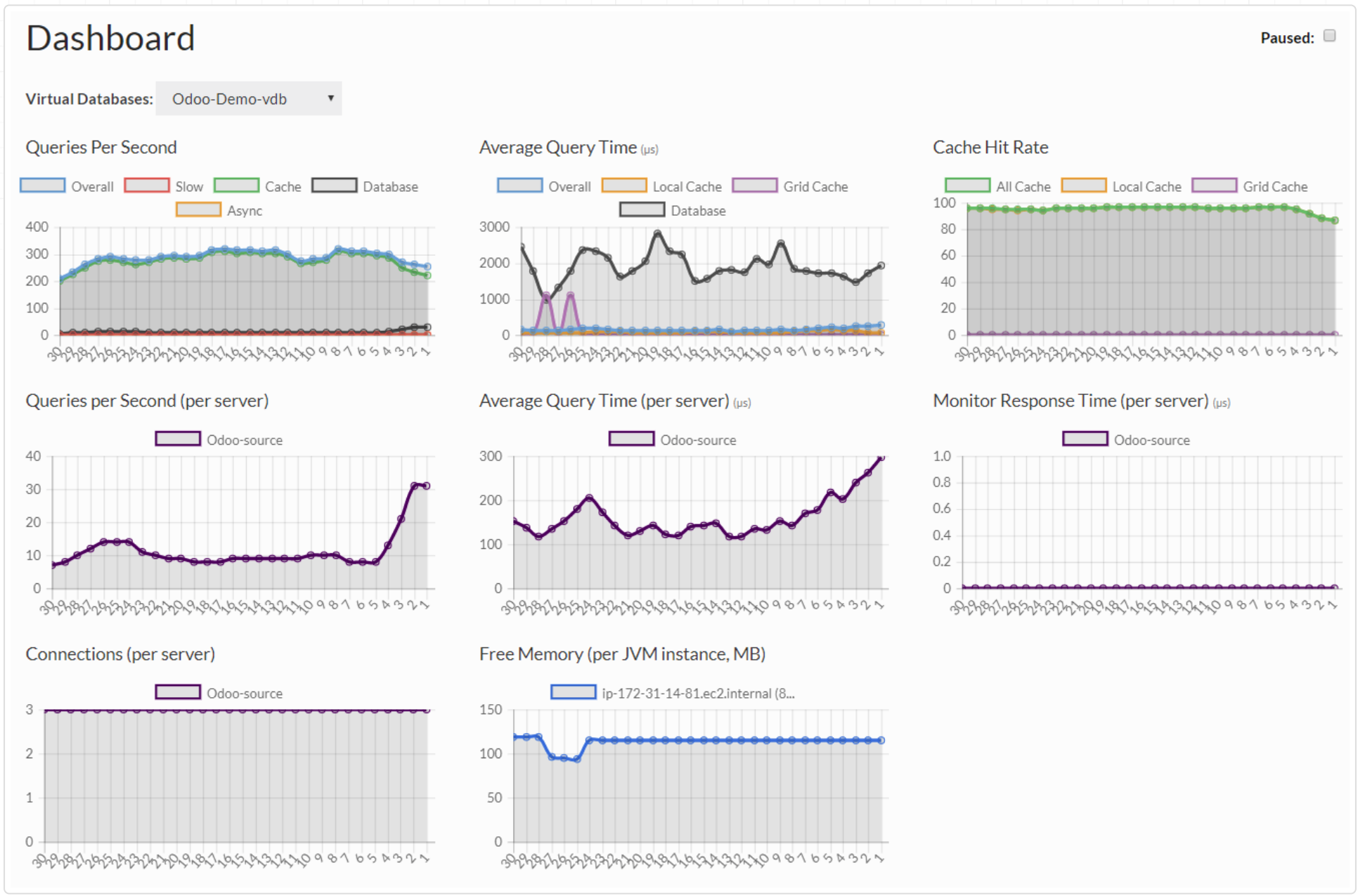
The Query Distribution part of VDB preview provides a quick view of some data aggregated during the proxy runtime. You need to enable certain options to view some of them.

- **Writer** - percentage of all queries that did go to writer.
  This statistic is enabled by default.

- **Reader** - percentage of all queries that did go to reader due Read/Write split.
  To enable this statistic Read/Write Split must be enabled and configured properly.
- **Cache** - percentage of all queries that hit the cache.
  To enable this statistic Caching must be enabled and configured properly.
- **Connection Reuse** - A ratio of how many connections were opened on the front vs on the back.
  To enable this statistic connection pooling must be enabled and configured properly.

These statistics can be reset by restarting the proxy in the Status tab.

Dashboard Overview



The dashboard provides live statistics for many key performance indicators, and can be filtered by VDB. In the case no vdb is selected, then the statistics will be for the resources available to be observed by a user based on their administrative filter, if any.

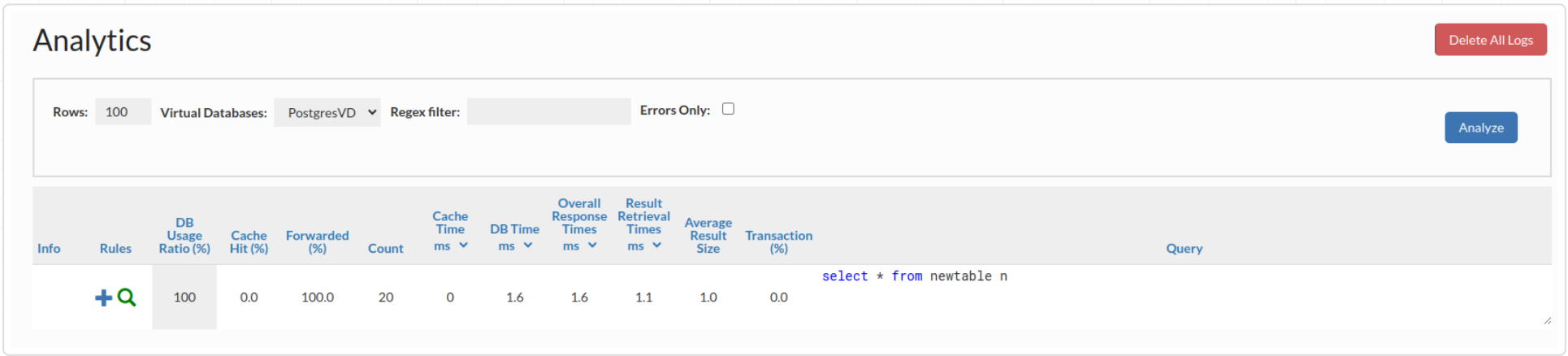
The statistics provided are:

- Queries per second, divided into Overall, Slow (taking longer than 10ms by default), Queries from cache, Database, and Async.
- Average Query time, divided into Overall, Local (L1) cache, Grid (L2) cache, and Database
- Cache Hit rate, divided into Overall, Local Cache, and Grid Cache
- Queries per Second (per Server), showing the overall actual query rate reaching each server
- Average Query time (per server), again showing the average of queries making it to a server
- Monitor response time (per server) showing the monitor query performance, including opening a connection, issuing the query, and closing the connection
- Connections (per server), showing the active or idle connections on each server node at the connection pool (if present) level or actually used.
- Free Memory (per JVM Instance, in MB), the amount of heap available for each node, to help track memory usage issues that may impact caching or the application when in JDBC mode

At the top of the screen, there is a pause button, which will pause rendering of all graphs.

If a particular item is clicked on in the legend of a graph, its visibility can be toggled, so as to help highlight items that may be obscured due to overlapping lines, and/or due to scale issues. The combination of the pause and the item toggle allows each graph to be explored in more detail.

Monitoring Overview



The analytics tab data is provided to allow a per query-pattern view of the queries passing through the system, as logged with the Log action or with global VDB logging.

Filters

Before analyzing the data, one of several filters can be specified to control what data is viewed:

- **Rows**: The number of rows to display. Large numbers may reduce the performance of the browser. Note, if using the regex filter for something in a rare query, it may be necessary to set a large number of rows to get the data desired.
- **Virtual Database**: Drop-down selector of the VDB to filter on
- **Regex Filter**: A filter (done at the Javascript level) to narrow down the queries to observe. The rows must have been returned from the server for this filter to apply.

On clicking the Analyze button, the results will be returned, with the following columns:

- **Info**: This column may show one of several visual indicators. The green page indicates that a Query plan is available to be displayed for the query, and the red circle around an ! indicates that there is an exception registered against this query pattern.
- **Rules**: Provides some information and control over the current rules in effect. The blue + is used to add a new rule based on the query pattern, a green magnifying glass provides a list of rules that match, and the disk icon with a lightning bolt provides an indicator of the cache status of the rule, with green for an explicit regex match, and a red indicating explicit no-cache or a zero ttl.
- **DB Usage Ratio (%)**: The percentage of time overall time waiting on the server represented by this particular query pattern. This column should add to 100% for all queries on a given VDB (rounding excluded).
- **Cache Hit (%)**: The percent of queries a particular query pattern has been served from cache vs. retrieved from the Origin.
- **Forwarded (%)**: The percent of queries a particular query pattern has been forwarded vs. retrieved from the Origin.
- **Count**: How many times this query pattern has been observed.
- **Cache Time**: The average time it took for a cache hit to be pulled from the cache
- **DB Time**: The average time it took for a cache miss to be pulled from the database
- **Overall Response Time**: The average time it took for all queries to be served to the client, blending the cache hit and miss times
- **Result Retrieval Time**: Like the Overall Response Time, but measured from the time the query is sent to the server to the time the result-set is closed. This can include additional time spent by the application processing the result-set, and may be significantly longer than the Query Response Time for very large responses.
- **Average Result Size**: The average result-set retrieval size for this query pattern. Large results generally correlate to larger query times.
- **Transaction (%)**: The percent of time this query pattern was executed in a transaction, i.e. not in auto-commit mode, and/or inside of an explicit transaction. In general, for most applications, this will be 0 or 100%, but in some cases where the same query is executed both inside and outside of a transaction, it may show other values. This can be used to diagnose frameworks that put 100% of queries into a transaction, even when not necessary.
- **Query**: The query pattern itself. Note--when hovering over the query, a pop-out icon will present itself in the upper right-hand corner, and is used to access the extended query view.

Note: Time units for any appropriate columns can be selected in the display itself, to scale as necessary, so are not documented here.

Extended Query View

Each query, when the mouse is moved over, will provide a small icon in the upper right-hand corner. If selected, extended information may be provided including:

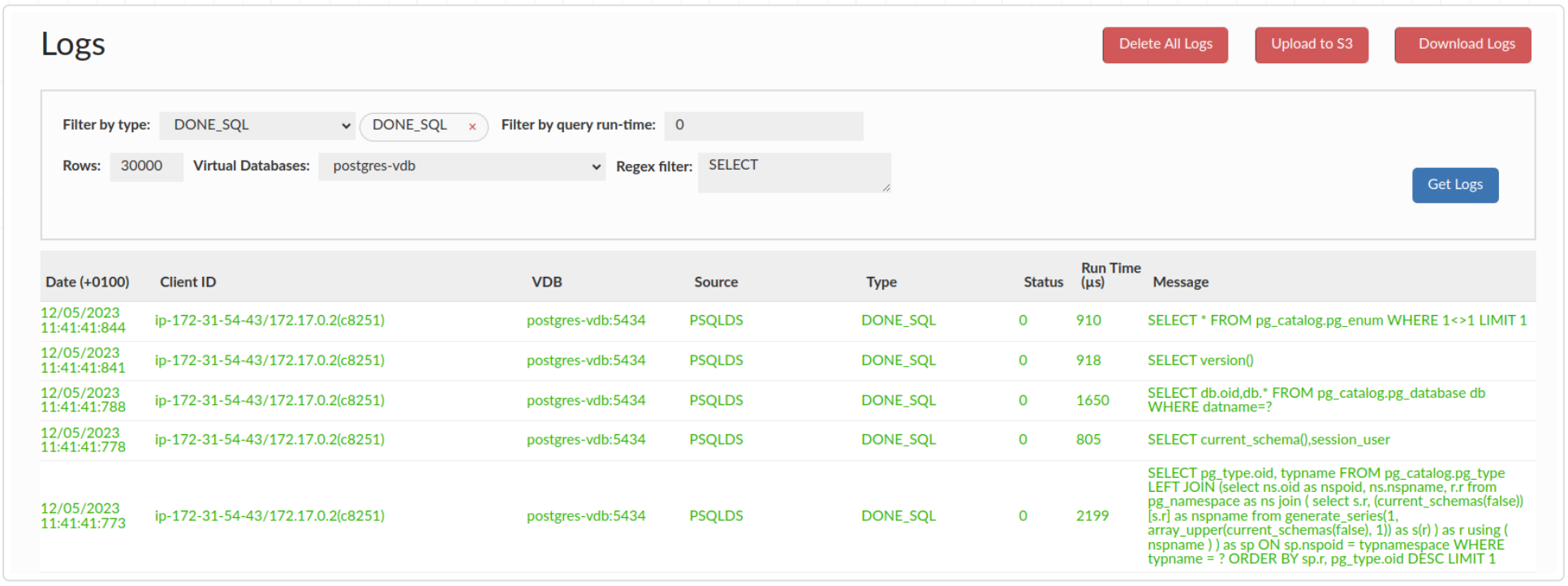
- The query statistics for the query;
- A formatted view of the query;
- The tables detected for this query;
- Any exceptions that have been observed for the query;
- Any saved query plans for this query.

This view is intended to be useful to provide to developers to help diagnose any performance problems that are apparent with the query pattern.

Viewing Errors

The check-box "errors only" will result in a view that only contains queries that have SQL exceptions associated with them.

Logs Overview



The log tab is provided to allow inspection of the actual detailed logs being processed by the system. The table contains the following columns:

- Date (with timezone indicator)
- Client ID: The ID of the client that generated the log entry
- VDB: The VDB (if any) the log is associated with
- Source: The data source associated with the entry
- Type: A categorization of the record type
- Status: An integer field to indicate if a query was in various states (please see the logging section for more details)
- Run Time: The time associated with the event that is logged, if any
- Message: The actual message contained in the log entry

Deleting logs will also reset the Proxy Starts counter in the Status Tab.

Please see the logging section for a more detailed breakdown of system logging.

Logs types

The "Filter by type" at the top of the "Logs" section allows us to filter generated logs depending on what are those related to.

The available logs types are listed below:

- **METHOD_COMPLETE**: Generated each time non-resultset JDBC method is called. "Log Methods" in VDB configuration needs to be enabled to generate such logs.
- **CONNECTION_OPEN**: Generated each time new connection is established. "Log Connections" in VDB configuration needs to be enabled to generate such logs.
- **CONNECTION_CLOSE**: Generated each time any connection is closed. "Log Connections" in VDB configuration needs to be enabled to generate such logs.
- **DONE_SQL**: Generated each time any query goes through proxy. "Log All SQL" in VDB configuration needs to be enabled to generate such logs.
- **TRACE**: Generated with trace actions.
- **TRANSACTION_START**: Generated each time transaction is started.
- **TRANSACTION_END**: Generated each time transaction is committed or rollback.
- **DEBUG**: Provides additional logs generated in rare cases. "Verbose Debug Mode" in VDB configuration needs to be enabled to generate such logs.
- **SQL_EXCEPTION**: Generated each time SQLException is thrown.
- **INTERNAL_EXCEPTION**: Generated each time any other exception is thrown.
- **NEW_PATTERN**: Generated when observing a new query pattern. "Learn Pattern" rule needs to be set to generate such logs.
- **NOTIFY**: Generated for rules tagged with a "notify".
- **CONSOLELOG**: The most basic type of log in Heimdall.

- **NOCACHE**: Generated when cache revalidation fails.
- **PROXY_START**: Generated each time any proxy is started/restarted.
- **AUTHENTICATION**: Provides information about each user authentication attempt. "Log Authentications" in VDB configuration needs to be enabled to generate such logs.
- **ERROR**: Generated when something unexpected or unwanted happened, but no exception was thrown.
- **SYSTEM_STATUS**: Generated for Heap Dumps and Stack Traces.
- **METRICS_DISCOVERY**: Generated for JMX statistics propagation between central manager and proxy.

Log File Rotation

When a log file is rotated, as controlled by settins in the admin->system properties, it will execute a command (if present) named rotatelog.sh. A very simple example would be:

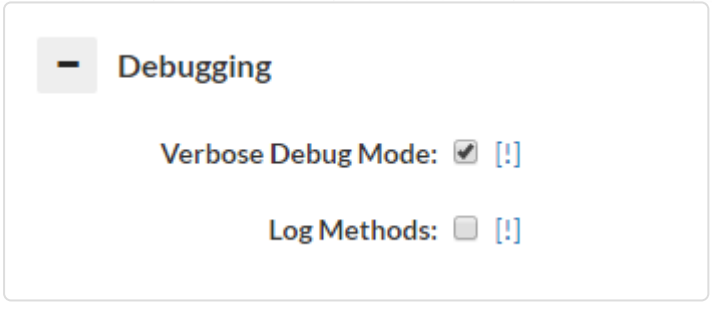
```
#!/bin/bash

tar cvzf $1.tgz $1
```

This command can execute other code, such as to transfer the file to S3 or another location. Please note that on a restart, there is no guarantee that a file will be rotated, so periodic cleanup may be needed.

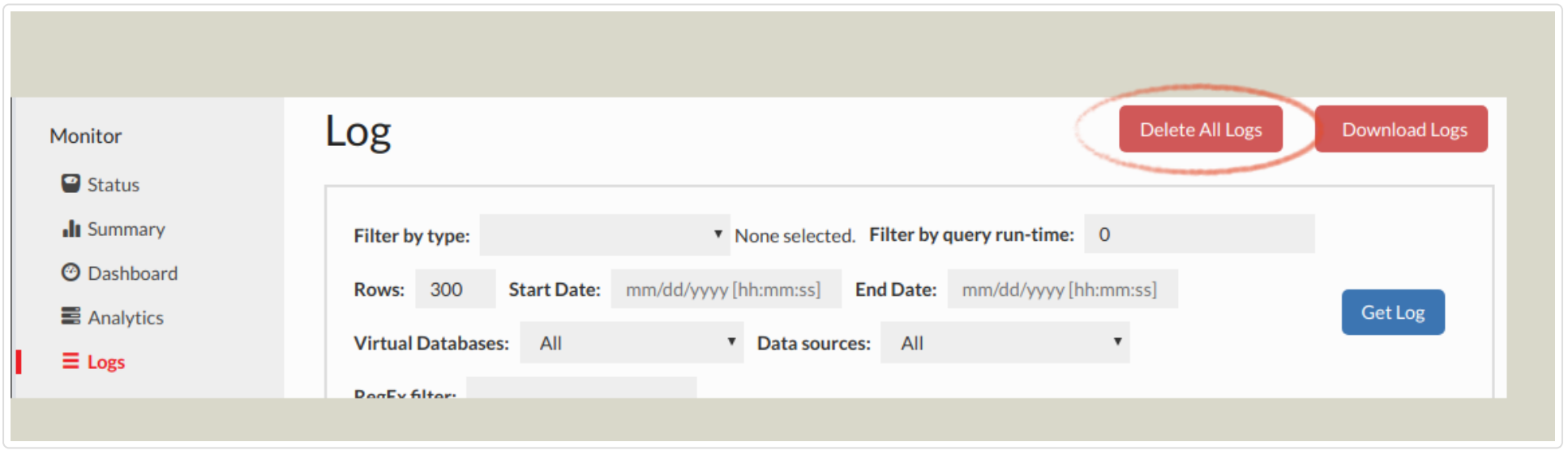
Debugging Overview

If there are issues that require Heimdall support assistance, please follow the following directions (as appropriate)--please note, most debugging should be done in low-traffic scenarios for best results.

- If the queries being made are particularly sensitive, enable "paranoia" mode in the vdb->advanced advanced features section (this will hide the actual SQL being generated, so should only be enabled if necessary).
- Enable verbose debugging on the VDB experiencing the issue (Note, this will add some extra load on the proxy and central manager, in particular if under high load)



- Delete the logs prior to reproducing the issue:



If a packet trace is requested, please follow one of the next two steps as appropriate:

- **On Linux/Mac/Unix systems:** If requested, please capture packets on the system the proxy resides on, including the ports for both the proxy port and the port the database resides on.

Using tcpdump (on most Unix-like system):

```
tcpdump -iany -s0 -wcapture.pcap tcp port <proxy port> or tcp port <database port>
```

If both ports are the same, i.e. 1433 for SQL Server, then only one port is necessary, i.e.:

```
tcpdump -iany -s0 -wcapture.pcap tcp port <proxy port>
```

Note: capture.pcap will be the filename saved--it may be necessary to download this using a tool such as scp, winscp (https://winscp.net/eng/download.php), or filezilla (https://filezilla-project.org/) in order to e-mail this file

- **On Windows:** Install Wireshark or a similar packet capture tool (https://www.wireshark.org/download.html) and start a packet capture for the proxy and database ports. Note: If the proxy is on the same system as the application and/or database, then capturing the "loopback" interface is necessary. Please contact support for assistance if necessary with this.
- Reproduce the problem, preferably with a minimum amount of traffic being generated, and note the time the reproduction was successful, including timezone, to include in the problem report
- If necessary, terminate the packet capture (CTRL-C for tcpdump, or the stop button in Wireshark)
- Disable debug mode in the VDB

- Download the logs from the Log or Analytics tab, and e-mail them and any packet capture generated to Heimdall support, and include a description of the problem, any screenshots that are relevant of the behavior. If the traces or logs are too large (generally ~20MB), please use a file sharing site such as Google Drive or Dropbox to share a link to the files for analysis.

Heap Dumps

At times, if there is a memory utilization issue, Heimdall support may ask for a live or full heap dump of the server or the proxy. Here are the steps to perform this:

1. Connect to the Heimdall instance command line as root, typically via Putty or SSH
2. use "ps -ef | egrep "heimdallserver.jar|heimdalldriver.jar" to find the pids of the processes
3. For a full dump, use the command: "jmap -dump:file=heap-full.hprof "
4. For a live dump, use the command: "jmap -dump:live,file=heap-live.hprof "
5. Download and provide the files to Heimdall support as appropriate based on size, i.e. via a file transfer site, or e-mail

The difference between a life and full dump is that a live dump only shows objects that are not pending cleanup. Both may be requested, so we can see what memory is being taken by objects yet to be cleaned up.

Stack Trace

If a Heimdall process is getting "stuck" at any point, which could be from a logic deadlock or similar issue, a stack trace can be used to find where in the code the lock is happening. To gather a stack trace, here are the steps:

1. Connect to the Heimdall instance command line as root, typically via Putty or SSH
2. Use "ps -ef | egrep "heimdallserver.jar|heimdalldriver.jar" to find the pids of the processes
3. Use "jstack > stack.txt"
4. Download and provide the file to Heimdall support

TCP Keepalive

In order to control idle connections better and prevent NLB timeouts, Heimdall activates TCP keepalives, and sets the keepalive behavior to be one keepalive every 300s while idle. This should be under the 350s that is hard-set for AWS NLB. In other environments, the idle timeout should be set higher than this if keepalive is desired to keep alive connections, and under this value if idle connections should be culled.

Cache Debugging

One of the most common issues when implementing Heimdall is the lack of cache hits. There are many different reasons why with a default cache policy as generated by the configuration Wizard, cache hits will not be observed. Each of these will be detailed below and how to diagnose them.

Show queryinfo

When using a tool that performs simple queries (i.e. not a prepared statement query) such as psql, after a query is executed, the command "show queryinfo" can be executed, which will provide information about the last query, and how it was processed, specifically the properties that were attributed to that command. An example output:

```
gpadmin=> show queryinfo;
      Attribute       |        Value
----------------------+----------------------
 cache:ttl            | 3600000
 all:stop             | false
 all:resultQueryTrans | true
 all:querytimeout     | 0
 all:printtiming      | false
 all:printtables      | false
 all:printresults     | false
 all:printmeta        | false
 all:printmatch       | false
 all:printcapture     | false
 all:onlytrans        | false
 all:olderthan        | 0
 all:notrack          | false
 all:maxburst         | 10
 all:logger           | hdlog
 all:log              | true
 all:invalidate       | true
 all:capture          | true
 nocache reason       | cache is not enabled
 connid               | 13
 command              | select 1
 cache                | true
 Autocommit           | true
                      |
(24 rows)
```

Here, we see the "nocache reason" is set to "cache is not enabled", which is the first reason encountered that prevented caching of this query. If we resolve this issue, then we get:

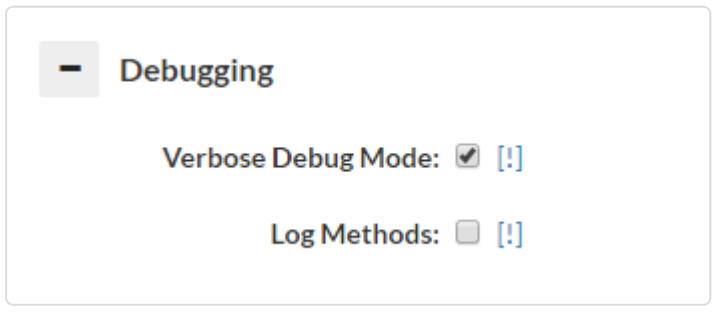
```
gpadmin=> show queryinfo;
      Attribute       |      Value
----------------------+------------------
...
 nocache reason       | empty table list
...
```

Here, an empty table list is encountered, which indicates that caching was not enabled because no table names were extracted from the query, which defaults to disabling caching. See below for a list of nocache reasons.

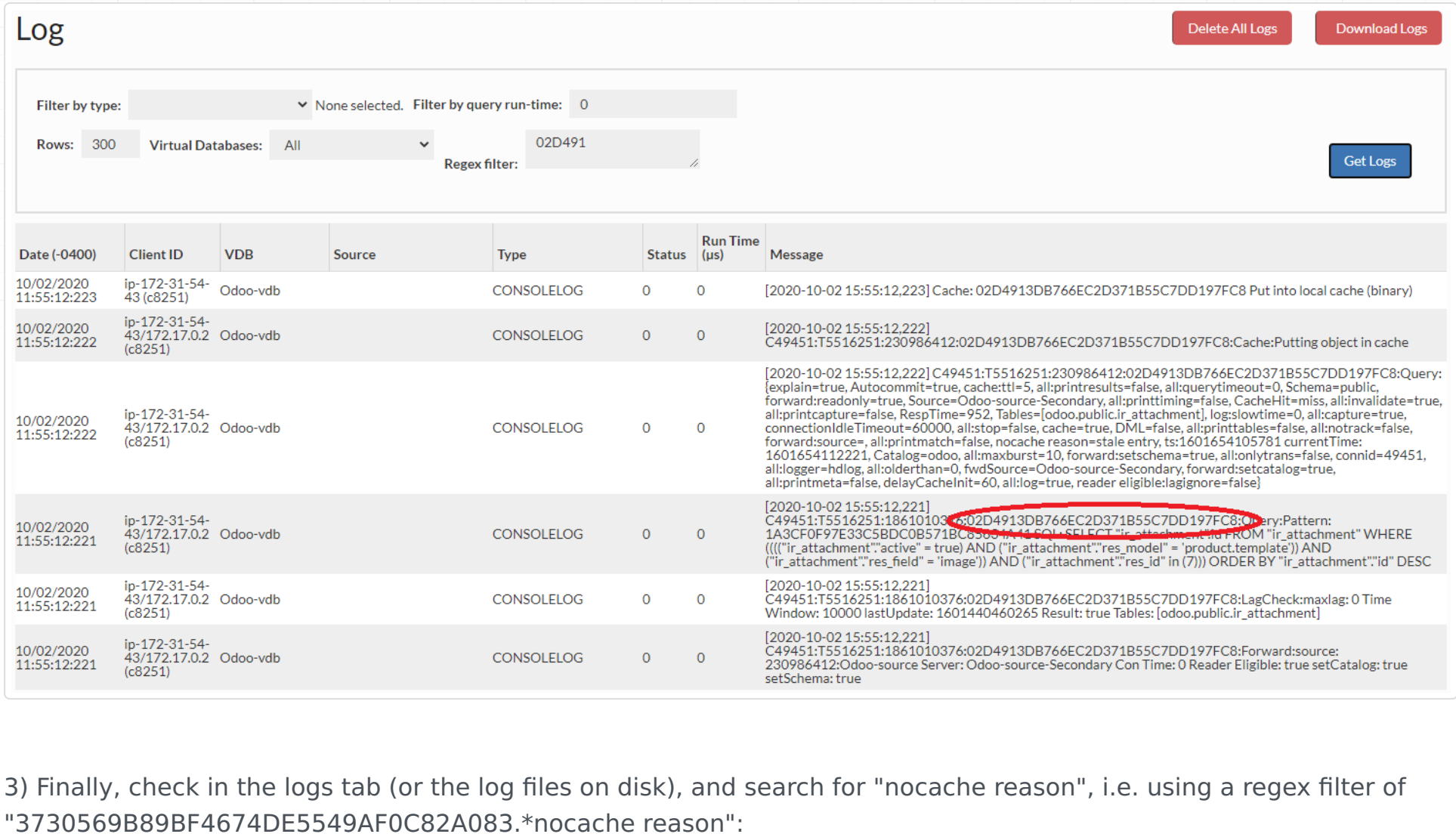
General Debugging

First off, when debugging caching, the following steps should be followed:

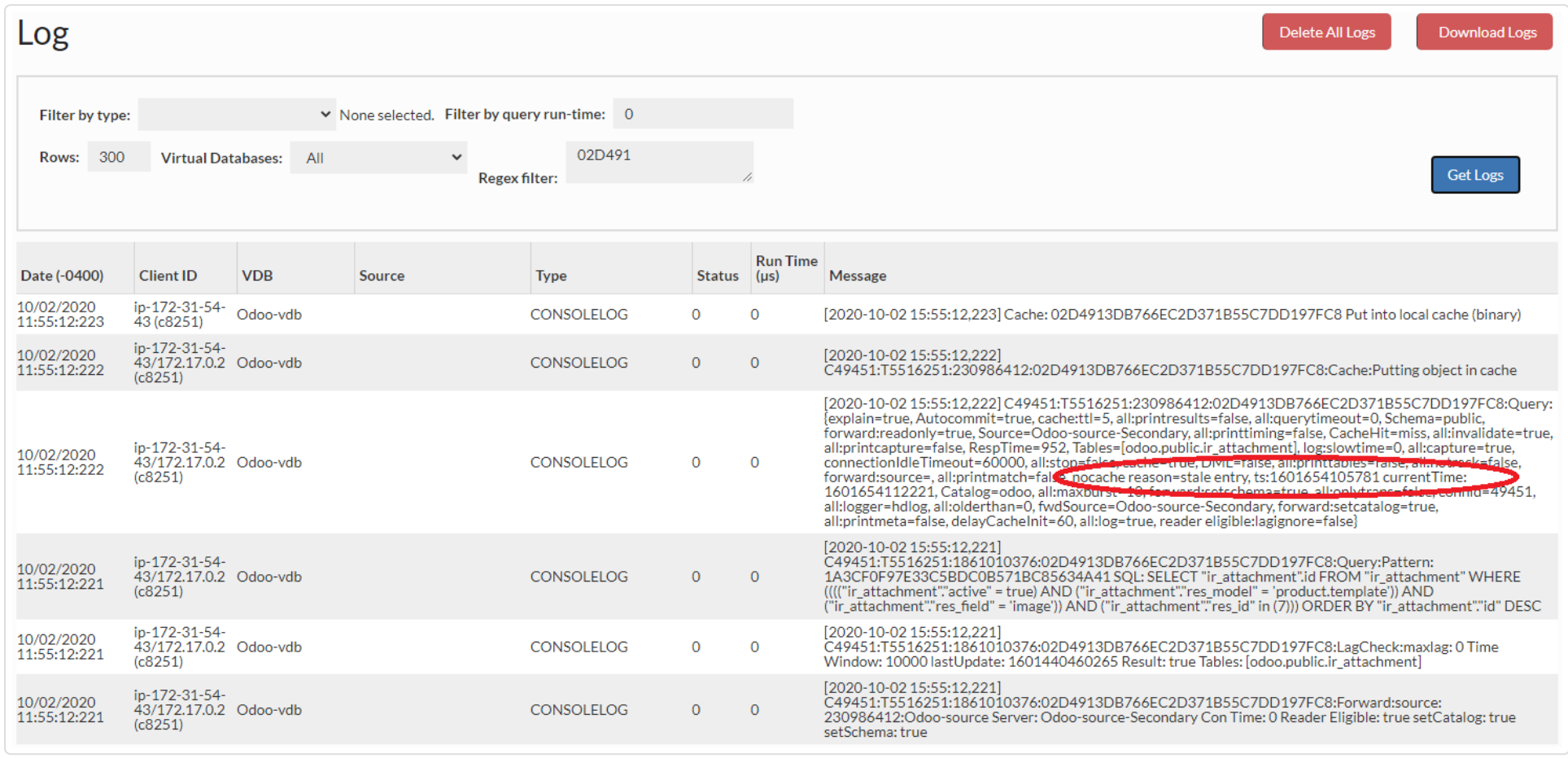
1) In the vdb tab, enable the "verbose debugging" option:



2) Next, pass traffic you expect to be cached, and find the query hash value in the log tab (each unique query will have a unique hash):



3) Finally, check in the logs tab (or the log files on disk), and search for "nocache reason", i.e. using a regex filter of "3730569B89BF4674DE5549AF0C82A083.*nocache reason":



The nocache reason will provide a simple explanation in most cases of why a particular request was not served from cache. Below are some of the cases and details on how to trigger caching, as appropriate (this is not guaranteed to be a comprehensive list):

- **No "no-cache" reason provided at all**: If no cache rule is present or matching, than no no-cache reason will be provided
- **multiple result sets**: In this case, a single query is generating multiple results, which is currently non-cachable. Resolution: none
- **Output Parameters returned**: A called stored procedure is called with a return result. As this is likely to have side-effects, it is non-cachable. Resolution: none
- **cursor execute|fetch|open...**: SQL Server cursor execute result. As this may be scrollable, it is non-cachable. Resolution: none
- **Explicitly nocache per rule**: A no-cache rule is configured. Resolution: Remove or adjust the nocache rule

- **System table, without cache override**: A Database internal table has been detected, and is deemed to be non-cachable, as it may change behind the scenes without notice. Resolution: On the cache rule, you can specify the "cachesystem" property with a value of true to enable caching.
- **empty table list**: If a query doesn't have a detected table associated with it, then it won't be considered cachable. Example "select 1". Resolution: To enable caching such queries, create a rule with the property of "tables" and specify "none", which will provide a special behavior of instructing the system to allow caching anyway.
- **dml**: If Heimdall detects DML or otherwise doesn't flag it as NOT DML, then this will be set. This may be due to it being a "select...for update" query, for example. Resolution: To override the DML flag, the property of "update" and with true or false, depending on if it should be considered a DML operation or not.
- **Result-set properties disallow caching (not forward only or (scroll-insensitive && read_only))**: Certain result-sets may not be compatible with caching, due to scrolling or updatable flags. Resolution: none
- **cache is not enabled**: The cache is not enabled at the VDB level. Resolution: Enable the cache
- **vdb reset timer**: Via the UI or API, a timer was set to disable caching until a particular time. Resolution: Wait until the time expires
- **table reset timer (internal)**: A DML has triggered a cache invalidation interval on this proxy. If a table is written to more than once every two seconds, this may be a continuous issue for the table. Resolution: Wait until the time expires or stop writes to the table at issue.
- **table reset timer (external)**: Like with the internal invalidation, but due to invalidations from an external source, i.e. another proxy.
- **SQL contains (), use unconditional flag to override this logic**: Detected () in the SQL query. To override this logic, select the action cache and set unconditional flag as true for the given regex.
- **sequence usage detected**: Sequence usage has been detected. This logic can be overridden with unconditional flag in rules configuration.
- **temp table**: A temporary table was detected as part of the query. Resolution: Avoid using temporary tables.
- **non-deterministic function call**: A function was detected as part of the query that was determined to be non-deterministic by the database or cache engine. Resolution: Avoid using non-deterministic functions for queries that need to be cached.
- **nocache reason in the query, use unconditional flag to override**: Certain patterns in a query will trigger nocache behavior, such as "sql_calc_found_rows" and "()". To cache such queries, the property of "unconditional" can be set to true to bypass these checks.
- **exception creating fixed cached rowset impl**: An unexpected error was encountered creating a copy of the result-set
- **grid cache is null**: For some reason, the grid cache is not enabled. This may indicate a bug or other error. Consult Heimdall support for assistance
- **grid cache is not ready**: This may indicate that something is wrong with the cache engine selected, i.e. Redis is selected, but the cache server is unavailable.
- **no cached object found for key**: In general, this indicates that there was no cached object for the query. Resolution: retry the query again
- **No matching cache rule or nocache rule was applied**: This may indicate that there are no matching rules that will allow caching. Resolution: Create and apply caching rules or modify an existing one to match the requirements.
- **Cached object removed due to table eviction of table, ts=**: There was a cached object, but the table has had an eviction event, and is being evicted.
- **stale entry**: The cached object has surpassed it's allowed TTL.
- **global cache reset since stored**: On the UI or via API, the clear-cache call was made, triggering a global eviction of objects. Any objects pre-dating this event will be considered stale.
- **ttl value set to 0**: A ttl value was set to zero for a cache rule

Queries in Transactions

As queries in a transaction may have different side-effects than those that are not, so, by default, the wizard creates a rule that does not trigger caching or cache retrieval while in a transaction.

This cause will NOT have a nocache reason (it simply won't be shown), as it is triggered in the rule match process itself.

As queries in a transaction may have different side-effects than those that are not, by default, the wizard creates a rule that does not trigger caching or cache retrieval while in a transaction.

System Table Detected

Certain table names such as pg_* will be flagged as being system tables, and which may be changed behind the scenes by default. In these cases, the default behavior is to disable caching. This can be overridden with the cache parameter of "cachesystem".

No Tables Detected

As invalidation is tracked at the table level, if no table is detect in a query, the default behavior is to disable caching. This includes for queries such as "Select 1". This can be overridden by tagging the query with a table of "none". This is a special value, in that it explicitly removes this tracking behavior.

Table Was Written To

As a DML is detected, the table will enter an invalidation window, which is a period of two seconds during which neither cache hits nor cache stores will occur. This includes for the entire duration of a transaction the DML is detected within, until a commit or rollback. Once the DML is completed, the invalidation window will no longer be in effect, however, for remote nodes, this window will occur for at least the full two second period. This is to reduce the number of messages traveling between nodes. Updates to refresh the invalidation window will occur every second until no dml is pending against the table. Updates can be made that don't trigger an invalidation window by using the parameter "invalidate" with a value of false.

Temporary Table Detected

When a temporary table is detected in a query, this will suppress caching as well.

Non-Deterministic Function Call Detected

With SQL Server, function calls are parsed, and non-deterministic functions are detected. When present in the query, this will trigger cache suppression.

Debugging read/write

This page describes reasons, why Heimdall might not use reader servers, when load balancing is enabled.

Show queryinfo

When using a tool that performs simple queries (i.e. not a prepared statement query) such as psql, after a query is executed, the command `show queryinfo` can be executed, which will provide information about the last query, and how it was processed, specifically the properties that were attributed to that command. An example output:

```
dbConsole => show queryinfo;

      Attribute       |        Value
----------------------+----------------------
forward:source        | MySQLDataSource
cache:ttl             | 300000
all:stop              | false
all:querytimeout      | 0
all:printtiming       | false
all:printtables       | false
all:printresults      | false
all:printmeta         | false
all:printmatch        | false
all:printcapture      | false
all:onlytrans         | false
all:olderthan         | 0
all:notrack           | false
all:maxburst          | 10
all:logger            | hdlog
all:log               | false
all:invalidate        | true
all:capture           | true
noreader              | forward:readonly is set to false, can't use reader
nocache reason        | cache is not enabled
connid                | 5
cache                 | true
autogenKeys           | true
Tables                | []
Source                | MySQLDataSource-Primary
Schema                | empty
RespTime              | 1779
DML                   | true
Catalog               | myDb
CacheHit              | miss
Autocommit            | true
```

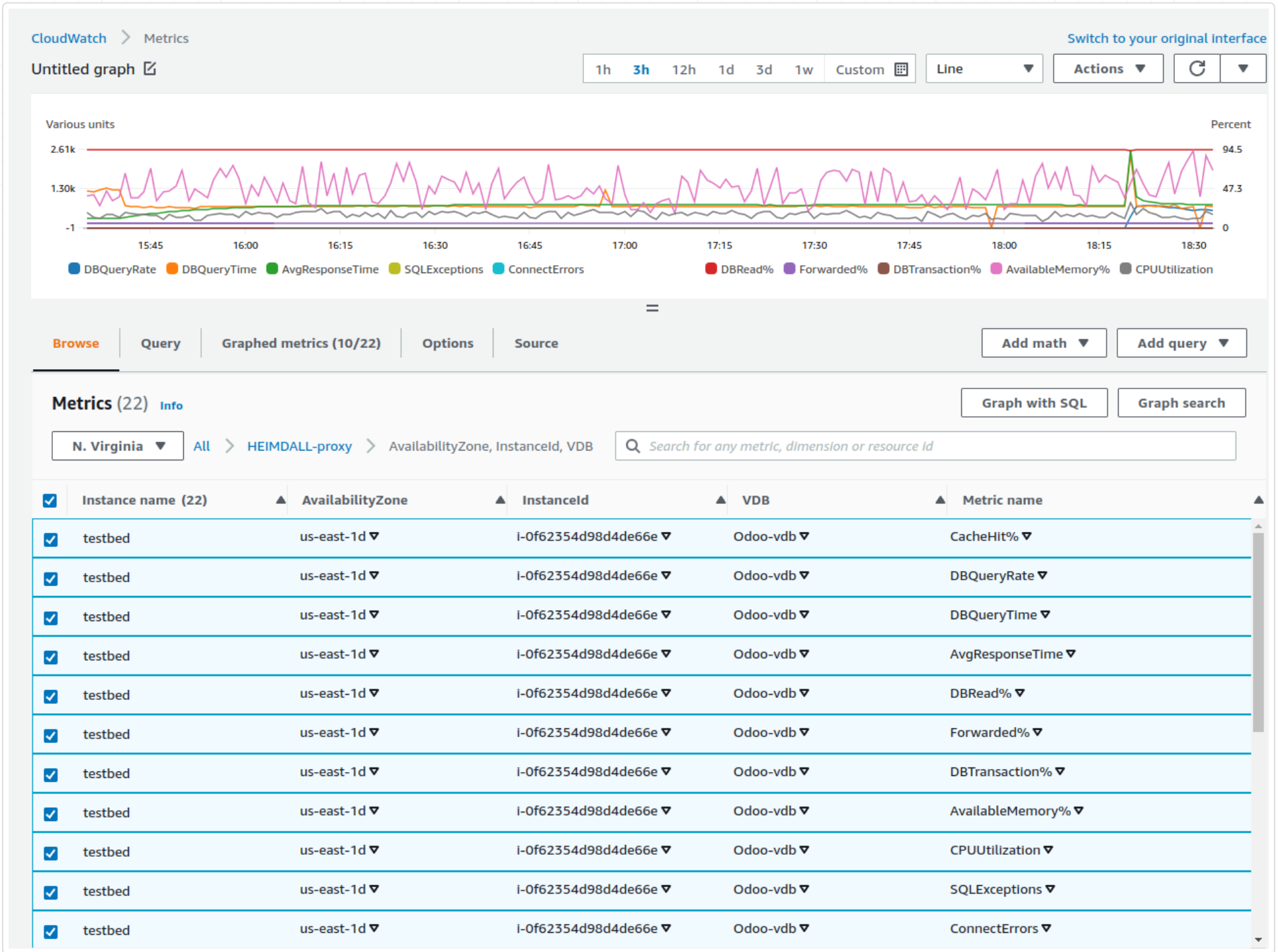
noreader

noreader can have a few causes, they are explained in more detail in list below:

- **forward:readonly is set to false, can't use reader**: When property forward:readonly is set to false, it means that currently Heimdall is connected to writable server.
- **No servers available, cannot forward query to reader.**: There aren't any servers available in Data Source at that moment in time, therefore reader server is not being used.
- **Couldn't get Data Source while forwarding query, reader can't be used.**: Error occurred while retrieving Data Source to forward query. It's good reason to verify existence of chosen Data Source.
- **Connection is set to null, reader can't be used.**: Connection to Data Source couldn't be established.
- **Reader can't be used, exception on setting catalog on forwarded connection.**: Exception on setting catalog on forwarded connection, check catalog/schema on current/parent connection.
- **Statement is null, query cannot be forwarded.**: Forward statement couldn't be created, therefore reader cannot be used.
- **Exception on existing connection, reader can't be used.**: Exception on existing connection, forwarding interrupted, reader is not in use.
- **Connection is set to null, reader can't be used.**: Exception on closing connection existing connection.
- **Statement is DML/DDL type, reader can't be used.**: Statement provided to the reader is DML/DDL, then it cannot be used on the readonly database.
- **Couldn't connect to source, cannot forward query, check data source in vdb configuration.**: Heimdall couldn't establish connection to chosen Data Source, therefore query cannot be forwarded and reader is not in use. Please validate vdb configuration.

- **Couldn't connect to source, cannot forward query.**: Exception on connecting to source, query cannot be forwarded thus reader used.
- **Server isn't reader eligible, reader can't be used.**: Current server isn't reader eligible, so Heimdall won't even try using reader.

CloudWatch



When in AWS, and with cloudwatch integration enabled, various KPI indicators are provided to allow monitoring and alerting

Enabling CloudWatch Metrics from Private IP connections

https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/cloudwatch-logs-and-interface-VPC.html

KPIs

The following metrics are provided:

- **Available Memory**: The percentage of memory allowed to be allocated by the proxy that is available for use. In general, this can change over time dramatically, but if lower than 20% on average for a vie minute interval, an alert should be generated;
- **CPU Usage**: The percentage of cpu time used by this proxy. While multiple proxies may make this metric harder to guage, in a single proxy environment, if this is above 70% for more than five minutes, an alert should be raised;
- **DB Transaction Percent**: The percentage of queries that are considered to be in transctions;
- **Cache Hit Percentage**: The percent of queries that are cache hits. This will vary for each customer, but can be used to set an alarm if caching drops below an expected value;
- **DB Query Rate**: The rate that queries are made against the database, i.e. are not cached;
- **DB Query Time**: The average response time from the database. This can be used to generate alerts if above the expected SLA;
- **Average Response Time**: The average response time for all queries, cached or not. This also can be used to generate alerts if above an expected SLA;
- **DB Read Percentage**: The percent of queries that are reads, which is a pre-requesite for caching or read/write split.
- **SQL Exceptions**: The number (per second) of SQL exceptions generated as a result of executing queries.

- **Connection Errors**: The number (per second) of connections errors.
- **Forwarded Percent**: The percent of queries that are forwarded via a forward rule, or via read/write split. A query may still end up on the writer node depending on configuration however.

Each of these metrics are reported on a per-minute basis, as part of the per-minute logging that is done with other logs to the cloudwatch log channel as well.

A dashboard widget that provides critical CPU usage information with the following query (as of the June 18th build):

```
SELECT max(CPUUtilization) FROM "<specific proxy CloudWatch namespace>-proxy" GROUP BY AvailabilityZone, VDB
```

(if no custom namespace is provided, default namespace will be HEIMDALL-proxy)

This will graph per AZ the top CPU using proxy's CPU load on a per-VDB basis. This allows at a glance the ability to spot if a single proxy is receiving too much load due to uneven distribution of traffic, potentially causing issues. Alerts can then be attached to this to notify operations staff of the issue for corrective action.

JMX Statistics Overview

JMX statistics describe proxy and its minor features performance. If proxy has healthcheck enabled, JMX metrics associated with specific proxy will be exposed to read.


```
{
  "jmxBeans": {
    "Server Performance: com.heimdalldata.PostgreSQL-datasource.Master:type=Statistics,name=ServerPerformance": {
      "TotalCacheTime": "0",
      "AverageTotalCacheTime": "0.0",
      "StddevTotalCacheTime": "0.0",
      "StddevSlowQueriesCount": "0.0",
      "TotalQueryTime": "0",
      "StddevTotalQueryTime": "0.0",
      "AverageLocalCacheTime": "0.0",
      "CurrentLag": "2147483647",
      "AverageDbQueryTime": "0.0",
      "SinceStartedAsyncExecutes": "0",
      "SinceStartedCacheHits": "0",
      "LocalCacheTime": "0",
      "LastActive": "1694003643689",
      "SinceStartedSlowQueries": "0",
      "StddevCacheHitsCount": "0.0",
      "StddevLocalCacheHitsCount": "0.0",
      "AverageTotalQueryTime": "0.0",
      "CurrentOpenedConnections": "0",
      "SinceStartedOpenedConnections": "1",
      "AverageTotalUpdatesCount": "0.0",
      "AverageClosedConnectionsCount": "0.0011904761904761904",
      "ClosedConnections": "0",
      "SinceStartedTotalQueries": "0",
      "AverageOpenedConnectionsCount": "0.0011904761904761904",
```


Endpoints

Healthcheck server handles several types of endpoints:

- **http://{healthcheck-host}:{healthcheck-port}/jmx** - retrieve all JMX statistics associated with specific VDB
- **http://{healthcheck-host}:{healthcheck-port}/jmx/{stats-type}** - retrieve specific group of JMX statistics (example: **/jmx/server-stats** will show data source servers performance stats)
- **http://{healthcheck-host}:{healthcheck-port}/jmx/{stats-type}/{stat-name}** - retrieve specific JMX statistic (example: **/jmx/server-stats/SinceStartedSlowQueries** will show SinceStartedSlowQueries statistic value for every data source server)

Available JMX Statistics Types

- **heimdall-management** - general Heimdall statistics (application/proxy)
- **data-source-server** - data source server configuration basic options
- **server-stats** - data source server performance
- **connection-pool** - connection pool statistics
- **user-pool** - user pool statistics
- **jvm** - JVM hiccup and heap allocation rates

Users Overview

Heimdall currently provides a built-in user database that allows individual users to have defined authentication (password) and login locations defined.

Further, if desired, a user can be configured as a read-only user, and Google Authenticator compatible two-factor authentication.

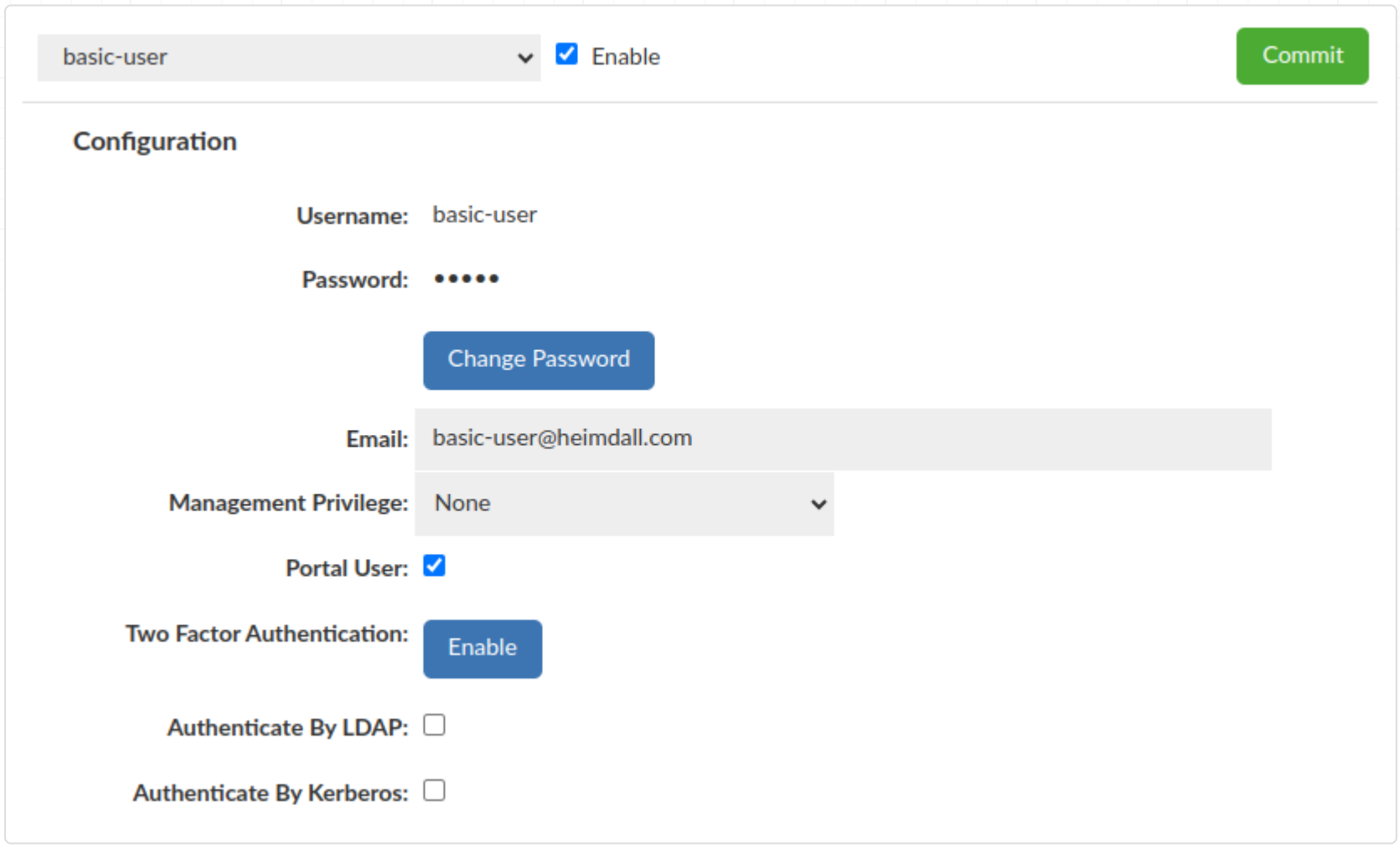
Internally, passwords are stored in the same way that OpenLDAP stores them by default, i.e. Salted SHA. Example, {SSHA}cca9bbffe6879f8367ab681952da2f995bf1668f

Configurable fields:

- **Username**: the login of the user or JDBC connection
- **Password**: The password of the user–please avoid using “:” as it may impact authentication
- **Hostname or IP address**: One of
  - IPv4 IP address in the format defined here
  - IPv6 IP address in the format defined here
  - Subnet address defined with either an IPv4 or IPv6 network address plus “/” and the subnet size
  - A DNS hostname that resolves to one or more IPv4 or IPv6 addresses. If more than one is provided, than any resolved IP is allowed.
  - In the event no users are defined, than unrestricted access is allowed
  - If no hostnames or IPs are provided for a given user, then the user is provided unrestricted access from any network
- **Admin**: If enabled, users will have admin rights, allowing them to perform tasks such as configuring settings, managing users, and accessing all resources.
- **Read Only User**: If enabled, users will be unable to make any changes to the configuration, but they will retain access to resources based on their filter settings. Additionally, it will block access to management tabs, such as Users, Admin, Certificates, and various options like the 'Test Connection' button in the Data Sources tab.
- **Portal User**: If enabled, users will have access to the Portal.
- **Two Factor Authentication**: If enabled, it will present bar-code that can be scanned into the Google Authenticator software, and an account code, which can be used in place of the bar-code. This ID is in addition to the normal password authentication the user will be required to provide.
- **Authenticate By LDAP**: If enabled, user will be authenticated by LDAP server configured in Admin tab.

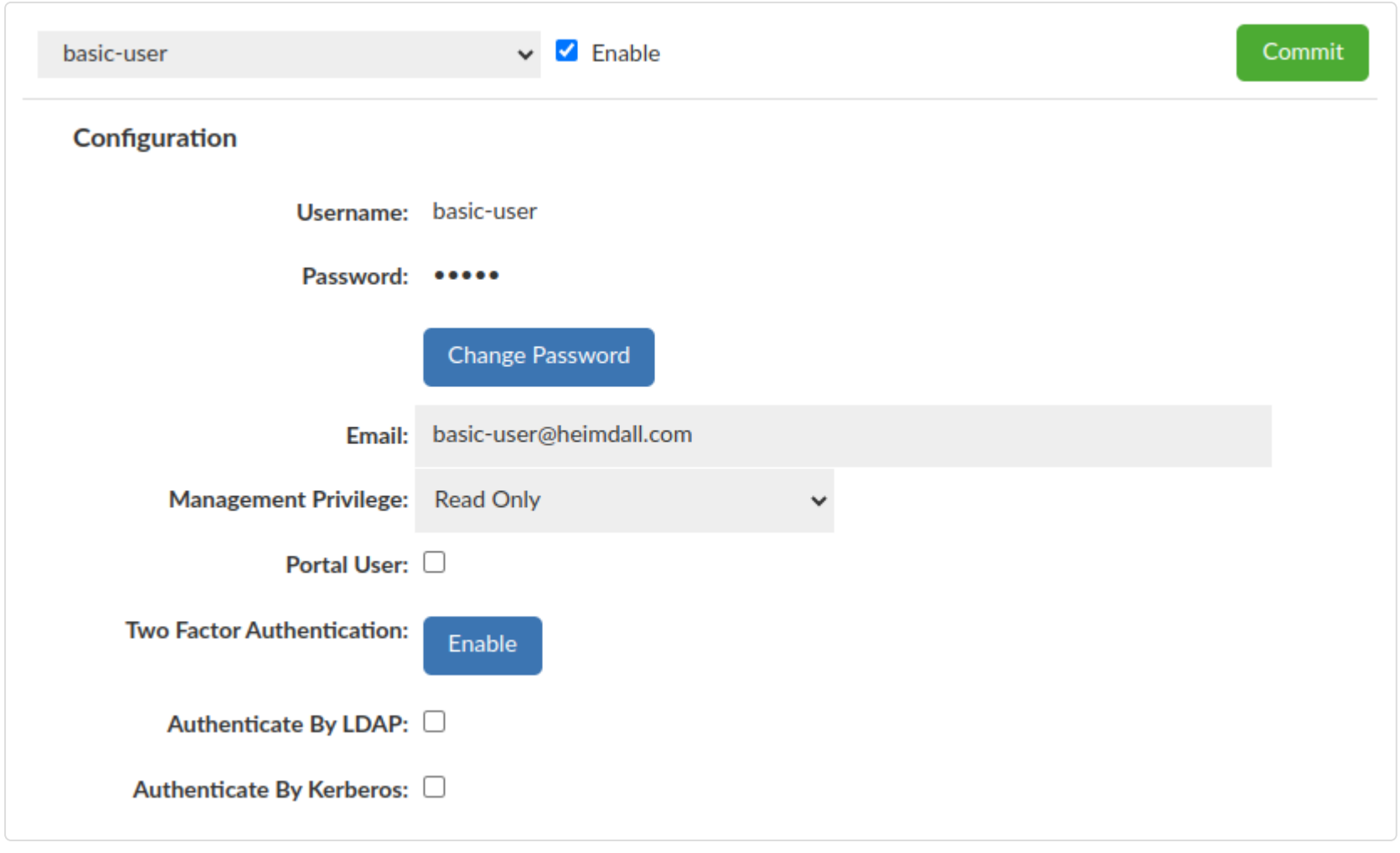
Basic User (with Portal Mode enabled)

Require "Username", "Password" and Portal User enabled



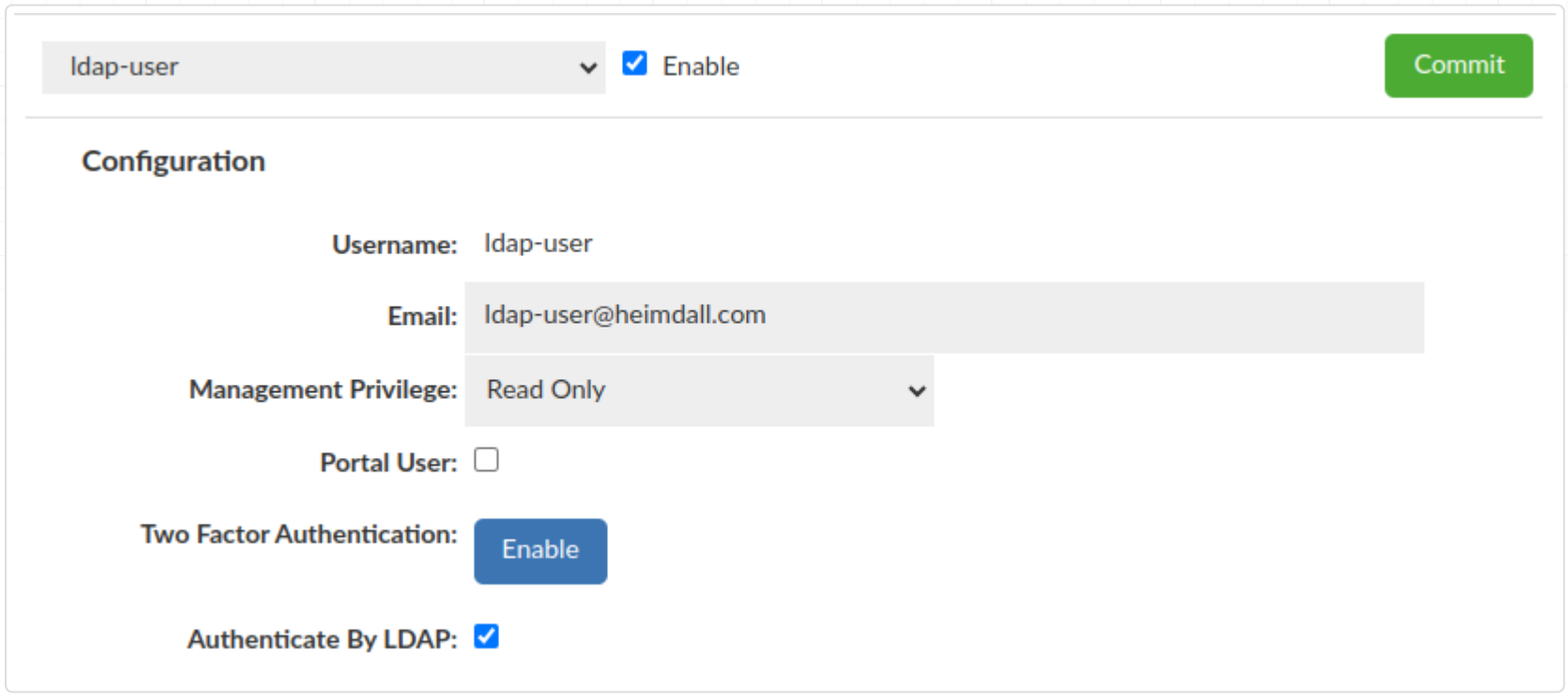
Basic User (without Portal Mode enabled)

Require "Username", "Password" and selected Management Privilege



LDAP user

Require "Username" and "Authenticate by LDAP"



You can set up LDAP Configuration in the admin tab. (Manage section -> Admin tab -> LDAP Configuration)

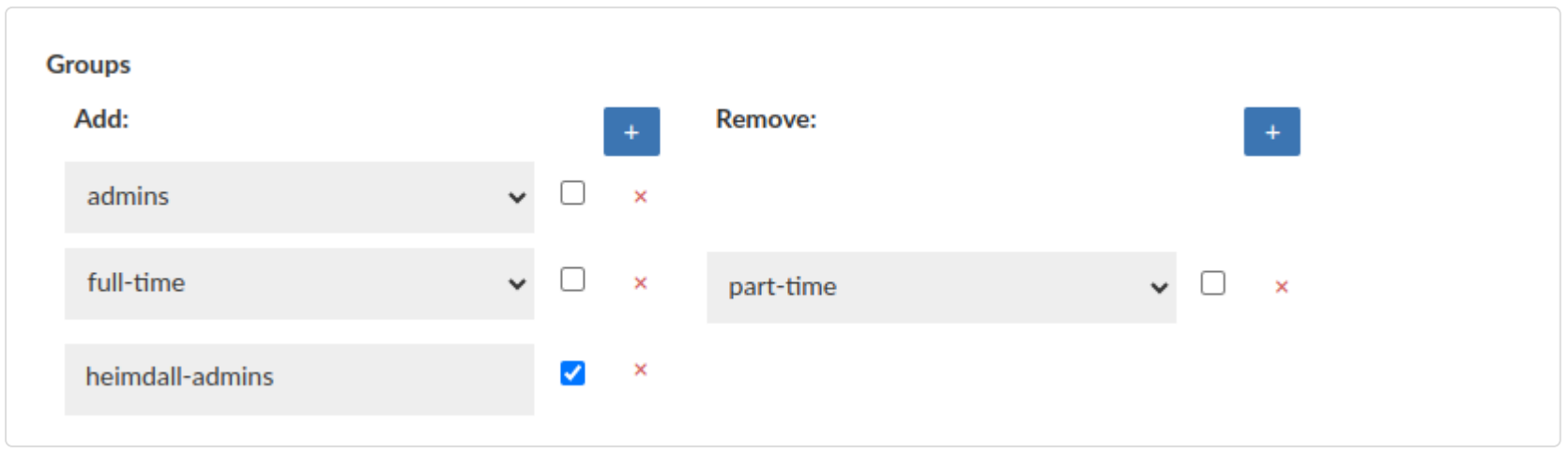
Groups

This feature is available for all Heimdall (GUI) users, doesn't mather whether user authenticates by LDAP or not.

This one allows to grant or revoke a group membership for testing purposes, e.g for LDAP based notifications (Notifications tab in admin tab).

For enhanced security measures, it is imperative that at least one real LDAP Group be extracted when utilizing this feature while using for LDAP authentication testing purposes.

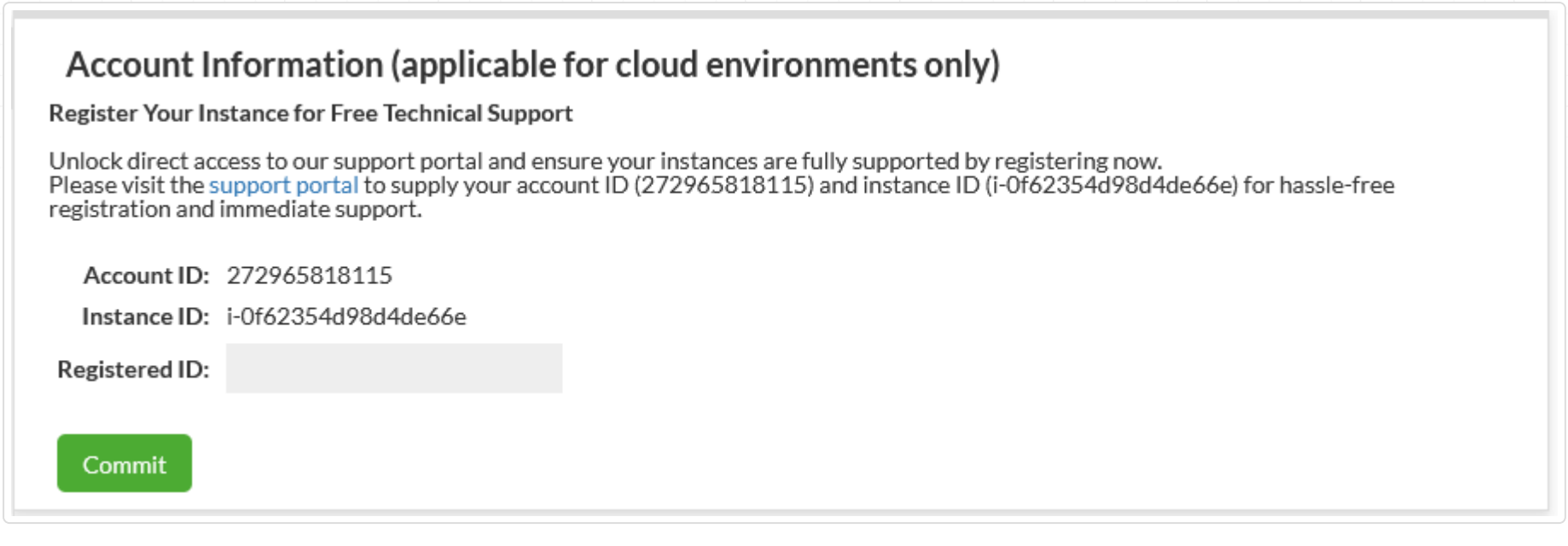
What is important configuration applied there **DOES NOT IMPACT** the real groups membership on LDAP Server.



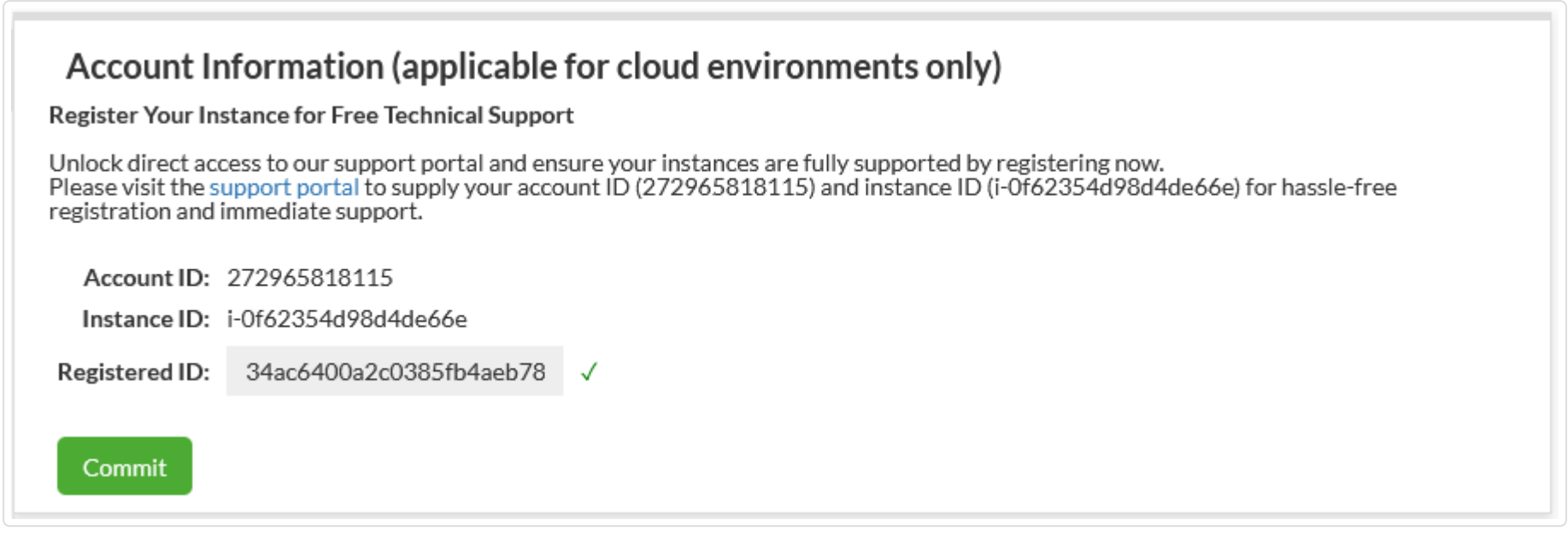
Admin Overview

The administration tab is divided into multiple sections covering a variety of administrative actions, as detailed below.

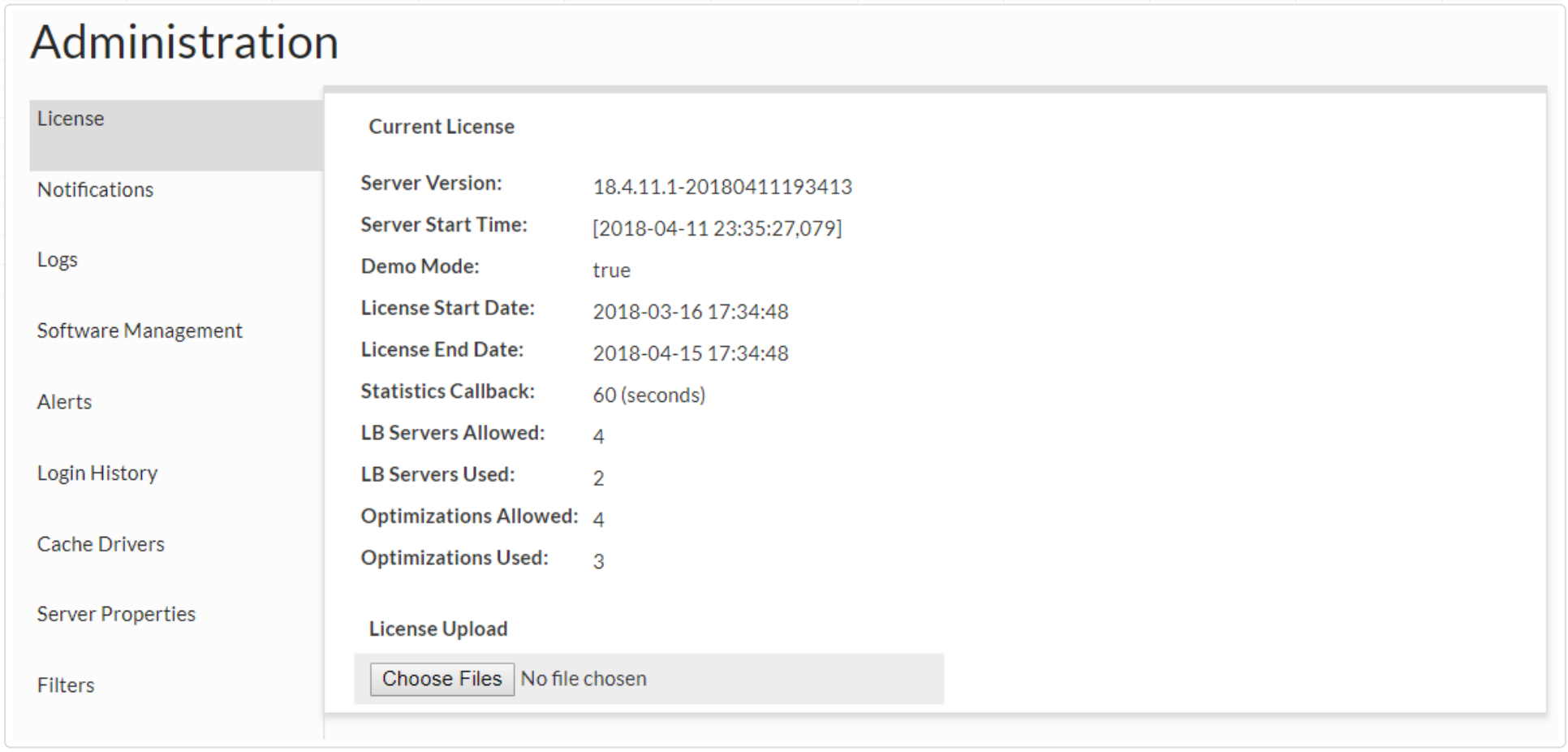
Account Information



This section is to view installation metadata to help customers work with Heimdall Support in enrolling for complimentary support. Account ID is an GUID of the installation, while Registered ID is provided from Customer Support to verify enrollment. Enrollment can be confirmed by a checkmark next to the field (as well as the removal of the UNSUPPORTED banner) see below:



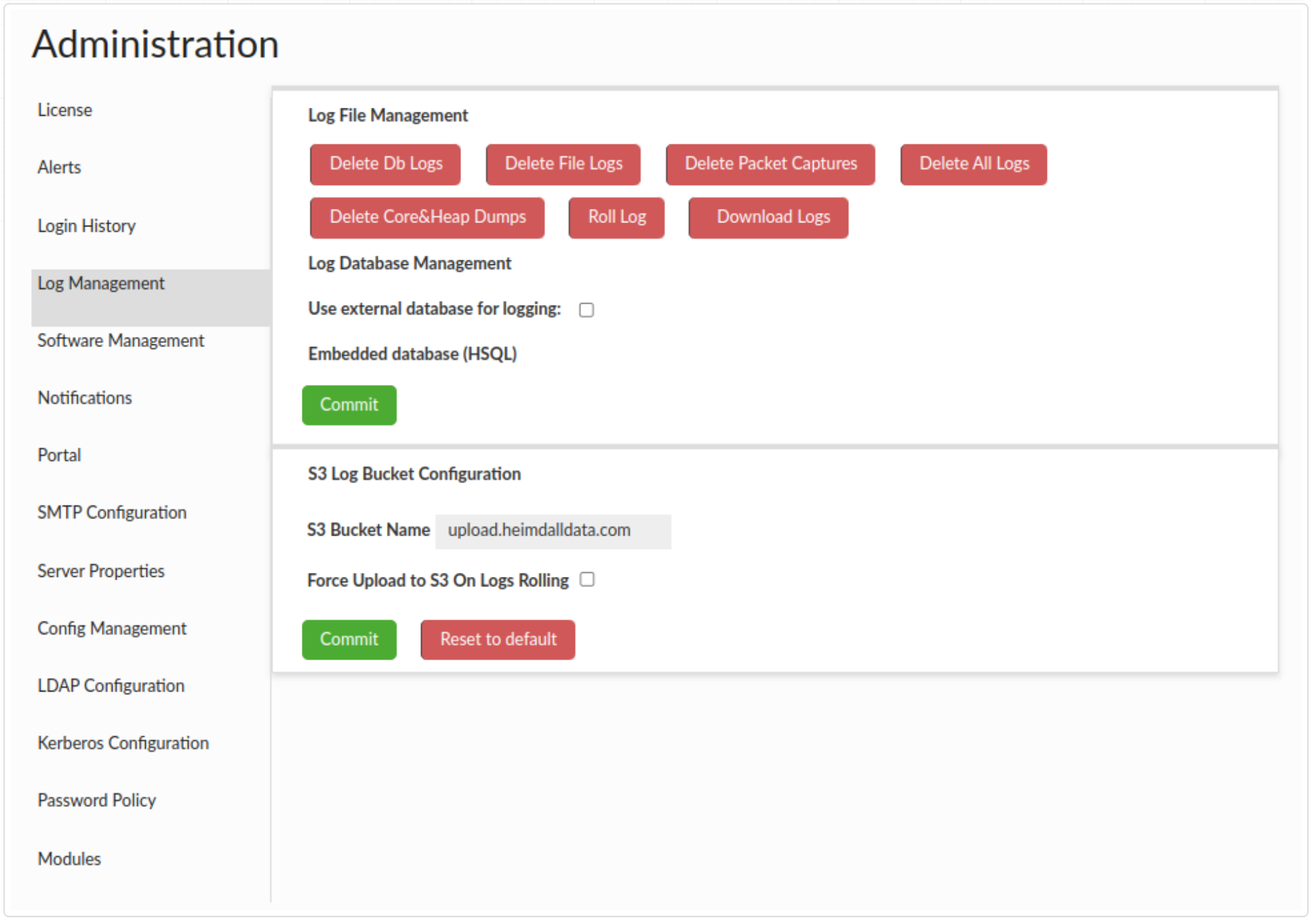
License Configuration



In this section, a new license file can be uploaded when provided by Heimdall, and the current settings and license use can be observed. In Demo mode, up to 4 LB servers can be used, and up to four VDBs with rules (optimizations) can be enabled for 30 days. The used count is only based on VDBs that have received traffic in the last 30 seconds, so will dynamically account for activity. If the used values surpass the allowed values, then a warning will be displayed on the console, but no restriction in features will be enforced. License handling is strictly advisory, i.e. while the system may complain about a license being invalid, it will not restrict what is done, to insure no outages occur as a result of licensing.

Note: When using an AWS paid-for instance, the proxy use on the instance will not count toward license use. Additionally, added paid-for instances can be reconfigured to be a pure proxy instance and will also not count to license use on another management server.

Log Management



Log Files Management

This section is provided to assist in managing log files and configuring data source used for logging. Core dumps are files that are generated by the operating system when a program terminates abnormally, such as by crashing or encountering an error. Heap dumps are files that contain a snapshot of the Java Virtual Machine (JVM) heap at a specific point in time. This functionality is removing all core files from /var/lib/systemd/coredump folder and also core files and heap dumps files from the project directory.

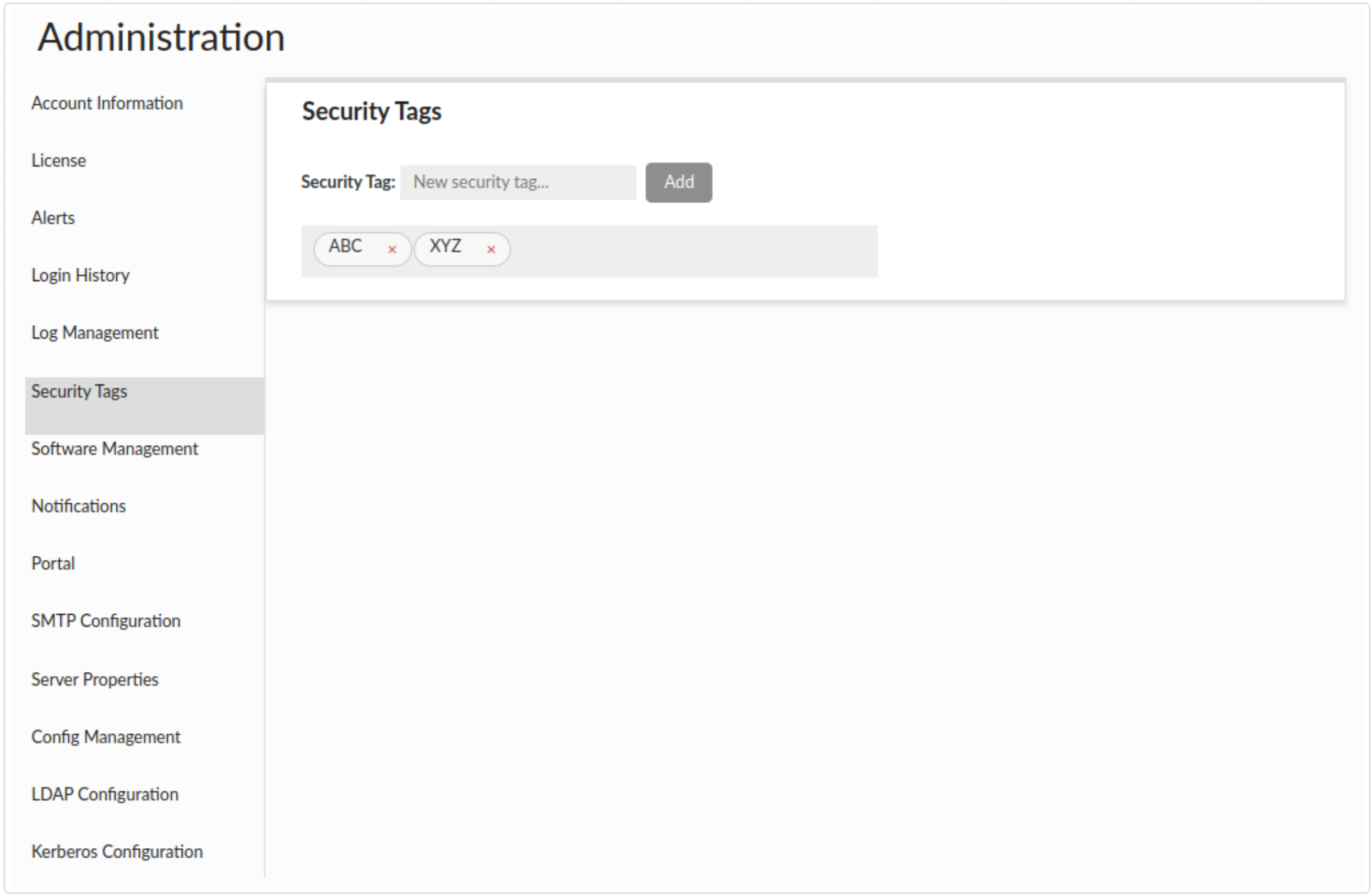
Log Database Management

By default, an embedded database supplied with Heimdall is used to store logs and statistics, however there is an option to select user configured data source (MySQL, PostgreSQL, Oracle and SQL Server supported). If Heimdall is unable to establish connection to selected source at any point it will fall back to using embedded one. User configured in selected data source has to have database root privileges.

S3 Log Bucket Configuration

In this section, we can configure log uploads to S3. There is an option to specify the S3 bucket where the logs should be saved.

Security Tags

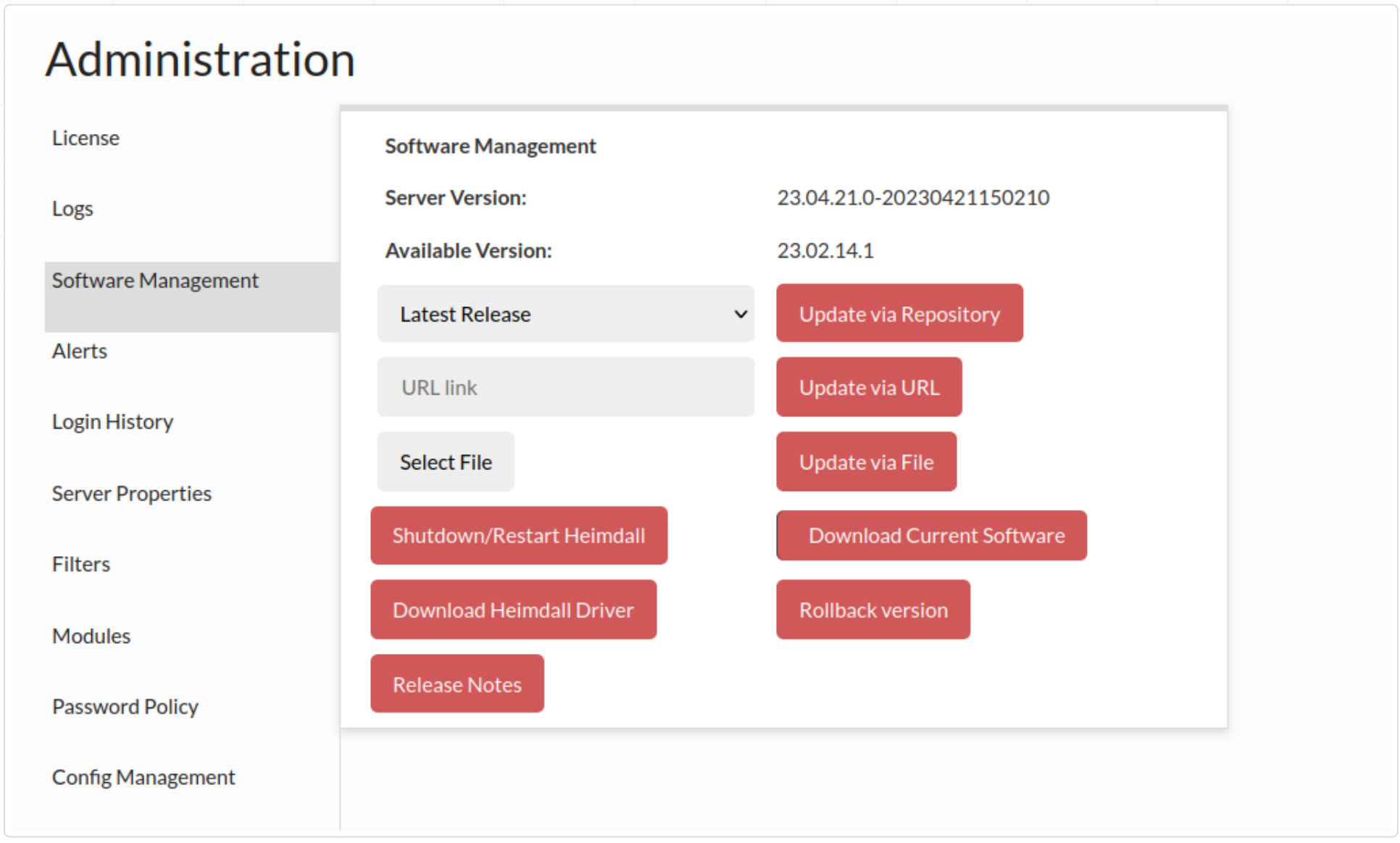


This section allows for the management of security tags, which can be applicable in the database browser section of Source tab. Each resource (i.e. databases, schemas, tables, views, columns) in the database browser can be tagged with one or more tags. The admin can select a specific tag from the dropdown and filter out the tagged resources.

Warnings:

1. If you delete a resource in your database that had assigned security tags and later decide to recreate it, it will inherit those tags.
2. If you change the name of a given resource in your database, the security tags assigned to it will be lost.

Software Management



This section allows for management of the software. In the first dropdown, you can select a version of code to update to. Release is the normal release train of code, test versions are generally just before a new release, and development are the often daily updates that have completed regression testing. Customers will often be asked to update to the development build if they have encountered a problem and want to apply the fix for it. To apply a particular version, select the version in the dropdown, then click "Update via Repository". Please note, if you have independent proxies running (vla NLB or as a service), you will need to restart them independently as well. To check the version of code each proxy is running, please refer to the status tab.

Note: Any build pushed as a development build have passed all regression testing that any previous version has passed. What may not yet be complete are regression tests for the new changes, which often are developed in parallel with customer testing of the changes. This can result in frequent development builds, but allows for rapid test/fix/deploy cycles during customer POC processes. Once a POC is completed, we will work with customers to schedule a full release build for use in production that has completed full and updated regression testing.

You can also update the version of Heimdall by providing the url link to the zip file, complete this action using Update via URL button.

In the event that you want to update to a particular version of Heimdall that was provided to you as a zip file, you can use the third section for a manual update. Simply select the heimdall zip file, then update via File.

The next option is the Shutdown/Restart Heimdall button, which will not explicitly trigger a restart, but will rely on any restart mechanisms on the server to trigger the actual execution of the server after it terminates.

In the event you need to download the Heimdall driver (say to use as a jdbc driver) the next option "Download Heimdall Driver" can be used.

If you need to roll back to a previous version of the system, the quickest way is to use the "Rollback Version" button. Heimdall stores two versions of the system - the current and the previous - and you can swiftly switch between them.

Following is the release notes link, for the most current release notes.

Note, for command line manual update, please do the following:

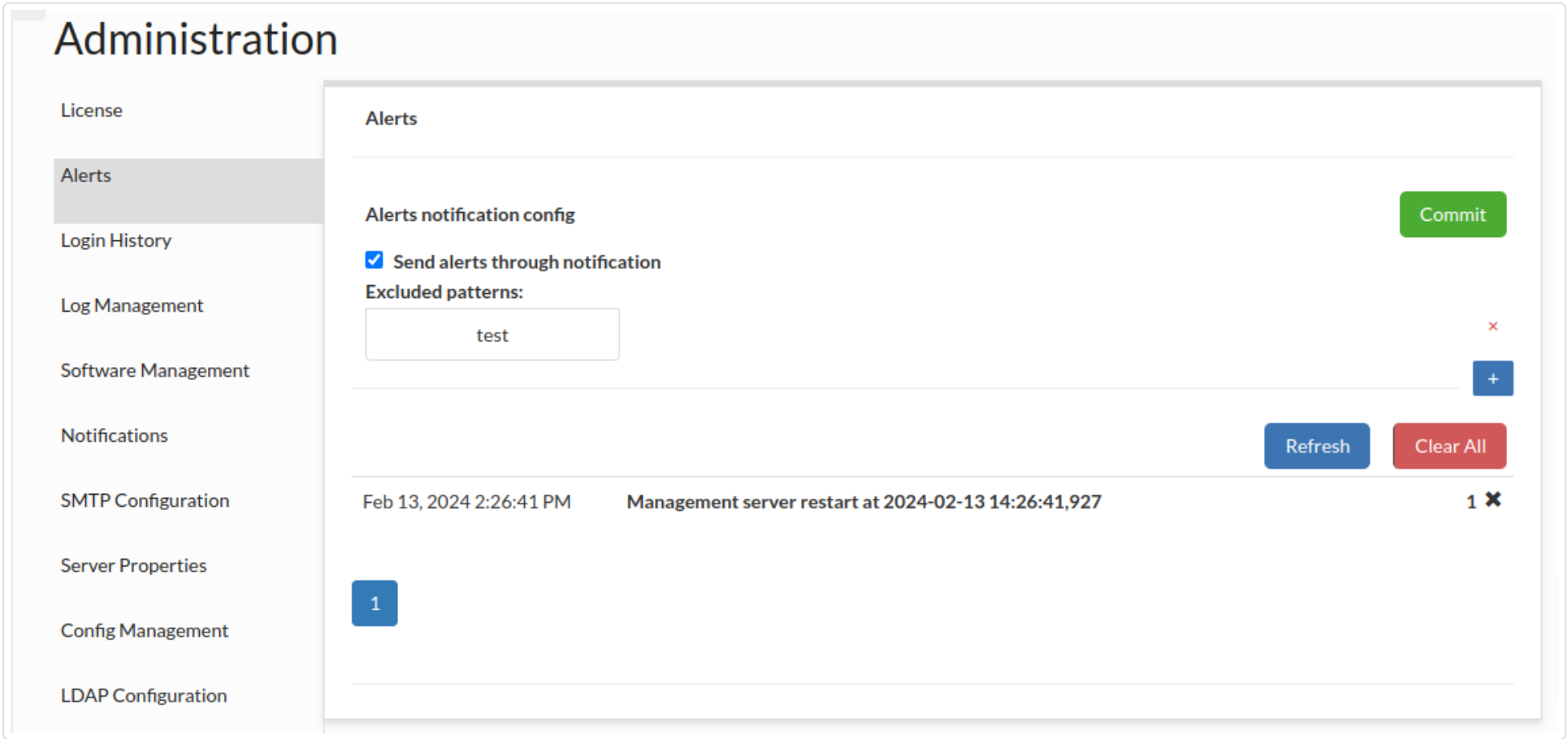
1) copy the file into the install directory as heimdall-new.zip 2) cd to install directory 3) execute the following commands:

```
* sed -i "s/\s*\"modules\/.*\",*//g" config/heimdall.conf
* unzip -j heimdall-new.zip
```

4) kill and restart the heimdallserver.jar process: ps auxw | grep heimdallserver.jar root 4198 0.0 0.0 6100 624 pts/0 S+ 18:03 0:00 grep --color=auto heimdallserver.jar root **26005** 30.8 4.0 9679444 638504 ? Sl 17:58 1:25 java -server -XX:+ExitOnOutOfMemoryError -jar /opt/heimdall/heimdallserver.jar

kill **26005**

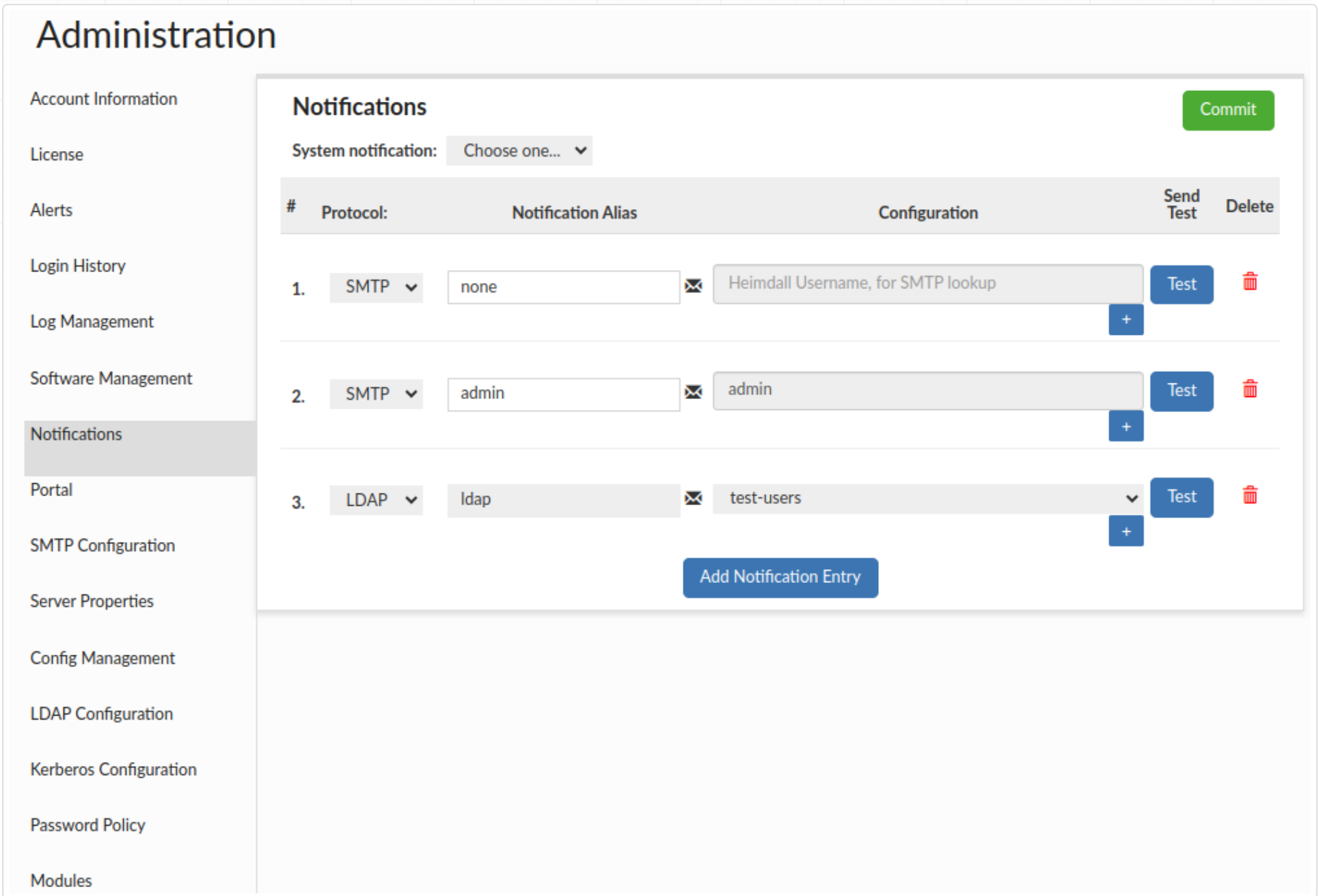
Alerts



This section provides a list of any alerts that have been dismissed with the check. Any alerts in this list will not be shown in the top of the GUI until cleared.

Sending alerts via notification: If this option is checked, every alert which message doesn't match any added pattern, will be sent via notification. If no pattern is added, all alerts will be sent via notification.

Notifications



This section provides the capability to create group notifications. Users can choose from three supported protocols: AWS SNS, LDAP, or SMTP. Additionally, there is an option to view the list of emails associated with a particular entry by clicking the envelope icon. This will display a popup showing the email addresses.

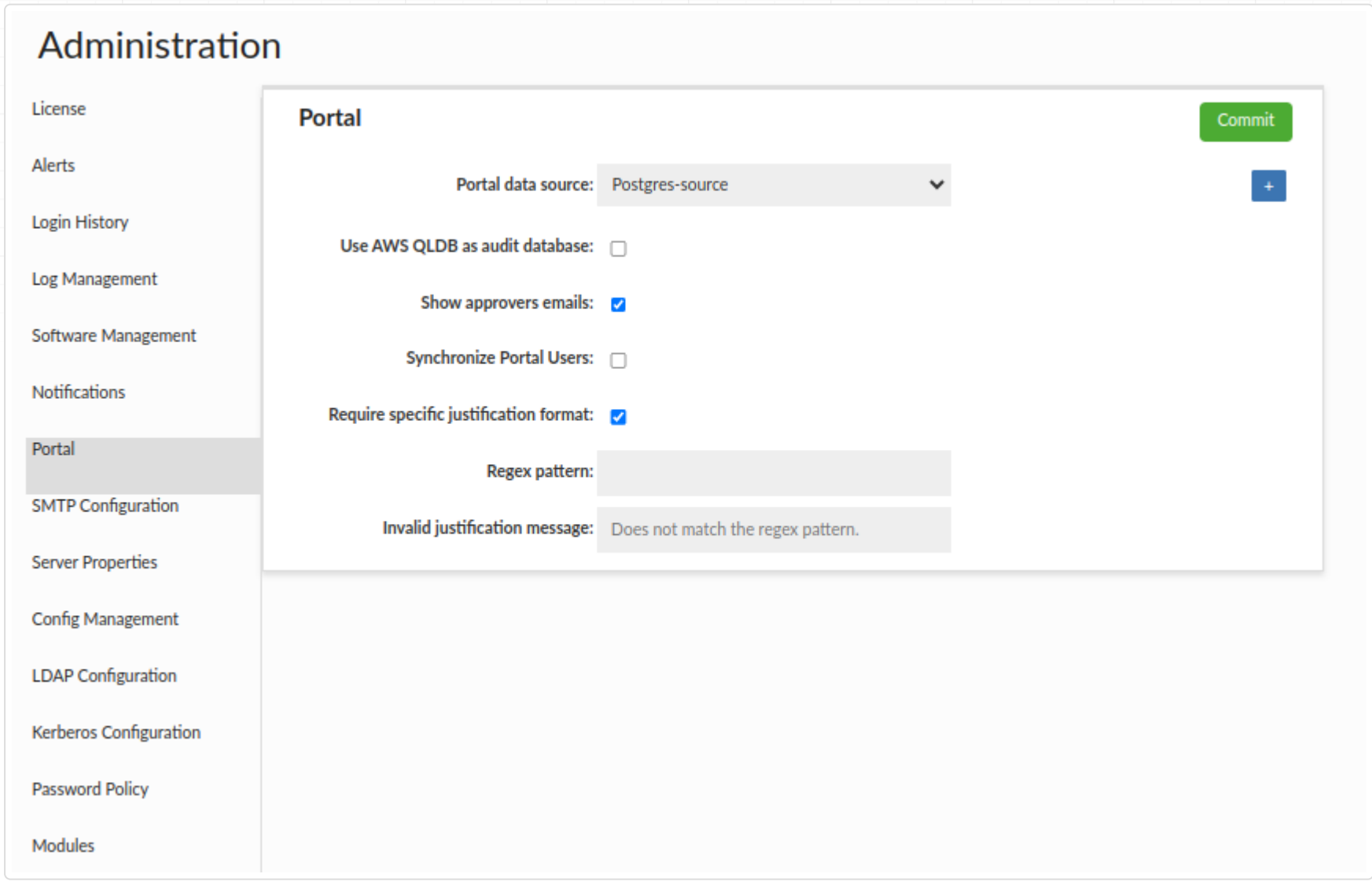
System notification: This option allows to choose which notification config is used as system notification and will be used to send various system messages, such as alerts by default.

When choosing AWS SNS for email (and potential SMS) notifications, it will require Heimdall to be hosted on the AWS platform, having appropriate AWS IAM roles and permissions configured to interact with AWS SNS.

When choosing SMTP, user will be able to define his mailing list (from existing Heimdall defined users in Users tab). Also, please remember to configure SMTP client in the Admin tab -> Smtp Configuration.

After making any changes, they must be submitted using the Commit button.

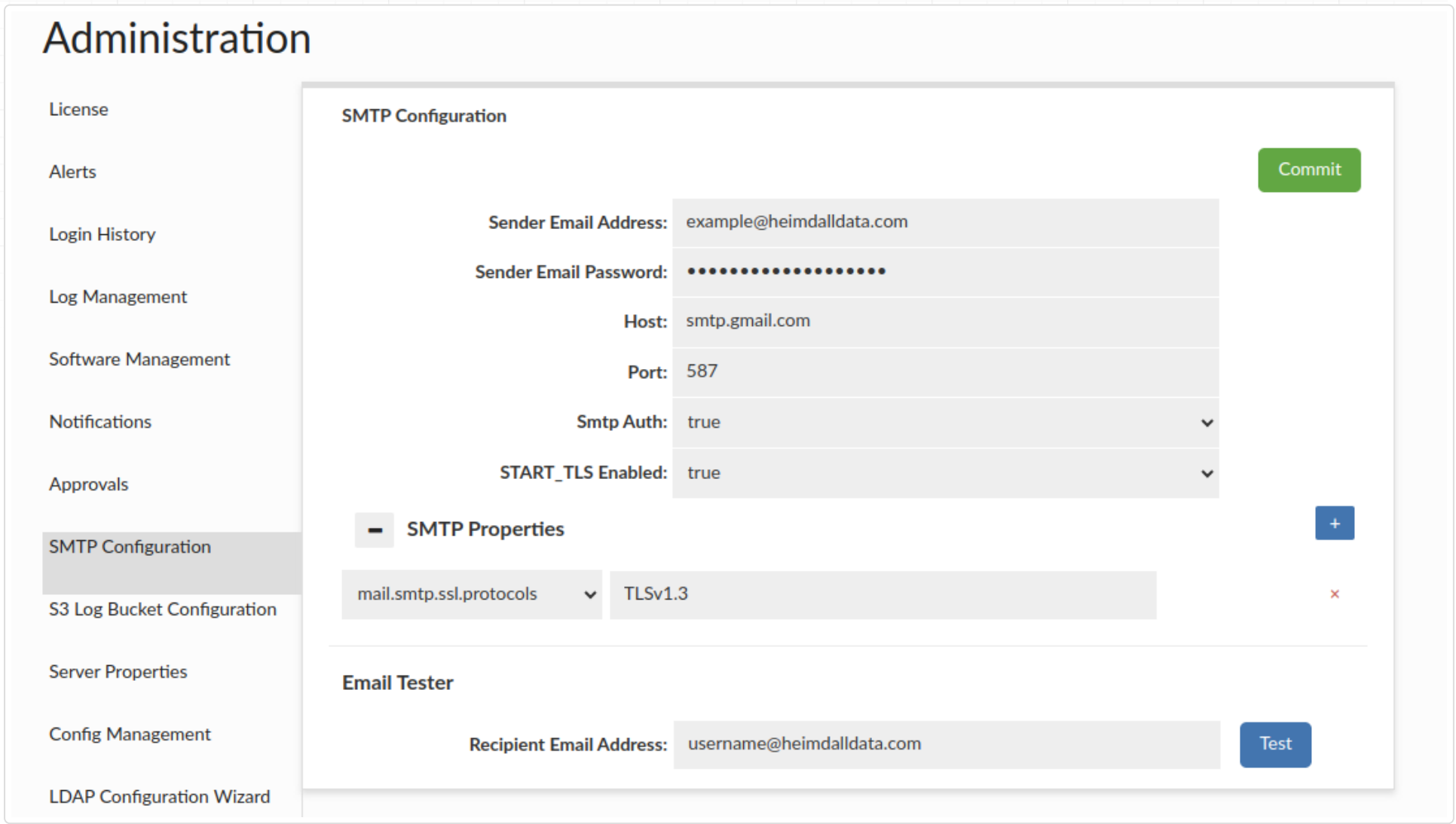
Portal



This section is generally used to configure settings related to the portal. Within it, we can define the LDAP configuration, from which portal users will be sourced, the data source where portal data will be stored, and several checkboxes.

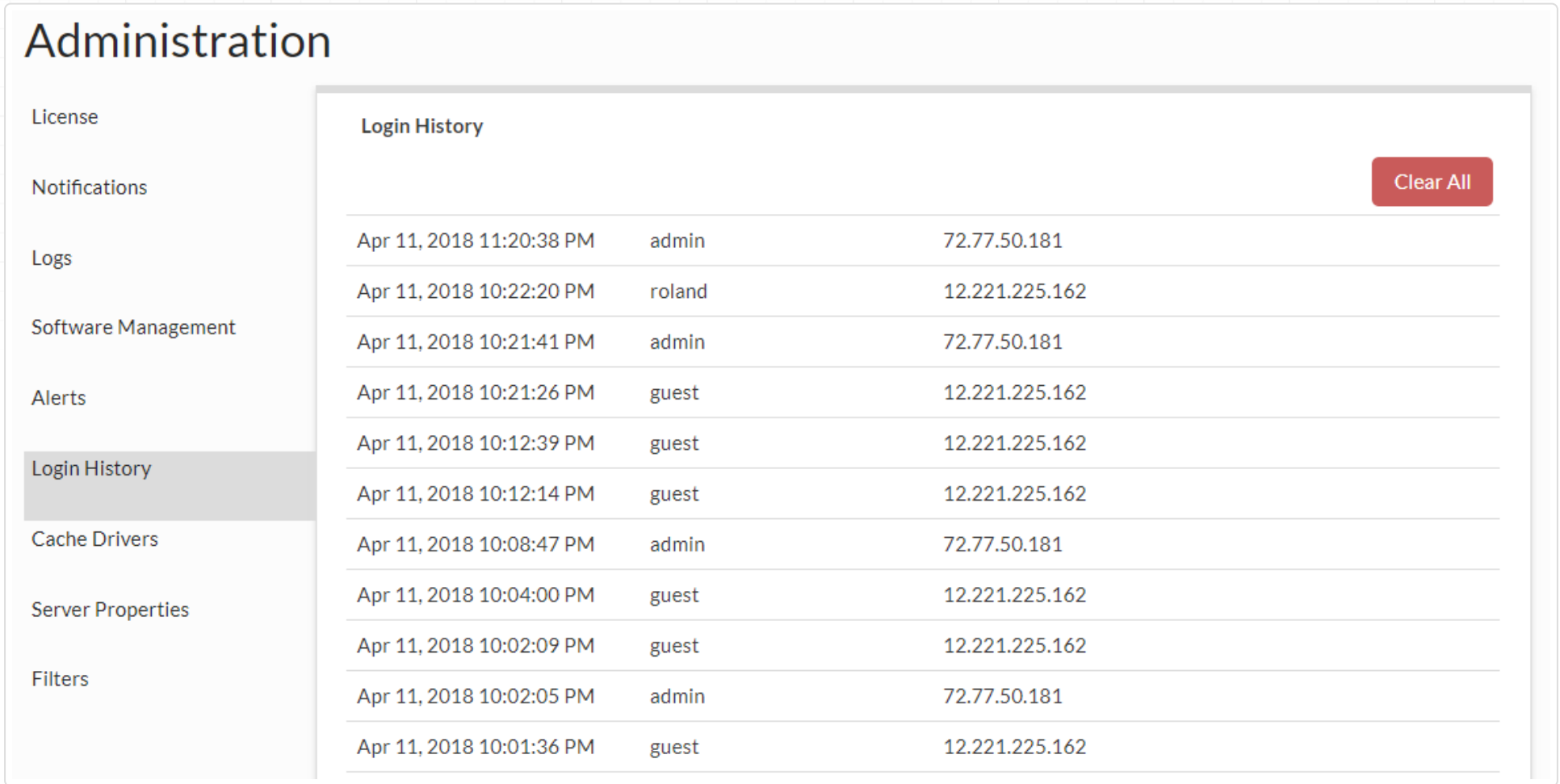
- **Use AWS QLDB as audit database** - Indicates whether the audit trail data should also be saved in the AWS cloud.
- **Show approvers emails** - If selected, it will be possible for all users to view approvers' emails for a specific role. Admins can always see these emails, even without the checkbox being checked.
- **Synchronize Portal Users** - Indicates that if the LDAP Group Filter matches, Heimdall (GUI) users will be created automatically.
- **Require specific justification format** - If selected, there is an option to specify the regex that the justification must match when requesting a session.
    - **Regex pattern** - Specifies the regex that must be matched in the justification field.
    - **Invalid justification message** - Specifies the message to appear when requesting a session if the justification content does not match the configured regex.

SMTP Configuration



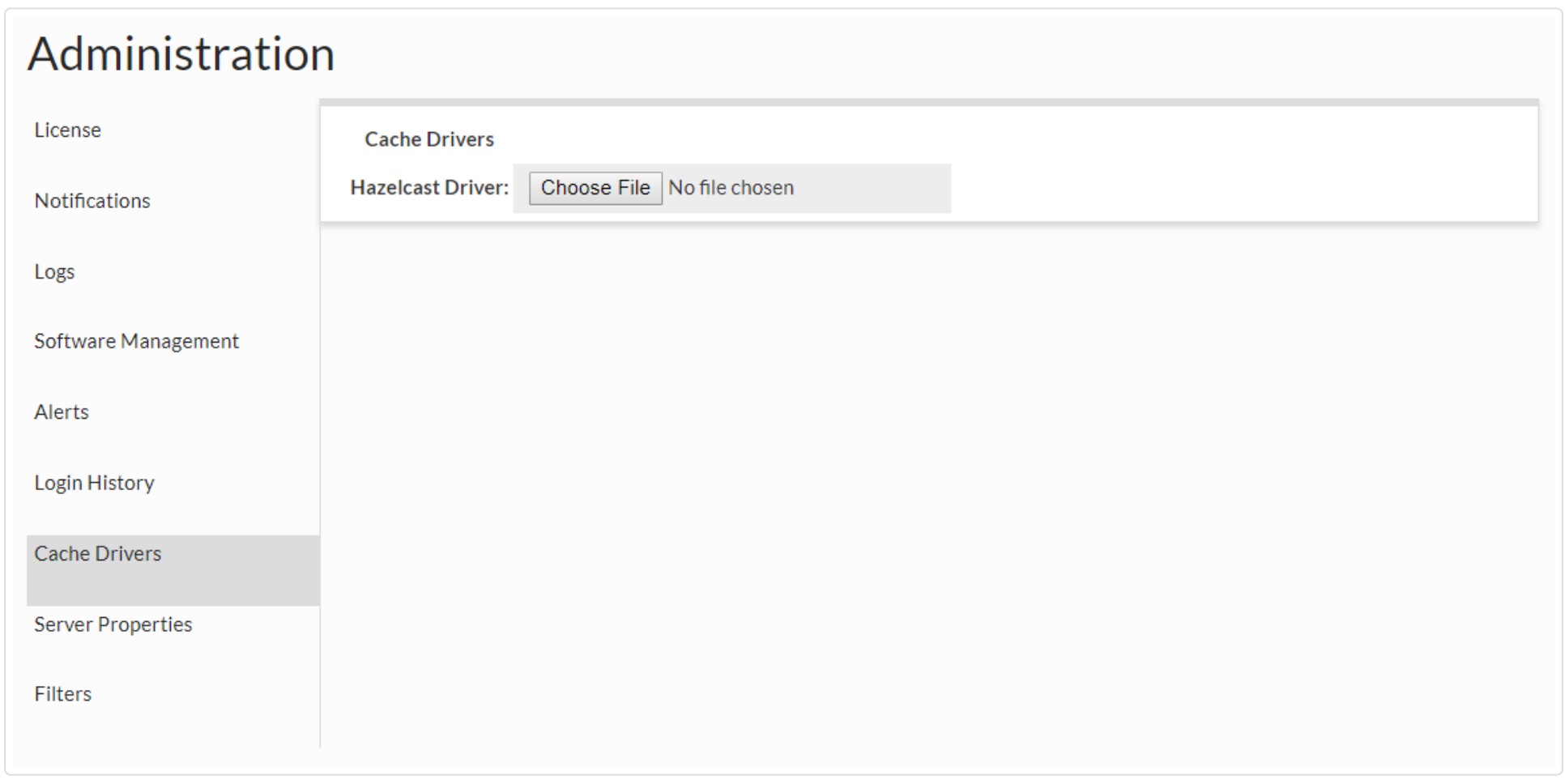
This section provides the functionality to establish SMTP configurations for email dispatch. Users can adjust the configuration to their preferences, by incorporating relevant properties in the SMTP Properties section (available properties can be found here: https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summary.html). The Heimdall team recommends users set the ssl protocol to TLSv1.2 or TLSv1.3, otherwise TLSv1 might be used. The Email Tester component, allows users to validate configuration before committing changes by sending a test email. If changes will are not committed, all entered configuration will be lost. While using e.g. Google smtp, user should consider using application password rather than their personal password - especially if the account has 2fa enabled, then without app password functionality might have problem to operate.

Login History



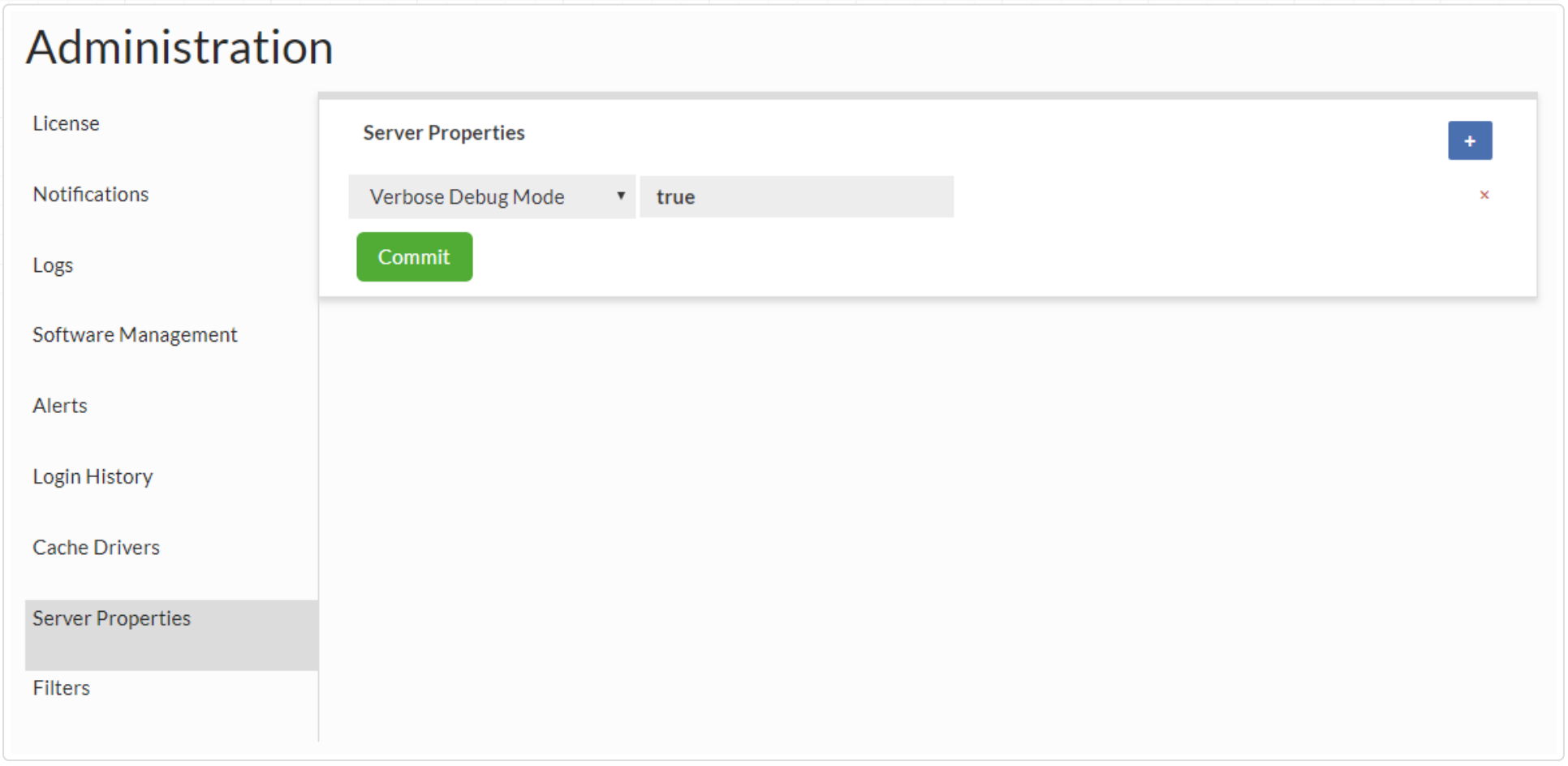
This provides the login times, account ID and source IP of the recentally recorded GUI logins.

Cache Drivers



This section allows a cache driver (specifically Hazelcast) to be uploaded for use by Heimdall instead of the built-in driver. This allows the driver to match what may exist in a customer's environment if our version causes a conflict.

Server Properties



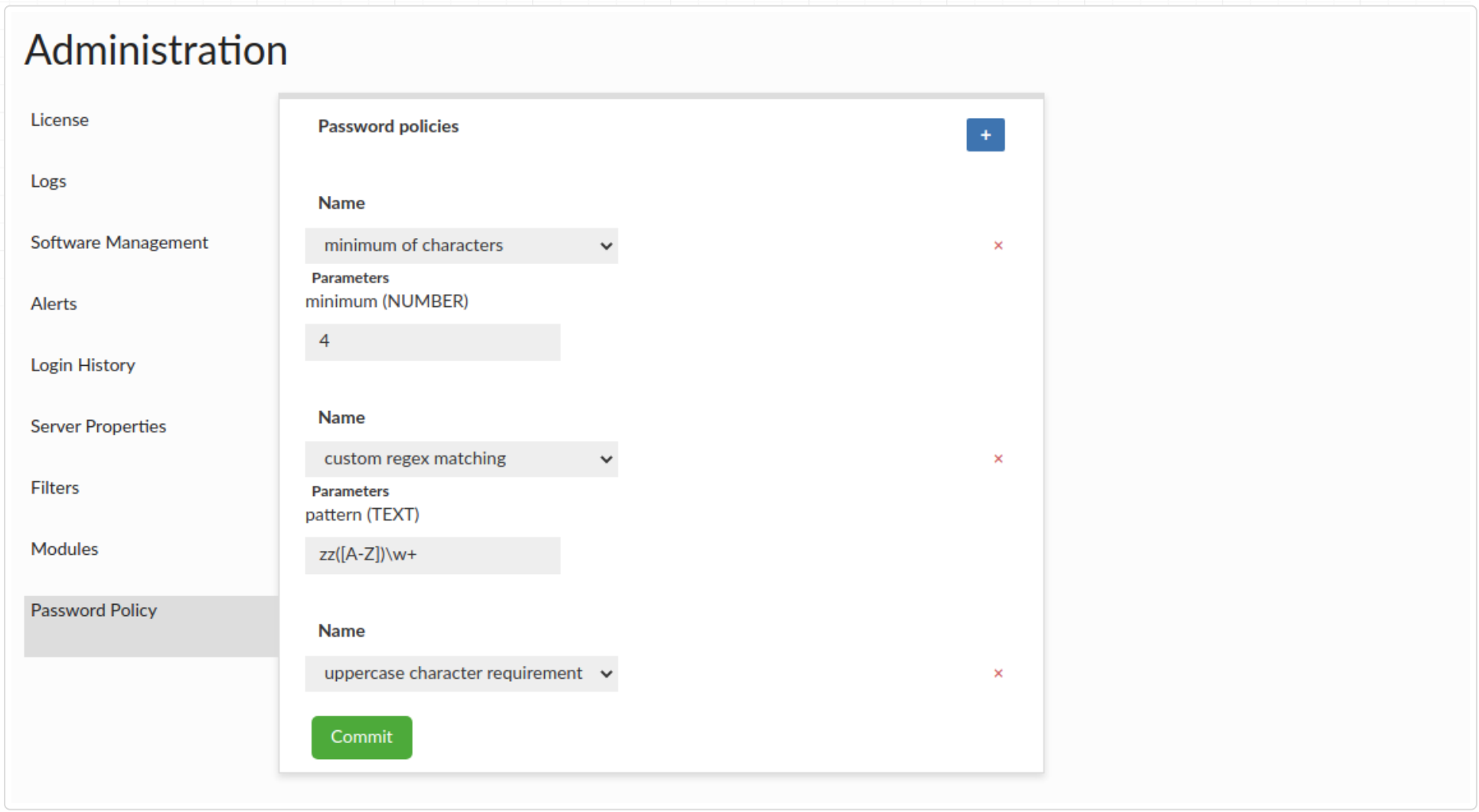
These are properties that can control overall Management server behavior, and will be added to over time. Current properties supported.

- **Disable Cert Validation:** Disable TLS certificate validation for software downloads (Boolean)
- **Enable Portal Mode:** Value that indicates if after successful login central manager or portal mode will be used (Boolean)
- **Minimum free disk space %:** configured how much space needs to be free for any configuration saves to occur. Defaults to 1 for 1%. This helps prevent configuration loss if not enough space is available to successfully save the full configuration. (Double)
- **Log Events To Console:** For debugging or container use, log ALL events to the stdout console of management server (Boolean)
- **Max Log Age:** Sets the maximum age in days of log files and log records (Integer)
- **Enables password validation:** Validate new users password along with provided rules (Boolean)
- **Proxy Host:** If a proxy is needed to access the Internet for code update notifications and downloads, the host to use for this (String)
- **Proxy Password:** Proxy password for proxy authentication, no default (String)
- **Proxy Port:** The port defaults is 3128 (Integer)
- **Proxy User:** Proxy user for proxy authentication, no default (String)
- **Redirect Config Fetches:** Value that indicates if server should redirect all HTTP config requests to HTTPS Tomcat port (Boolean)
- **Reserved Disk Space:** Sets the amount of FREE memory to maintain on the log filesystem (Double)
- **Verbose Debug Mode:** On the server, enable verbose debug mode (Boolean)

Modules

Allows modules to be added or removed as needed. The fewer the modules, the faster a Heimdall node can startup. In particular cache modules that are unused should be deleted as some can add significant load time.

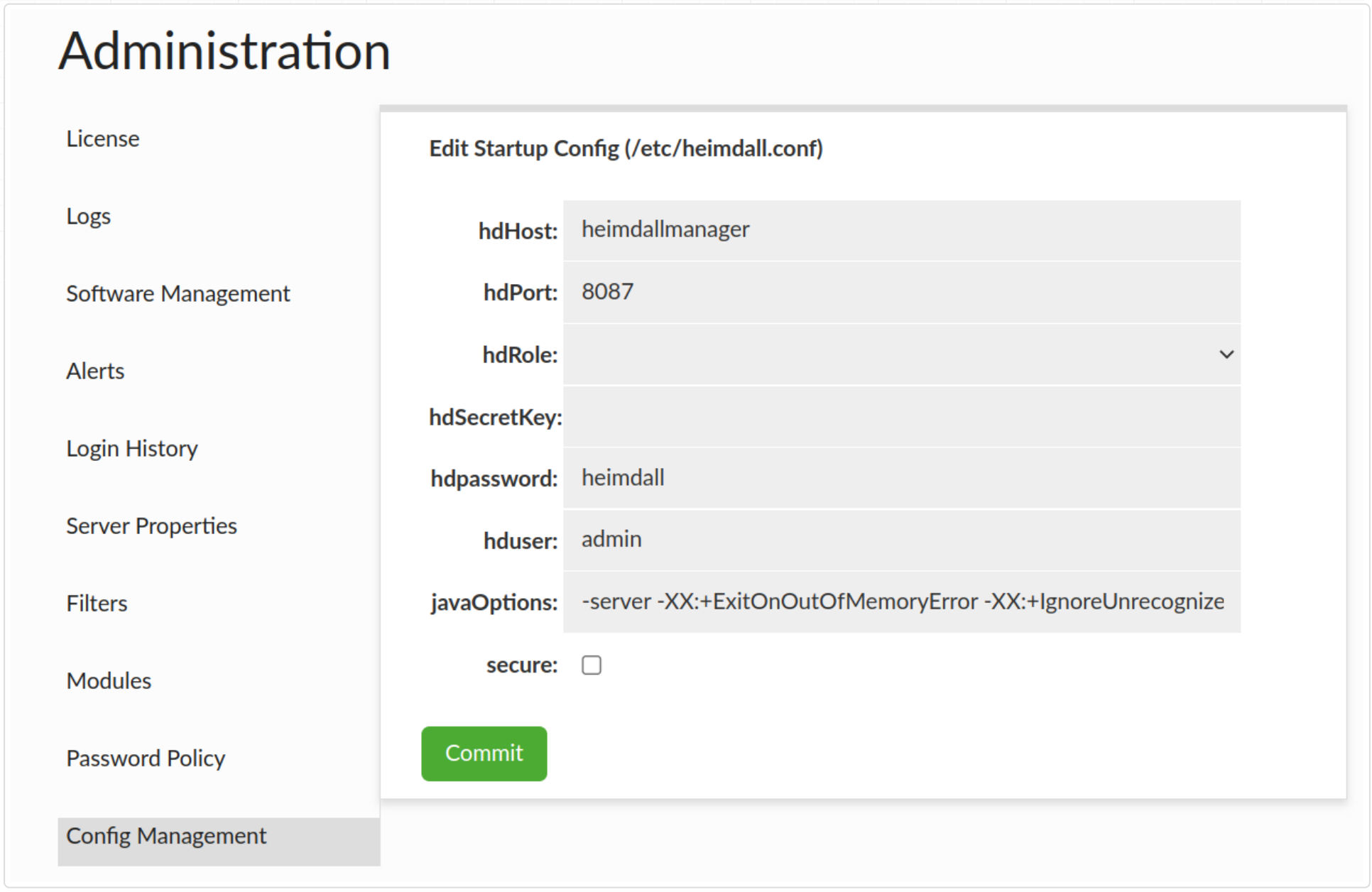
Password Policies



This section allows user to set up requirements that specific user password must meet to be accepted as a new one. Current rules supported:

- minimum/maximum of characters - minimum/maximum number of characters that password must contain (with **minimum**/**maximum** parameter tht defines the limit).
- digits requirement - rule defining that password must contain at least one digit
- special marks requirement - rule defining that password must contain at least one special mark
- uppercase character requirement - rule defining that password must contain at least one uppercase character
- custom regex matching - rule that provides a **pattern** parameter with the regular expression to which password must match

Config Management



This section allows editing startup `/etc/heimdall.conf` configuration file, which can be used to populate environmental variables or cloud environments. The user can set options described below.

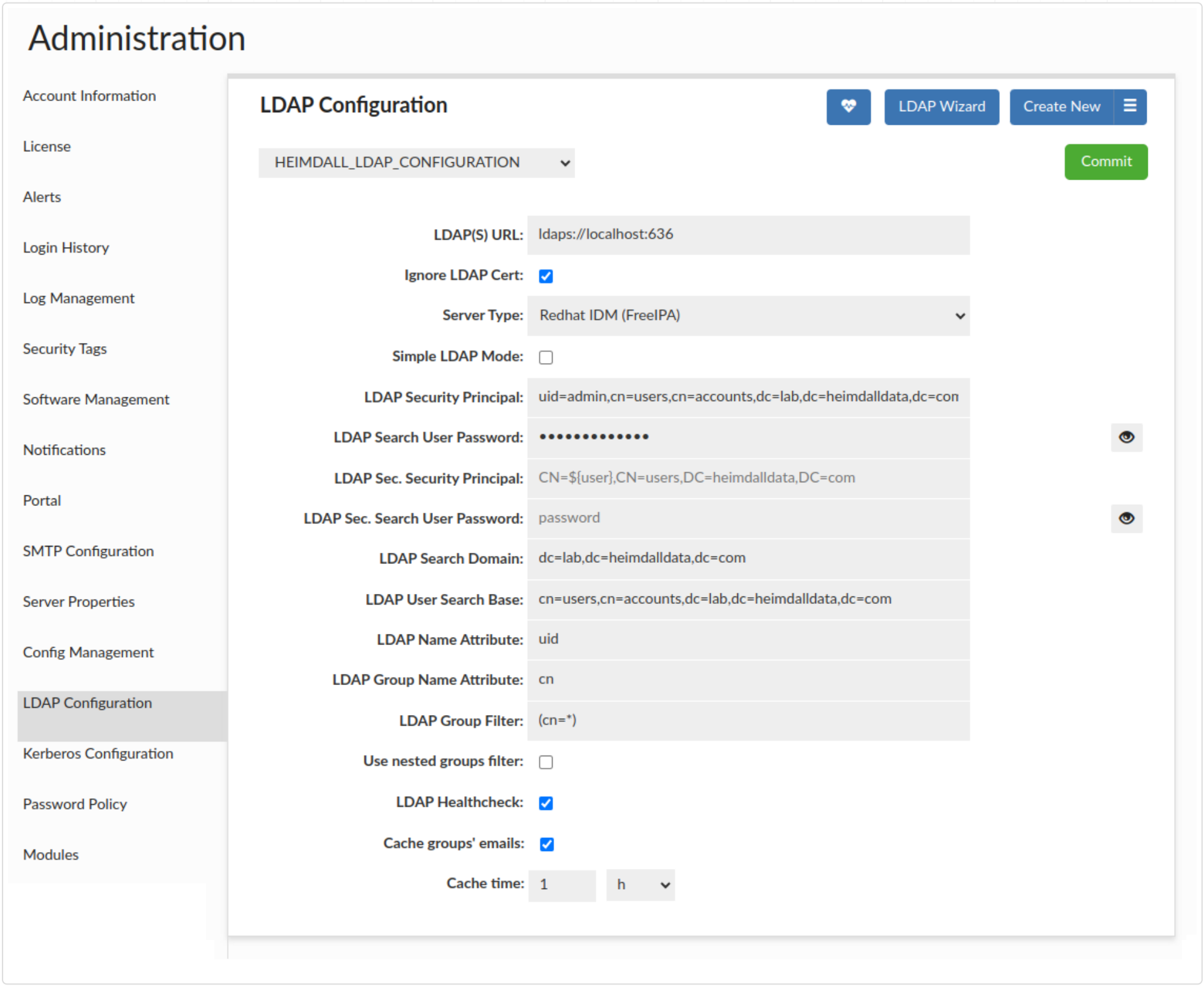
Option Name	Description	Default value
`hdRole`	proxy	
`hdHost`	Hostname of management server	heimdallmanager
`hdPort`	Port of the management server, generally 8087 or 8443	8087
`secure`	If the proxy should use HTTPS to connect to the manager	false
`hdUser`	Login username for the management server, can be admin	admin
`hdPassword`	Login password for the management server	heimdall
`javaOptions`	Any arbitrary options desired to be set	-server -XX:+ExitOnOutOfMemoryError -XX:+IgnoreUnrecognizedVMOptions -XX:-MaxFDLimit -XX:+UnlockExperimentalVMOptions
`hdSecretKey`	In AWS, use this as the name of an "AWS Secret" to store the configuration, protecting included passwords from being written to disk. To use, proper permissions must be set on the IAM role. This option provides two major benefits. First is that all passwords are stored in AWS Secrets, in an encrypted format. Second is that redeployment of a management server can be done with a configuration pre-populated, so there is no need to backup and restore configurations to account for failures. Simply terminate the old instance and a new instance with the same user-data will be created with the same configuration as the original.	

LDAP Configuration (Proxy & Central Manager)

LDAP Authentication provides authentication with using of LDAP server. LDAP Authentication provides two modes of authentication:

- Bind + Search Mode - authentication is made by binding to server as admin and searching information about authenticated user;
- Simple Mode - authentication is made by binding to server as authenticated user.

Bind + Search Mode



To configure Bind + Search Mode, below options can be set.

LDAP option	Required?	Description	Example value
LDAP(S) URL	yes	Specifies URL of LDAP server. Instead of just one URL, you can also supply a space-separated list of URLs. In this case, the LDAP provider will attempt to use each URL in turn until it is able to create a successful connection.	ldap://server.example.com:389
Server Type	yes	Specifies the type of LDAP Server	RedHat IDM (FreeIPA)
LDAP Security Principal	yes	Specifies name of user used to search for authenticated user	CN=ro-admin,CN=Users,DC=example,DC=com

LDAP option	Required?	Description	Example value
LDAP Sec. Security Principal	no	Specifies name of user used to search for authenticated user in case the primary search with LDAP Security Principal fails.	CN=admin2,CN=Users,DC=example,DC=com
LDAP Search User Password	yes	Specifies password for LDAP Security Principal user used to search for authenticated user	examplepassword123
LDAP Sec. Search User Password	no	Specifies password for LDAP Sec. Security Principal.	examplepassword321
LDAP Search Domain	yes	Specifies LDAP search domain, the domain in which authenticated user's groups will be searched.	DC=example,DC=com
LDAP User Search Base	yes	Specified LDAP user search base in which authenticated user will be searched.	CN=Users,DC=example,DC=com
LDAP Name Attribute	yes	Specifies name attribute by which authenticated user	sAMAccountName
LDAP Group Name Attribute	no	Optional, specifies group's name attribute which should be read during extracting authenticated user's groups. If not provided, LDAP Name Attribute will be used instead.	CN
LDAP Group Filter	no	Optional, option used during searching for authenticated user. Setting this option limits the number of groups to search user into them, only to particular group inside server. Setting this option makes it required to extract at least one group to authenticate the user.	(DistinguishedName=CN=group1,CN=Users,DC=exan
Use nested groups filter	-	Specifies if parent groups should be included, when searching for user's groups.	-
Ignore LDAP Cert	-	Specifies if TLS validation of LDAP server certificate should be performed	-
LDAP Healthcheck	no	Performs LDAP Healthcheck automatically every minute. An appropriate alert will be shown if the server goes down (Similarly, once the server is back online).	-
Cache groups' emails	no	Specifies whether emails for LDAP groups should be cached.	-
Cache time	no	Specifies the interval at which emails in the cache should be updated.	-

By choosing option Simple LDAP Mode, you can switch to Simple Mode.

Advanced group filter

LDAP group filter is used to limit number of groups for look up for authenticated user. Example value describes filter limiting number of groups to only one particular group, but this option value is added as written, what enables writing more complex filters. Please look on below example.

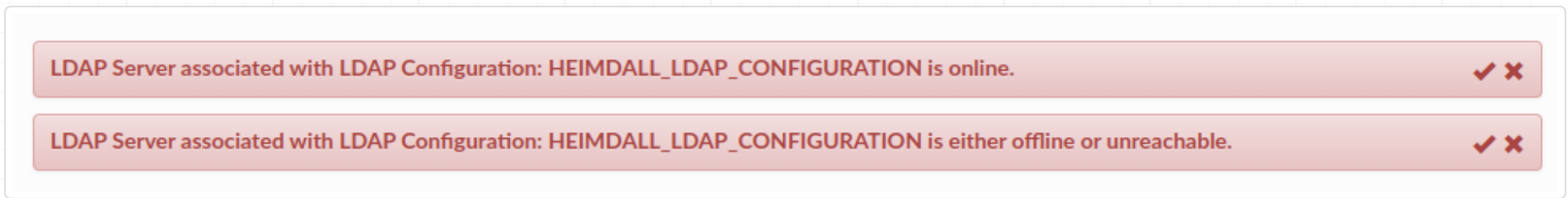
Let's assume that we want our user be from groups `group1` or `group2`. By knowing the syntax of search filter used in LDAP server we can set `LDAP Group Filter` as:

```
(|(DistinguishedName=CN=group1,CN=Users,DC=example,DC=com)(DistinguishedName=CN=group2,CN=Users,DC=example,DC=com))
```

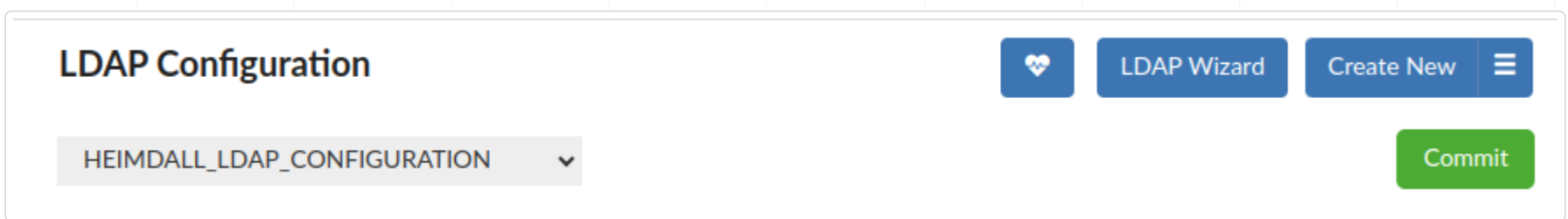
Important: If there is an LDAP Group Filter specified at least one group has to be extracted to authenticate the user.

LDAP Healthcheck

If enabled, performs in the background automatically every minute and may result in two alerts: once the server is down, and once the server is back online.

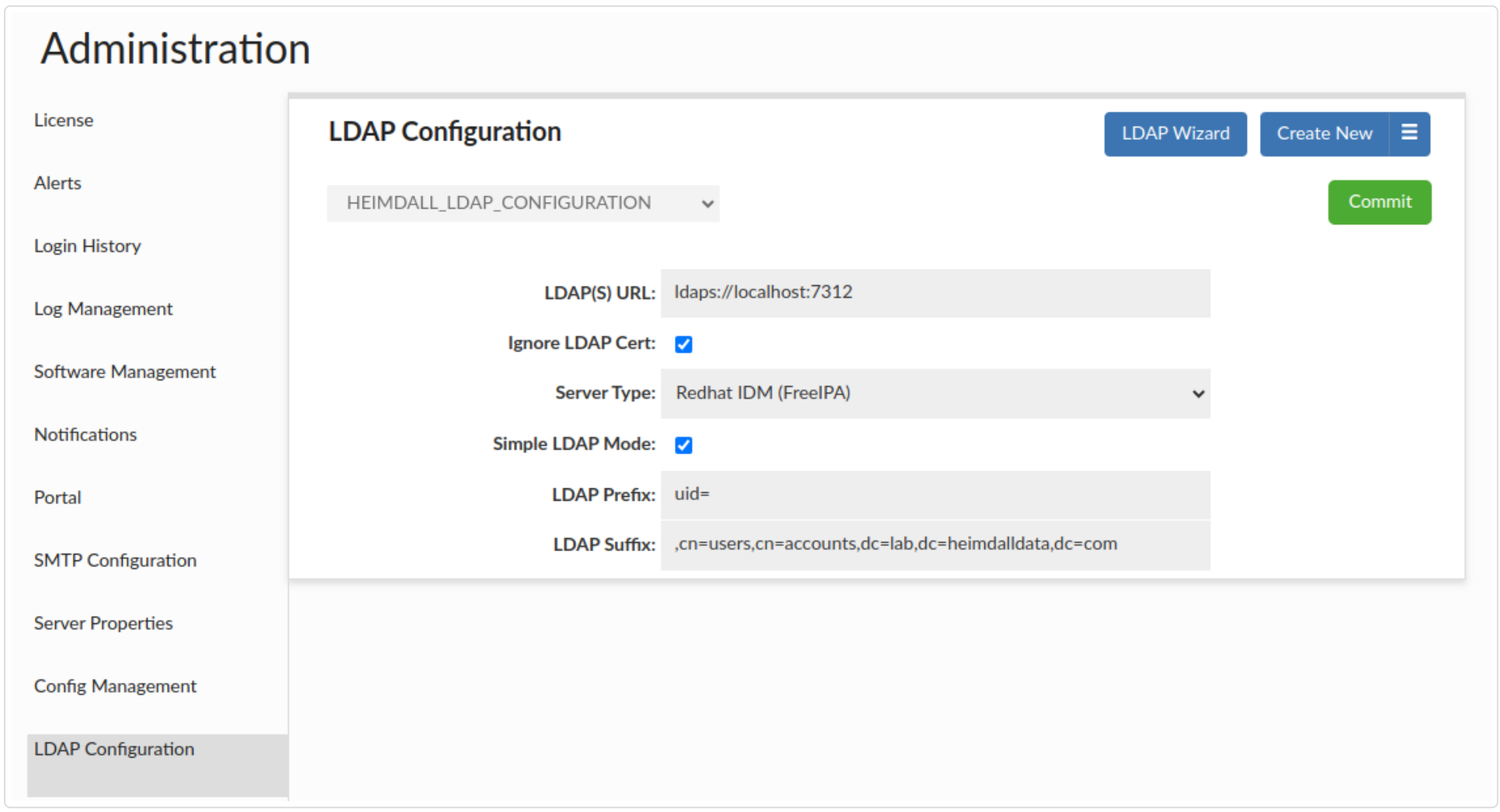


Nevertheless it can be performed on demand by using the button with hearth - if used, the JSON with the details of LDAP Healthcheck result will be opened in the new tab.



Important: LDAP Healthcheck on demand will be performed on **ALL** configurations with LDAP Healthcheck checkbox enabled.

Simple Mode



Simple Mode is simpler mode than Bind + Search mode, because uses only one request (which is binding request) to LDAP server.
Simple Mode can be turned on by choosing Active Directory (LDAP) Auth Enabled checkbox, and then choosing Simple LDAP Mode checkbox. Simple Mode requires to set below options to work properly.

LDAP option	Required?	Description	Example value
LDAP(S) URL	yes	Specifies URL of LDAP server. Instead of just one URL, you can also supply a space-separated list of URLs. In this case, the LDAP provider will attempt to use each URL in turn until it is able to create a successful connection.	ldap://server.example.com:389
LDAP Prefix	yes	Specifies prefix used in during bind authenticated user to LDAP server	CN=

LDAP option	Required?	Description	Example value
LDAP Suffix	yes	Specifies suffix used in during bind authenticated user to LDAP server	,CN=Users, DC=example, DC=com

To understand working of this mode, should be known that send request is binding composed of three parts: `prefix + username + suffix`. For given example values and user `exampleuser`, binding request would look like `CN=exampleuser,CN=Users, DC=example, DC=com`.

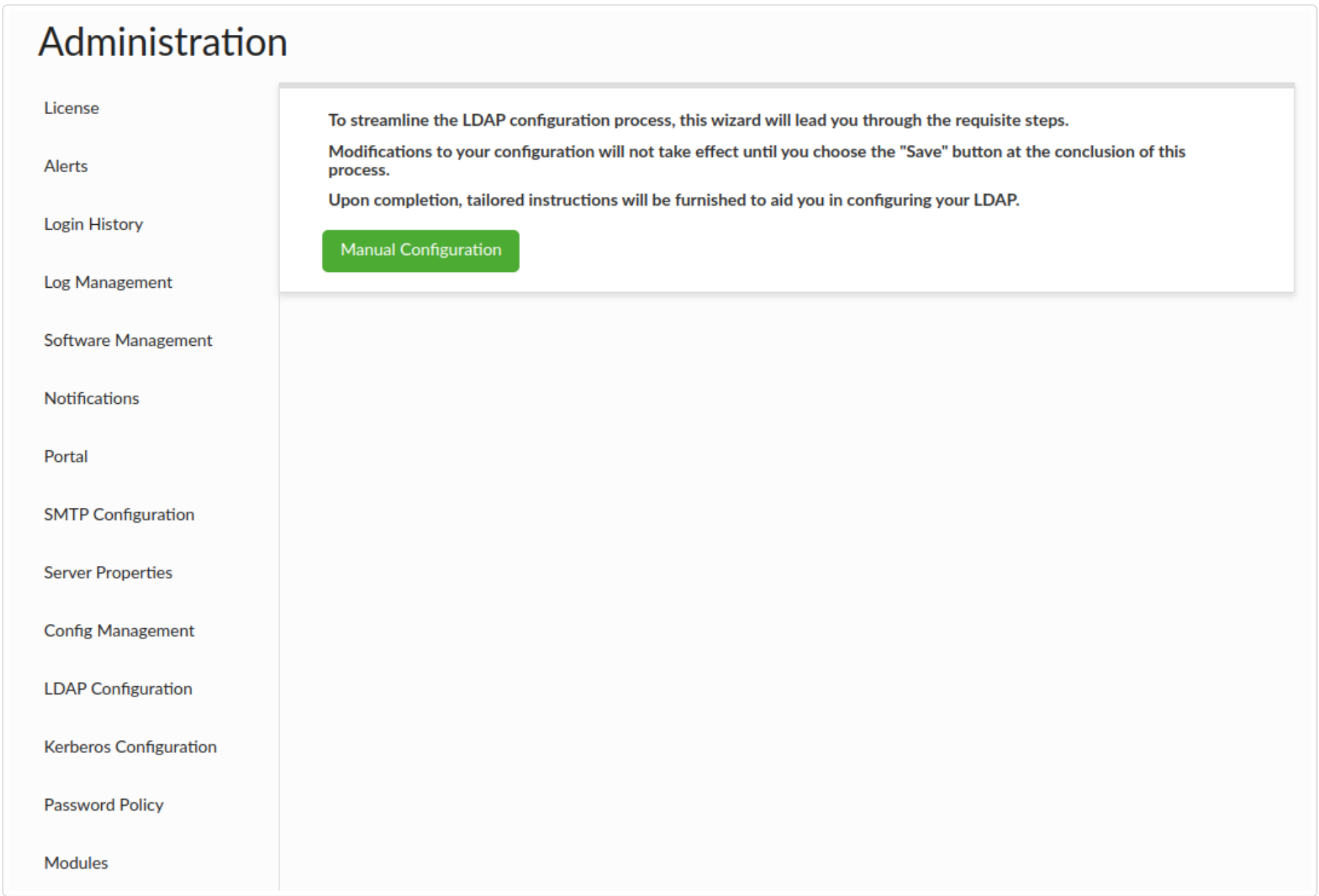
Central Manager Information

If you want to authenticate CM user, in the Users tab you should create new user(-s) with the same name as in your active directory. During creating new user don't forget to mark the "Authenticate By LDAP" option.

LDAP Configuration Wizard

The LDAP Configuration Wizard is provided to simplify configuration process as much as possible. It's a walk-through tool preventing from going forward if provided configuration is not valid. It can be accessed by clicking the "LDAP Wizard" button in the LDAP Configuration section.

Bind + Search Mode



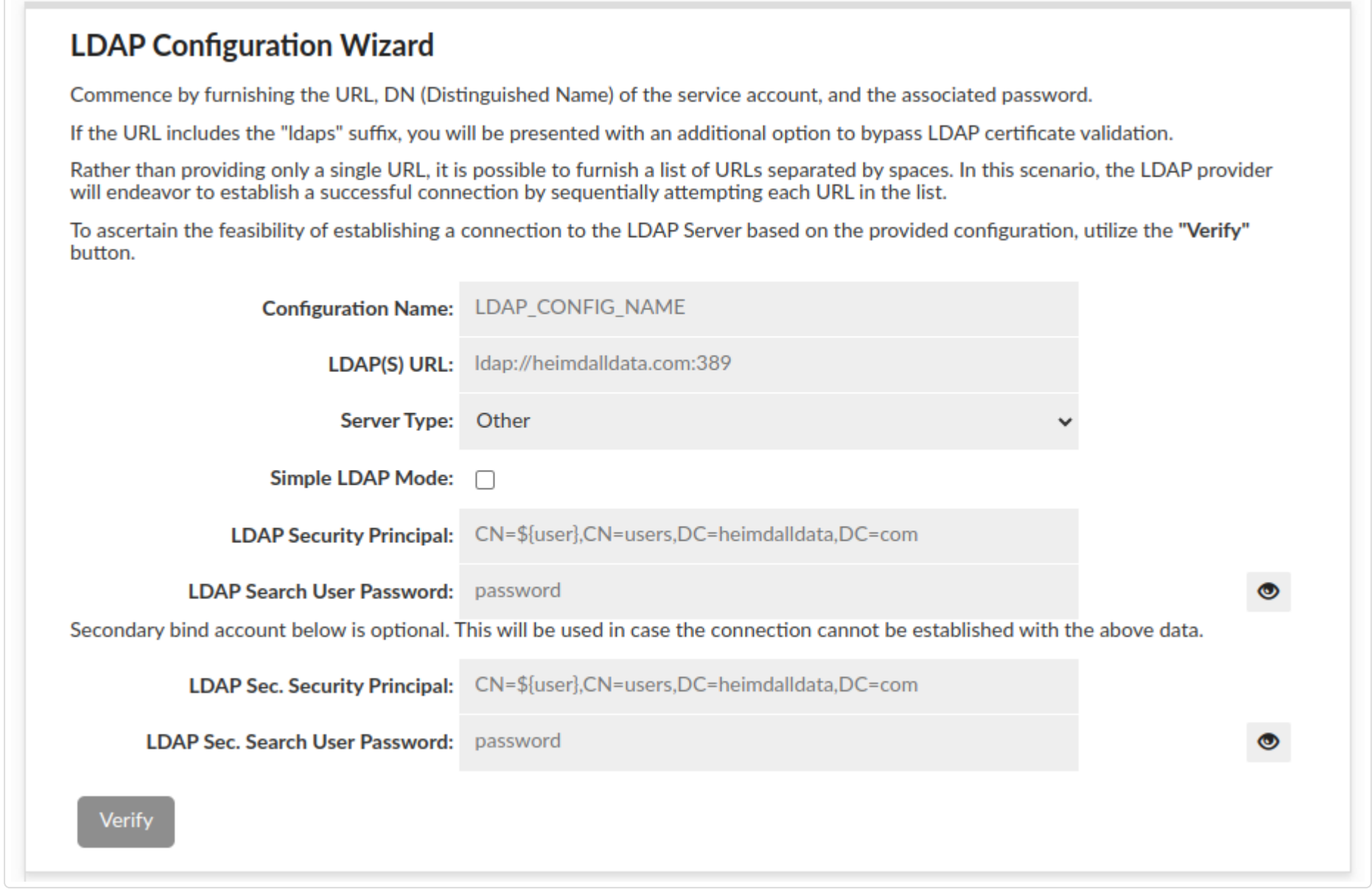
There is the list of information necessary to finalize the Wizard Configuration:

- **LDAP(S) URL** - Specifies URL of LDAP server
- **Ignore LDAP Cert** - Specifies if TLS validation of LDAP server certificate should be performed
- **LDAP Security Principal** - Specifies name of user used to search for authenticated user
- **LDAP Sec. Security Principal** - Specifies name of user used to search for authenticated user if the primary search fails
- **LDAP Search User Password** - Specifies password for LDAP Security Principal user used to search for authenticated user

- **LDAP Sec. Search User Password** - Specifies password for LDAP Sec. Security Principal user used to search for authenticated user
- **LDAP Search Domain** - Specifies the domain in which authenticated user's groups will be searched
- **LDAP User Search Base** - Specifies user search base in which authenticated user will be searched
- **LDAP Name Attribute** - Specifies attribute which should be used to identify a user
- **LDAP Group Name Attribute** - Specifies attribute, which should be used to extract information about a group (Optional)
- **LDAP Group Filter** - Specifies group filter, which will be added during a search for a user's groups (Optional)
- **Use nested groups filter** - Specifies if parent groups should be included, when searching for user's groups
- **LDAP Healthcheck** - Performs LDAP Healthcheck automatically every minute. An appropriate alert will be shown if the server goes down (Similarly, once the server is back online).
- **Cache groups' emails** - Specifies whether emails for LDAP groups should be cached (applies only if LDAP configuration is linked to the portal configuration).
- **Cache time** - Specifies the interval at which emails in the cache should be updated.

Service Account Validation

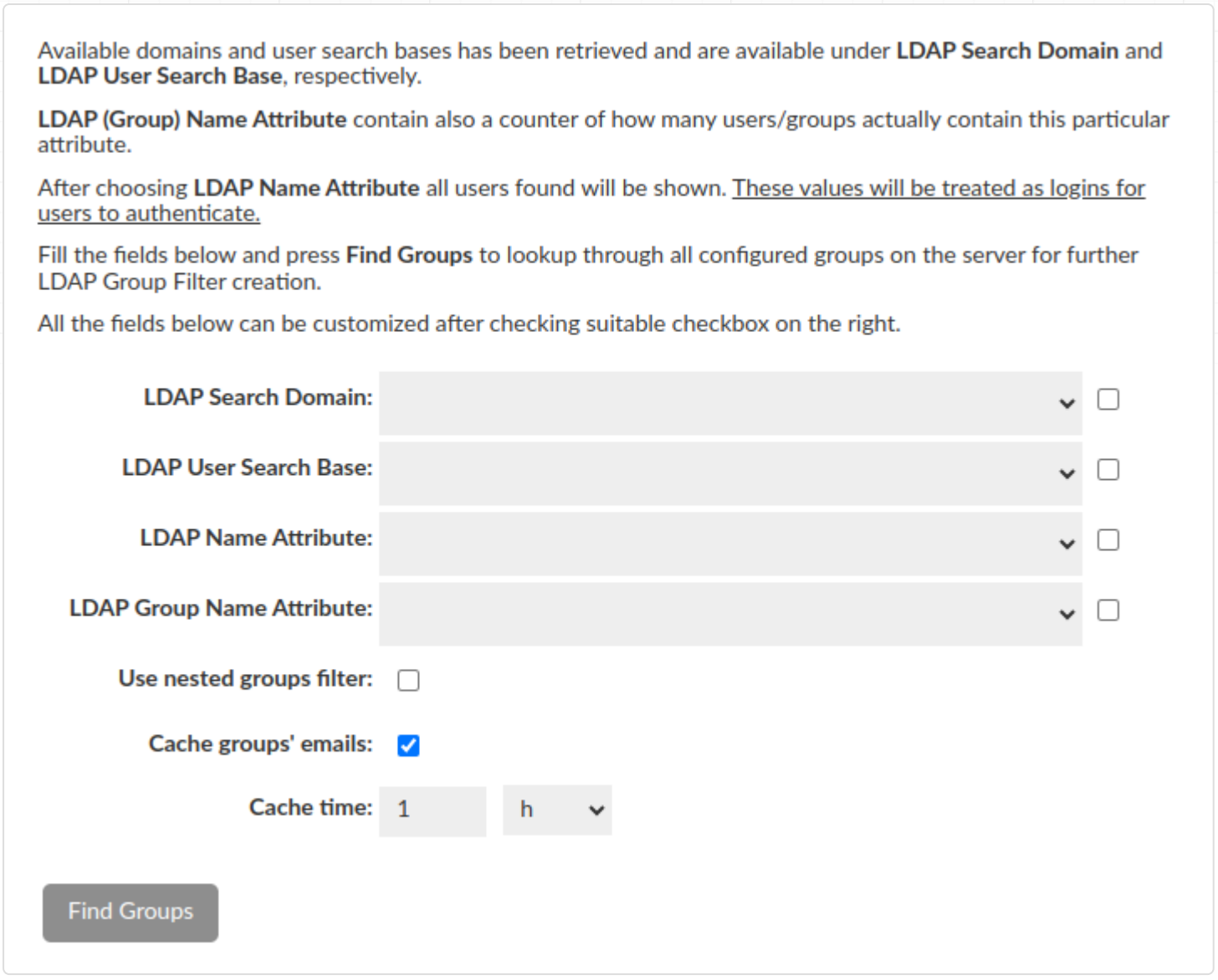
At the very beginning, the absolute basis of the correct and valid configuration:



- The fields above are required as the authentication is made by binding to server as admin and searching information about authenticated user.
- The grayed-out "Verify" button will remain gray until all required fields are filled in.
- When all the required fields are filled in, the button will turn green and when pressed, the connection to AD will be attempted.
- If it fails, a corresponding message with the reason for the error will be displayed.
- if it succeed, the next Wizard stage will be unlocked.

Users & Groups Search Details

The next step is necessary to properly look for information about the user being authenticated and the groups he belongs to + set whether nested groups should be searched for.



- In order to simplify configuration some of the fields can be filled using dropdowns (LDAP Search Domain and LDAP Search Base contains all possible options straight from LDAP Server).
- LDAP (Group) Name Attribute contains all attributes which are actually used among all configured users / groups with additional counter of how many users / groups actually contain particular attribute.
- Additional "Example users found" row will be shown after LDAP Name Attribute selection. This row is just a preview and has no affect on the final configuration, but what is important - these values will be treated as logins for users to authenticate.
- Caching emails for LDAP groups is enabled by default, with a cache update time of 1 hour. This can be customized to fit your needs or disabled altogether. However, the minimum update time must be at least 5 minutes.
- All of these fields can be customized after checking suitable checkbox on the right.
- When all the fields are filled in, the button will turn green and when pressed, the attempt to find Groups configured on the server will be performed.

Available domains and user search bases has been retrieved and are available under **LDAP Search Domain** and **LDAP User Search Base**, respectively.

LDAP (Group) Name Attribute contain also a counter of how many users/groups actually contain this particular attribute.

After choosing **LDAP Name Attribute** all users found will be shown. These values will be treated as logins for users to authenticate.

Fill the fields below and press **Find Groups** to lookup through all configured groups on the server for further LDAP Group Filter creation.

All the fields below can be customized after checking suitable checkbox on the right.

LDAP Search Domain: dc=lab,dc=heimdalldata,dc=com ☐

LDAP User Search Base: cn=users,cn=accounts,dc=lab,dc=heimdalldata,dc=com ☐

LDAP Name Attribute: uid / 20 ☐

Example users found: test1

LDAP Group Name Attribute: cn / 10 ☐

Use nested groups filter: ☐

Cache groups' emails: ☑

Cache time: 1 h

Find Groups

Found groups overall: **10**

Select subset of Groups which will be relevant to roles on the database.

Based on that appropriate LDAP Group Filter will be generated.

If no group is selected then no filter will be provided.

Find Groups: Find

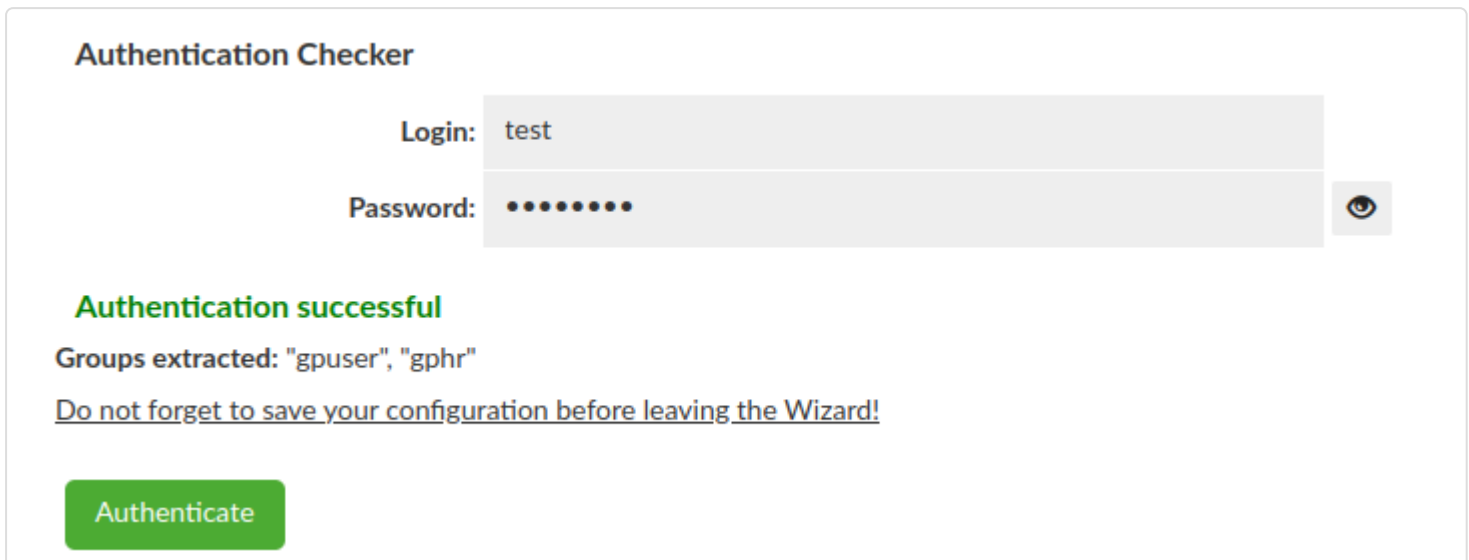
LDAP Group Filter Creation

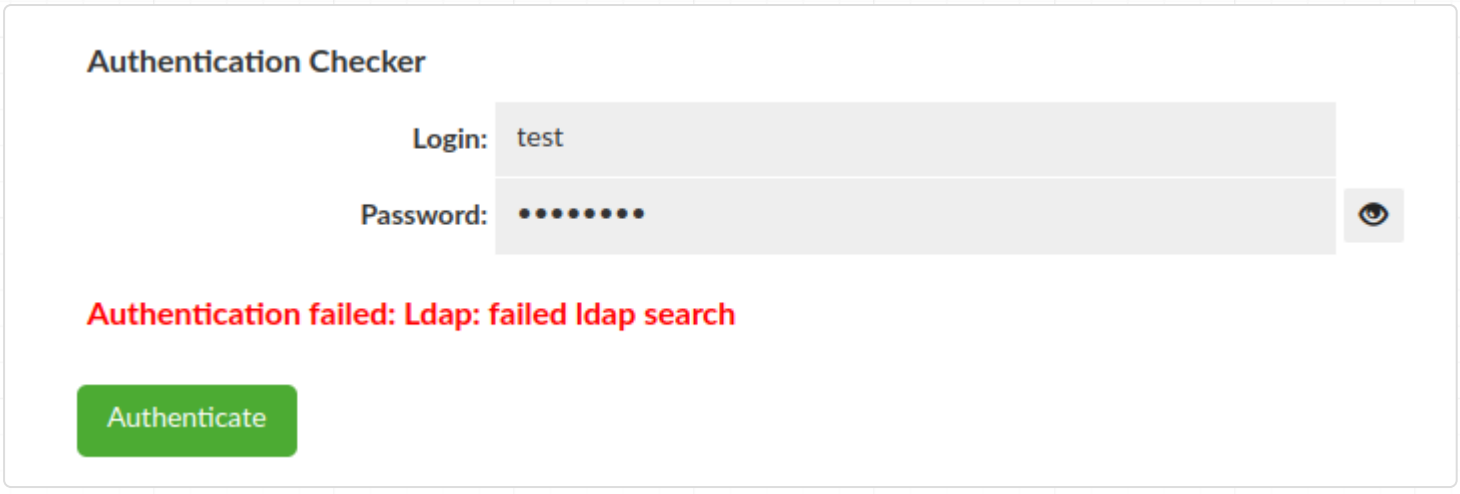
The last step of the Wizard is to select Groups of interest for the DB i.e. Groups which will be relevant to database roles. Based on the selected Group the LDAP Group Filter will be generated when you click "Generate Filter" button. If no group is selected then no filter will be provided and this is ok - LDAP Group Filter optional.

At this point your LDAP Configuration is complete and after saving it can be found under "LDAP Configuration".

Authentication Checker

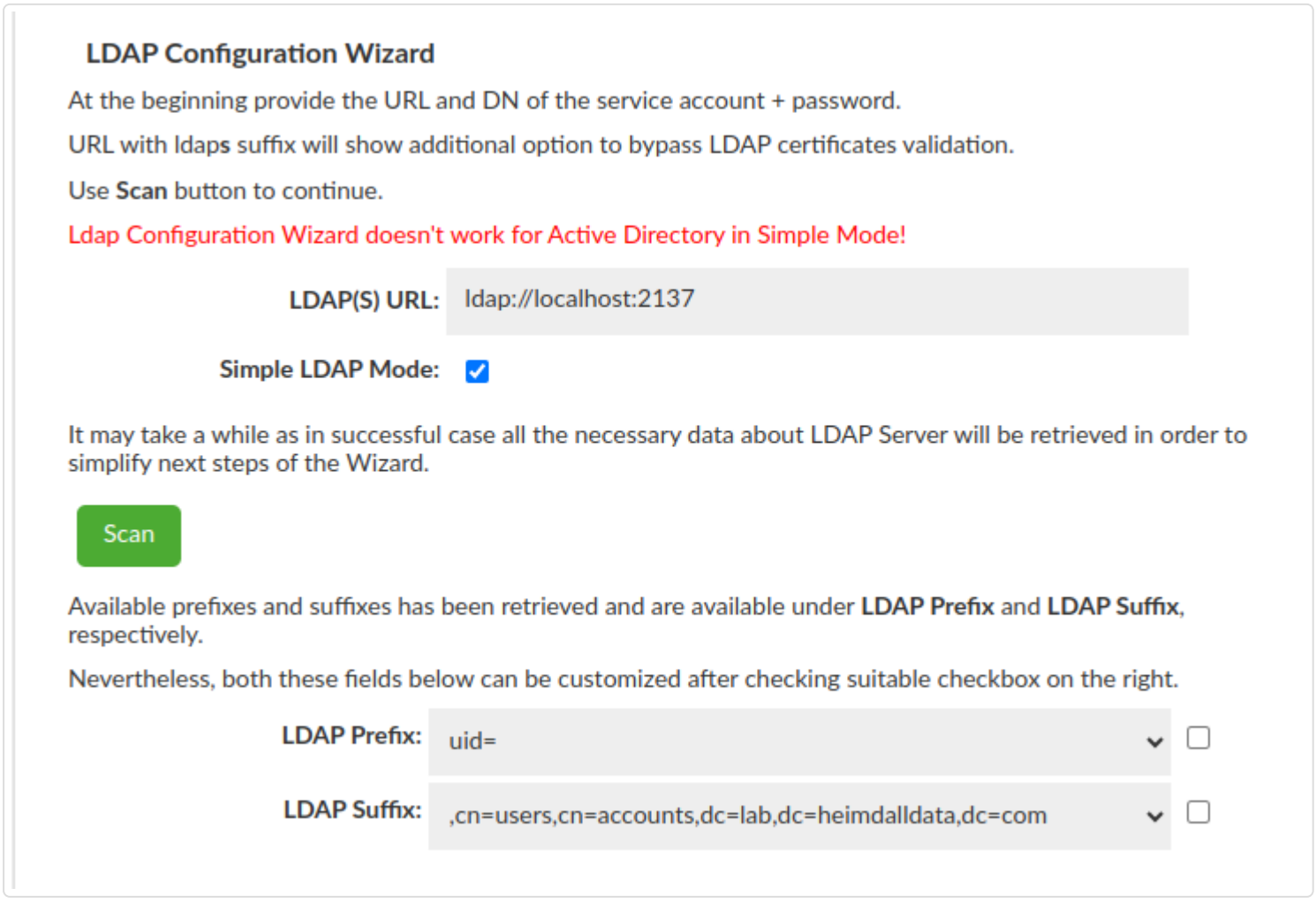
Provided configuration can be verified with the last part of the LDAP Configuration Wizard - Authentication Checker. In case of successful authentication extracted groups will be listed, as long as LDAP Group Filter is specified.





Simple Mode

In Simple Mode, unlike Bind + Search Mode, only the url suffice to start the configuration. Nevertheless what is important, the LDAP Configuration Wizard **does not work with Active Directory**, as this one has a strong security authentication and requires a successful bind before any data can be retrieved. Regardless of the success of the scan the next part of the Wizard wll be unlocked.



After successful scan all possible LDAP Prefixes and LDAP Suffixes will be available in a drop-down list. After failed scan drop-down lists will be empty, nevertheless both these fields can be customized after checking suitable checkbox on the right.

At this point your Simple LDAP Configuration is complete and after saving it can be found under "LDAP Configuration".

Certificates Overview

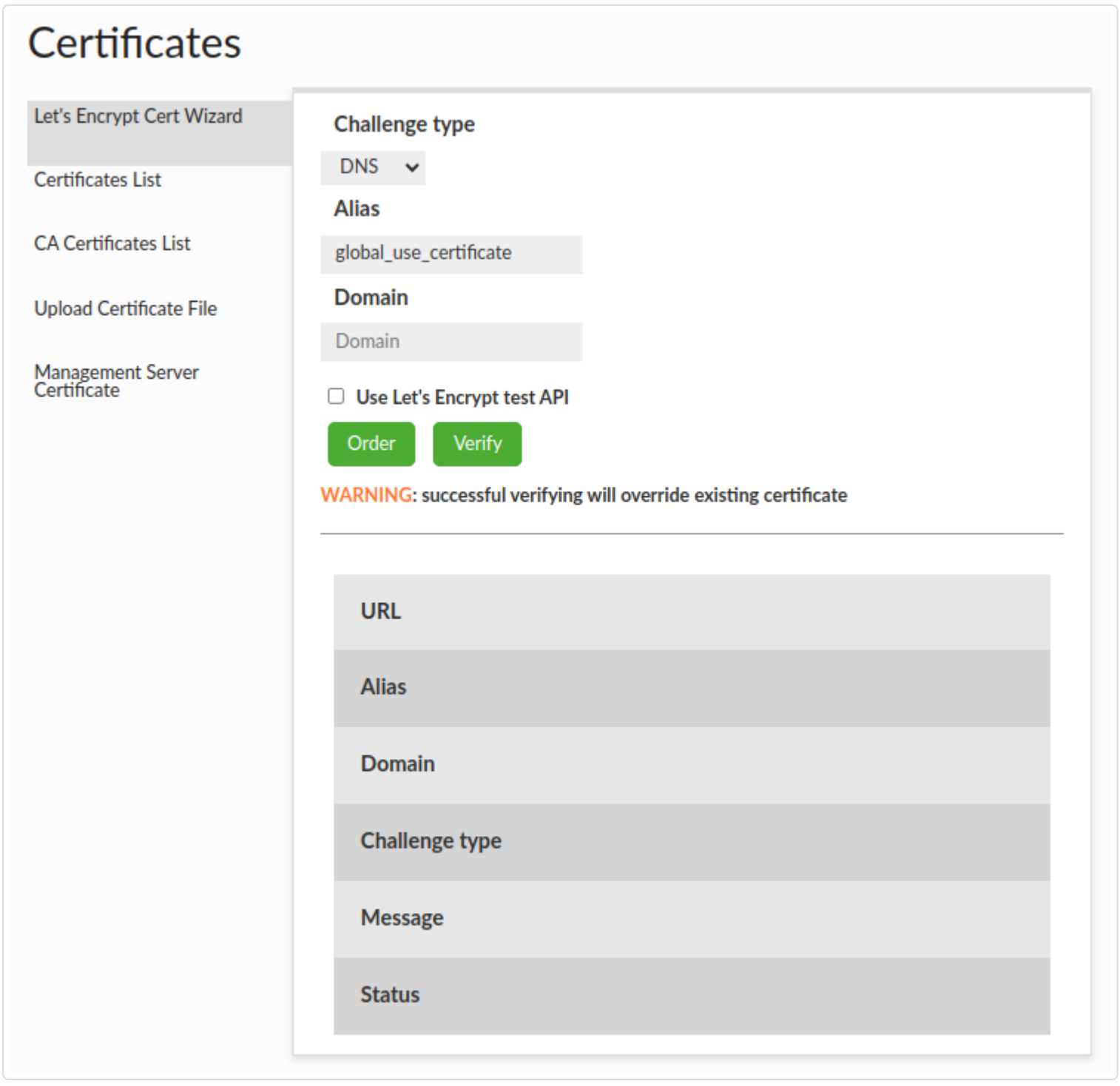
The certificates tab is divided into four sections:

- Let's Encrypt Cert Wizard
- Certificates List
- Upload Certificate File
- Server Management Certificate

These sections are to help you manage certificates in Heimdall.

Let's Encrypt Cert Wizard

Let's Encrypt Cert Wizard section provides a way to simply generate Let's encrypt requests and then the final certificate from this.



Prerequisites

In the certificate tab in Let's Encrypt Cert Wizard section, there are three fields that need to be set to create an order of the Let's Encrypt certificate:

- Challenge type (DNS is default)
- Certificate alias
- Domain

Challenge

Challenges are used to prove ownership of a domain. There are two available challenges DNS and HTTP:

- DNS: You prove to the CA that you are able to control the DNS records of the domain to be authorized, by creating a TXT record with a signed content.

- HTTP: You prove to the CA that you are able to control the website content of the domain to be authorized, by making a file with a signed content available at a given path.

Certificate alias

A certificate alias is a name, used in other places, which is going to be assigned to the generated certificate in the Keystore and Virtual Databases.

Domain

A domain is a subject of the generated certificate.

Flow

In case to generate new Let's Encrypt certificate you must:

1. Set up all fields and click "Order" button. Let's Encrypt API will request order certificate.
2. Make a challenge returned by Let's Encrypt API. You will find instructions in message field.
3. Click "Verify" button. If you did your challenge properly, Let's Encrypt API will generate new Let's Encrypt certificate and will place it in the Keystore and Virtual Databases with given certificate alias.

Be careful !!!

If a certificate alias already exists in Heimdall, the existing certificate will be overridden with a new Let's Encrypt Certificate.

Create order

After you set up all fields and click the "Order" button:

1. Let's Encrypt API creates your unique account in Let's Encrypt CA if you don't have one.
2. Send to Let's Encrypt CA request to create certificate order for the given domain name.
3. Let's Encrypt API returns you a message on how you can prove your ownership, based on the selected challenge type.

Verify order

After you do your challenge and click "Verify" button:

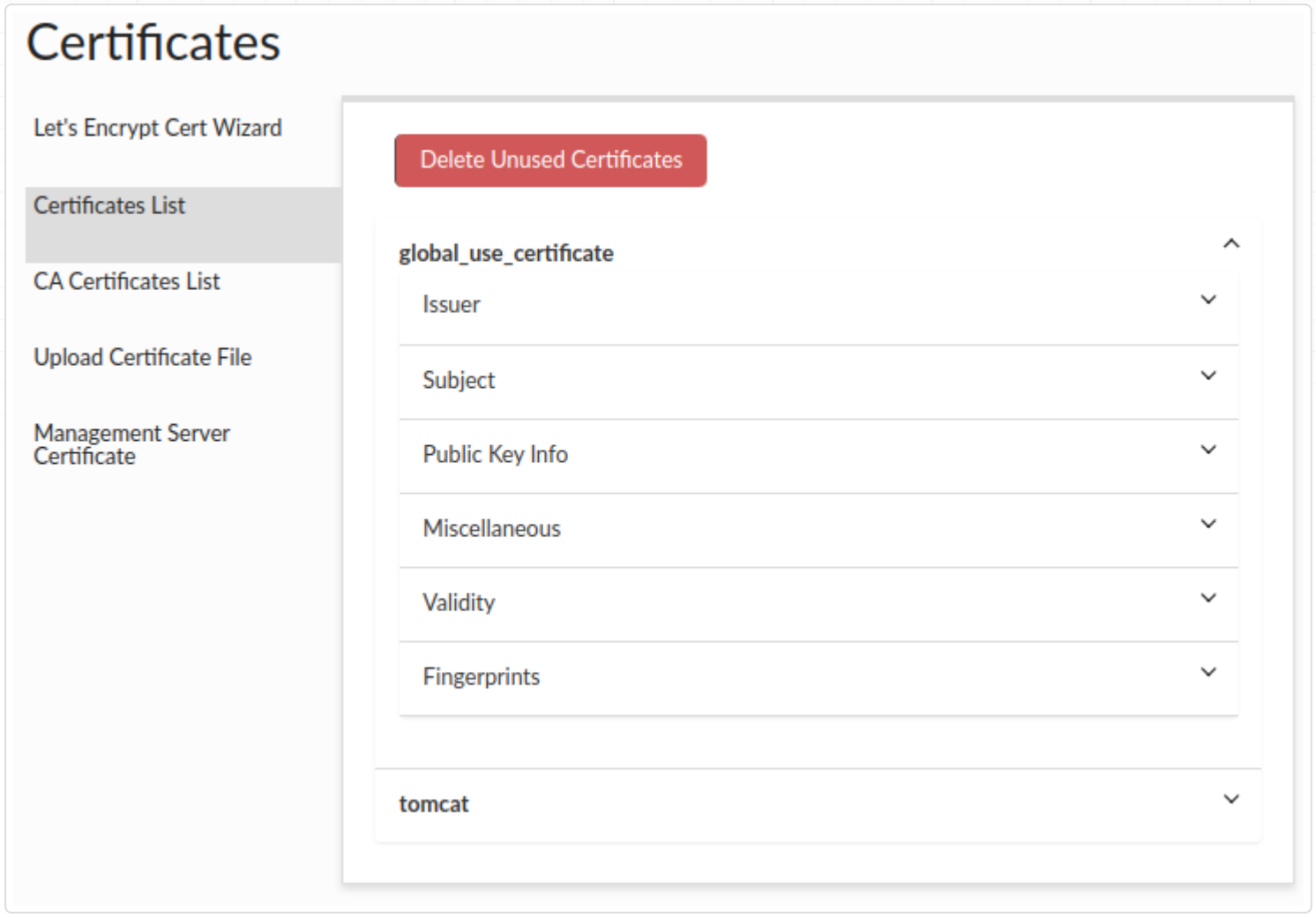
1. Let's Encrypt API send request to Let's Encrypt CA in case to verify your ownership.
2. If verification went properly, API will request for certificate.
3. Generated certificated is being uploaded to Keystore with given alias. If certificate with given alias already exists it will override existing certificate.

Extra information

You can test API by selecting the checkbox "Use Let's Encrypt test API"

Certificates List

There is a list of uploaded certificates shown by the aliases.



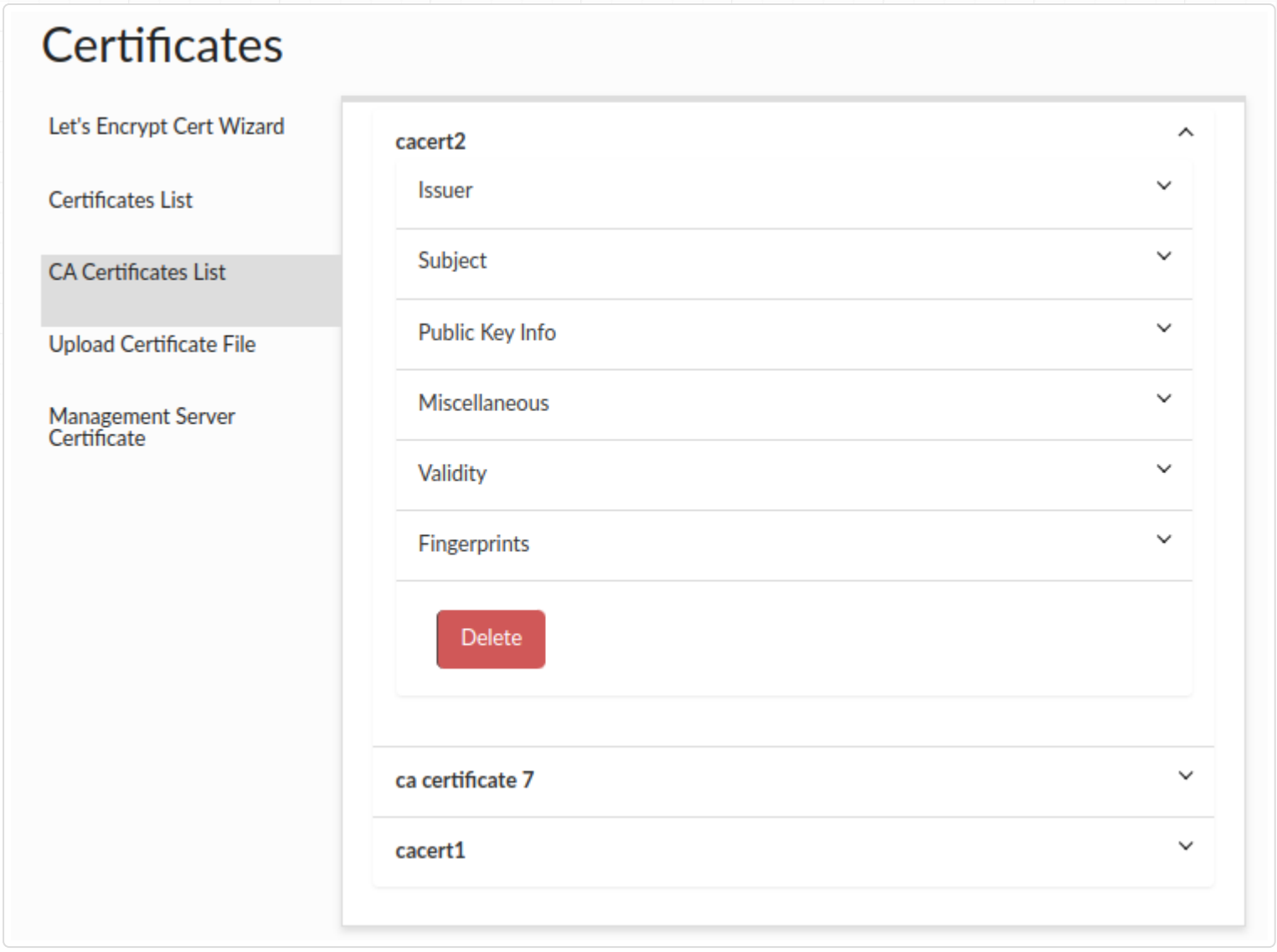
You can check the details of each certificate, such as:

- **Issuer** - The entity or organization that grants or issues the certificate. They are responsible for verifying the qualifications, credentials, or completion of specific requirements that make the recipient eligible for the certificate.
- **Subject** - The individual or entity to whom the certificate is issued. They are the recipient or beneficiary of the certificate.
- **Public Key Info** - The public key information such as: Algorithm - The specific cryptographic method or algorithm used to generate the key pair associated with the certificate, such as RSA, DSA, or ECDSA. Exponent - A component of the public key that is used in encryption and digital signature verification. It is a positive integer associated with the algorithm used. Key Size - The length or size of the cryptographic key, typically measured in bits. A larger key size generally implies increased security against certain types of attacks. Modulus - A mathematical term representing the modulus value used in the cryptographic algorithm. It is part of the public key and plays a crucial role in encryption and signature verification.
- **Miscellaneous** - Additional information such as serial number, signature algorithm, version.
- **Validity** - The period of time, where the certificate is valid.
- **Fingerprints** - A unique identifier generated from the certificate's contents using a specific hash algorithm.

You can remove all unused certificates (which are not used by any vdb) by clicking the "**Delete Unused Certificates**" button.

CA Certificates List

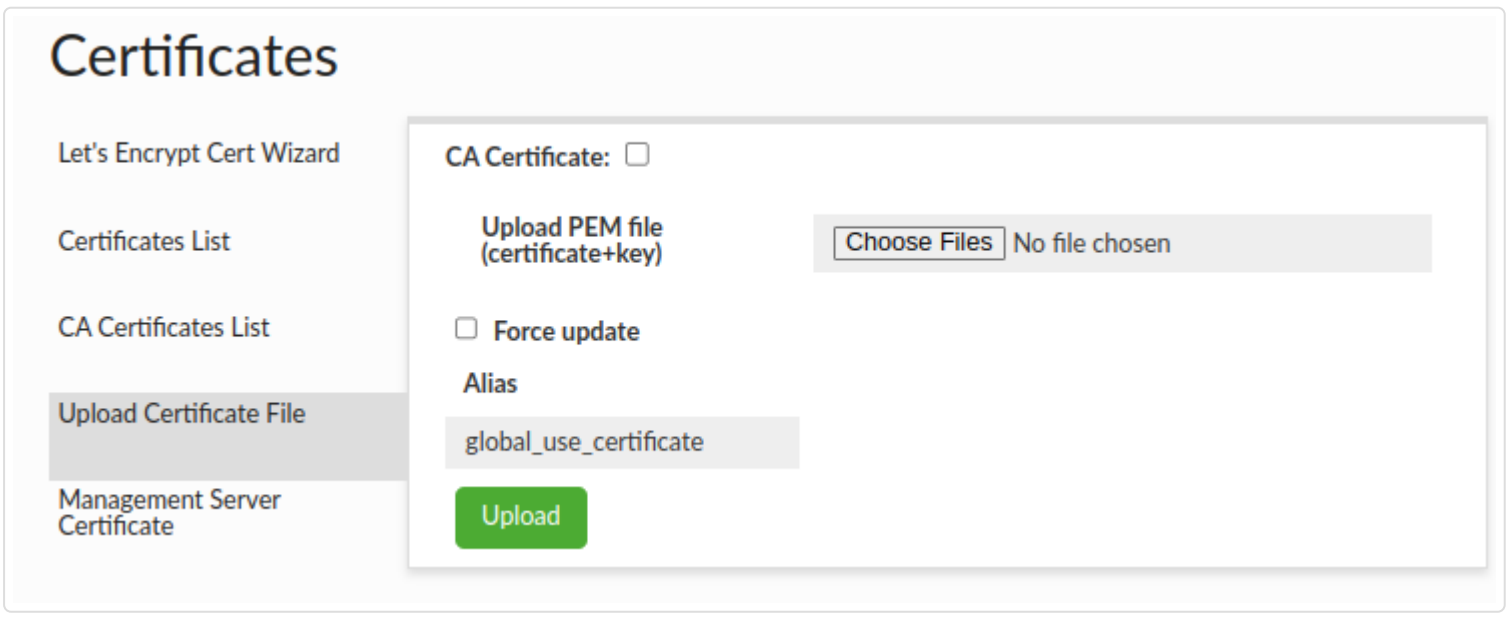
There is a list of uploaded CA certificates shown by the aliases in the same way as above certificates.



Under the details for each CA certificate there is a button to delete a given certificate. Remember! Before deleting the CA certificate, make sure that all proxies are enabled so that the certificate will also be removed from the proxy cacerts file if it is on a different machine than the central manager.

Upload Certificate File

Upload Certificate File section provides a way to upload your own certificate into Heimdall.



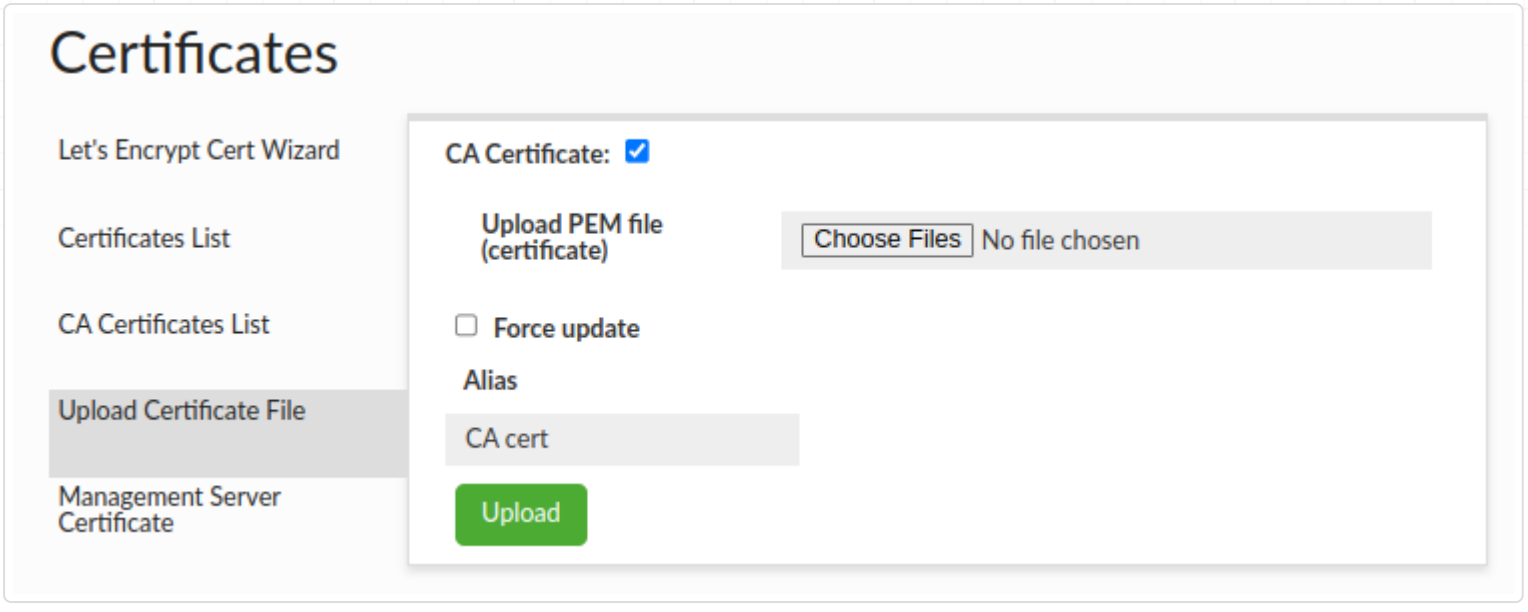
There are 3 fields in upload certificate file interface:

- Combined pem file (A file that contains private key and certificate in pem format with appropriate headers).
- Force update checkbox (If certificate with given alias already exists, force flag must be checked to override existing certificate in Keystore and Virtual Databases).
- Alias (Alias in keystore where certificate will be kept. By default, it's "global_use_certificate" alias).

You can also upload a CA certificate, which will be saved on the central manager side and transferred to all proxies. In both cases it will be saved in the cacerts file with the default password and in the default location appropriate for Java where the central manager/proxy is running.

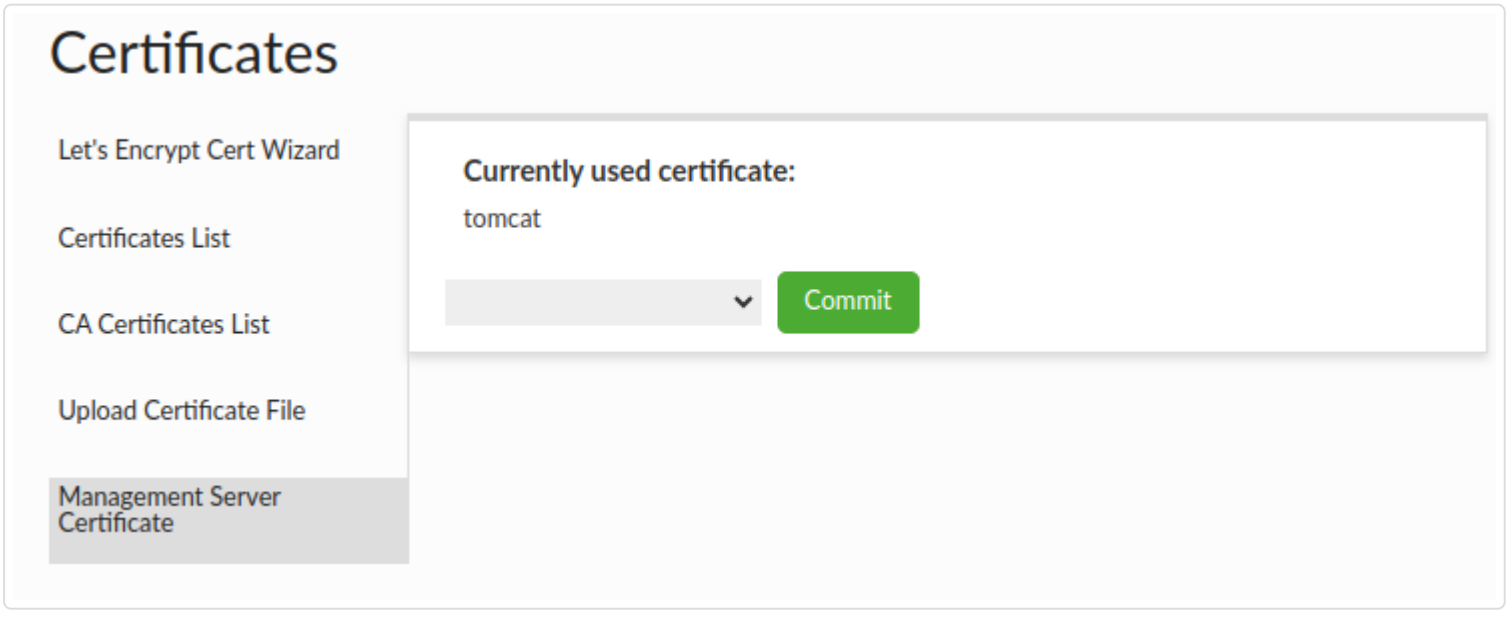
If you want to upload a CA certificate, just select "CA Certificate" checkbox. The pem file should contain only the certificate - public key.

The ca certs aliases are also stored in the ca_certs_aliases.json file in the config directory.



Server Management Certificate

Server management Certificate section provides a way to see your current server management certificate, and also to change your server management certificate with it.



Flow:

1. Choose certificate that you to want use for server management.
2. Click "Commit" button to assign new certificate to the server management.

To start using the new certificate, you must restart the Heimdall.

Database Password Rotation Support (Proxy Authentication)

In order to simplify rotating of database passwords, please follow these steps:

1. Configure a mapping of the old password to new password via a HTTP call, with a browser, curl or other tool. A GUI login is required for the step:

http://demoa.heimdalldata.com:8087/api/tlkv/put?key=admin:heimdalltestasdf&value=heimdalltest&ttl=60000

Here, the old username and password is "admin:heimdalltestasdf" and the new password is "heimdall". The ttl is specified in milliseconds, so the above example expires this mapping in 60 seconds.

1. Change the password on the DB At this point, the password failure should be detected by the driver or proxy, and will be looked up via the configuration set with the URL, above

2. Rotate the passwords on the application/proxy side within the TTL specified. During this window, the old password will be replaced by the new one automatically. Once the TTL is done, the mapping is lost, and any old passwords will again be rejected.

The goal of this feature is to allow a fixed window where the old password can be used to connect to the DB which is configured to use the new password, without errors.

HTTP API reference

In this document you can find description of various HTTP API endpoints that the Heimdall central manager provides.

Reload configuration

- Endpoint URL: **api/config/reload**
- HTTP Method: **GET** (Preferred, It takes all types of methods)

This endpoint allows for reload of Heimdall's configuration (heimdall.conf file).

Usage Example

Usage is pretty straightforward, just remember to authenticate the request. Curl pattern:

curl --user {username}:{password} http://{ipAddress}:{portNumber}/api/config/reload

Where:

- {username} -> Heimdall's username
- {password} -> Heimdall's password
- {ipAddress} -> Heimdall's IP or DNS name
- {portNumber} -> Heimdall's server port number

Example command with default values on local Heimdall instance:

```
curl --user admin:heimdall http://localhost:8087/api/config/reload
```

Export VDB configuration

- Endpoint URL: **api/config/{config}/export/{vdb}**
- HTTP Method: **GET**
- Parameters:
  - **{config}**: String value that controls what is being returned by this endpoint in JSON:
    - "vdb" -> returns whole VDB configuration (current VDB, Data Sources, Drivers, Rules)
    - "source" -> returns Data Sources and Drivers related to VDB
    - "driver" -> returns Drivers related to VDB
    - "policy" -> returns Rules related to VDB
  - **{vdb}**: Name of VDB which details should be exposed
- Response Body: JSON file with chosen properties

This endpoint allows for retrieval of selected properties of single VDB.

Usage Example

To effectively use this endpoint, provide a name of VDB to export and what should be extracted as request parameters. Call needs to be authenticated. Curl pattern:

curl --user {username}:{password} http://{ipAddress}:{portNumber}/api/config/{config}/export/{vdb}

Where:

- {username} -> Heimdall's username
- {password} -> Heimdall's password
- {ipAddress} -> Heimdall's IP or DNS name
- {portNumber} -> Heimdall's server port number
- {config} -> string value determining what's exported (described above)
- {vdb} -> name of VDB to export

Example command with default values on local Heimdall instance:

```
curl --user admin:heimdall http://localhost:8087/api/config/vdb/export/MySQLVirtualDatabase
```

Import VDB configuration

- Endpoint URL: **api/config/import/{config}**
- HTTP Method: **POST**

- Parameters:
    - **{config}**: String value that controls what is being imported:
        - "vdb" -> updates all VBD properties
        - "source" -> updates Data Sources of VDB
        - "driver" -> updates Drivers of VDB
        - "policy" -> updates Rules of VDB
- Request Body: JSON file with properties of VDB

This endpoint allows for import of single VDB with all related properties (Data Sources, Drivers, Rules).

Usage Example

To use this endpoint, provide what should be imported as a {config} parameter. File with desired properties should also be provided. Call needs to be authorized too. Curl pattern:

curl --user {username}:{password} --form files=@{fileName} http://{ipAddress}:{portNumber}/api/config/import/{config}

Where:

- {username} -> Heimdall's username
- {password} -> Heimdall's password
- {fileName} -> file with VDB properties to import
- {ipAddress} -> Heimdall's IP or DNS name
- {portNumber} -> Heimdall's server port number
- {config} -> string value determining what's imported (described above)

Format of {fileName}

```
{
  "vdbs": [
    {
      "acCaseSensitiveFields": false, // defines if field names are case sensitive
      "acCaseSensitiveTables": false, // defines if tables names are case sensitive
      "accessType": "mysql", // is underlying dataBase: sqlsever, mysql, postgres
      "cacheEnabled": false,
      "pciHipaaCache": false, // PCI/HIPAA compliant cache
      "cacheSize": 50000000, // size of cache in bytes
      "cacheSoftValues": true, // Determine use of soft references
      "cacheMaxExpiry": 0, // Maximum cache object age
      "cacheObjectCount": 0, // Maximum cache object count
      "cacheSizeRate": 1, // cache size unit rate
      "cacheType": "hazelcast", // type of cache
      "cloudMetrics": false, // determines if cloud specific metrics should be used
      "clusterCacheManager": false, // determines use of ClusterCacheManager
      "customKey": false, // false -> use vdb, catalog, user if true use custom cache key
      "debugMode": true, //debugMode for VDB
      "gridCacheName": "heimdall", // name of the cache to use
      "gridPort": 0, // grid cache port
      "databaseNumber": 0, // grid cache database number
      "id": 1606394877786, // config id
      "jdbcClass": "com.heimdalldata.HeimdallDriver", // class of JDBC driver
      "jdbcUrl": "jdbc:heimdall://\u003cHeimdall Server IP:port\u003e/MySQLVirtualDatabase?hduser\u003d\u003cuser\u003e\u0026hdpassw
      "excludeKeyCatalog": false, // True if exclude catalog part of the hash key for query result cache
      "excludeKeySchema": false, // True if exclude schema part of the hash key for query result cache
      "excludeKeySqlComments": false, // True if exclude sql comments part of the hash key for query result cache
      "excludeKeyUser": false, // True if exclude connected database user part of the hash key for query result cache
      "excludeKeyVdb": false, // True if exclude virtual database name part of the hash key for query result cache
      "localCacheEnable": true,
      "localhostOnly": false,
      "logConnections": true, // determines if Heimdall should log connections
      "loggingEnabled": true,
      "logMethods": false, // determines if method calls should be logged
      "logResultSetMethods": false, // determines if ResultSet method calls should be logged
      "logSql": true, // determines if SQL statements should be logged
      "logAuthentications": false, // determines if authentication should be logged
      "mgmtProxyEnable": true, // determines if managment proxy should be enabled
      "tlsEnable": false,
      "tlsRequired": false,
      "ldapAuthEnabled": false,
      "delayedTransaction": false, // determines if delayed transactions should be enabled
      "paranoia": false, // determines paranoia mode, which when enabled logs much more information
      "multiplex": false, // determines if multiplexing should be enabled
      "multiplexTimeout": 0,
      "trackQueryDistributionCount": 1000, // sets max number of queries to track
      "ldapGroupFilter": "", //ldap filter
      "ldapSimpleMode": false,
      "ldapPrefix": "",
      "ldapSuffix": "",
      "synchAuthentication": false, // determine if Authenticaton should be synchronized
      "proxyauthEnabled": false, // determines if proxy authorization should be enabled
      "authMode": "proxy", // type of authorization: proxy, ldap, sql, passthrough
      "proxyAddress": "0.0.0.0",
      "proxyPort": "3307",
      "xmx": 600, // max heap value for proxy
      "revalidateCache": false, // determines if cache revalidation should be enabled
      "secure": false, // is HTTPS on?
      "shipConsoleLogs": true, // determines if driver/proxy console logs should be sent to the server
      "writeLogsToFile": false, // determines if logs should be written to file
      "sqlDrivenAuthEnabled": false, // should SQL Authorization be used?
      "authorizationQuery": "select * from heimdall.pg_hba where enabled \u003d true order by line_number asc", // query to use duri
      "accessKey": "HIBa3RN0BpDovDkP", // configuration accessKey
      "secretKey": "1nCAUUqnxqblJF74", // configuration secretKey
      "useSsl": false, // determines if SSL should be used
      "verifyPeer": false, // determines if SSL Certificate should be verified
      "jmxHostname": "${hostname}",
      "jmxPort": 0,
      "testSSL": false,
      "userCaseSensitive": false, // determines if username if case sensitive
      "enabled": true, // determines if VDB should be enabled
      "file": "config/MySQLVirtualDatabase_1.conf", // path to VBD configuration file
      "name": "MySQLVirtualDatabase", // name of VDB
      "version": 1, // version of VDB configuration
      "dataSources": [ // array of Data Sources associated with VDB
        {
          "enabled": true, // determines if Data Source should be enabled
          "file": "config/MySQLDataSource_4.conf", // path to Data Source configuration file
          "name": "MySQLDataSource", // name of Data Source
```

```
    "version": 4 // version of Data Source configuration
  }
],
"rules": [ // rules associated with VDB
  {
    "enabled": true, // determines if Rules should be enabled
    "file": "config/RulesForAll_1.conf", // path to Rules configuration file
    "name": "RulesForAll", // name of Rules
    "version": 1 // version of Rules configuration
  }
],
"properties": [ // properties associated with VDB
  {
    "key": "",
    "value": ""
  }
],
"users": [ // underlying database credentials
  {
    "user": "root",
    "password": "password"
  }
],
"pinnedClients": [], // array of pinned clients
"modules": [ // array of Heimdall jars
    "modules/heimdallloadbalancer-1.0-20.11.26.1.jar",
    "modules/heimdallredis-5.3.0-20.11.26.1.jar",
    "modules/heimdallforward-1.0-20.11.26.1.jar",
    "modules/heimdallmysql-1.0-20.11.26.1.jar",
    "modules/heimdallsqlserver-1.0-20.11.26.1.jar",
    "modules/heimdallpostgres-1.0-20.11.26.1.jar",
    "modules/heimdalltrigger-1.0-20.11.26.1.jar",
    "modules/heimdallauth-1.0-20.11.26.1.jar",
    "modules/heimdallhazelcast-3.6.8-20.11.26.1.jar",
    "modules/heimdallextractplan-1.0-20.11.26.1.jar",
    "modules/heimdallfirewall-1.0-20.11.26.1.jar",
    "modules/heimdallasync-1.0-20.11.26.1.jar",
    "modules/heimdallaws-1.0-20.11.26.1.jar"
  ]
 }
],
"sources": [ // Data Source configuration associated with VDB
  {
    "url": "jdbc:mysql://localhost:3306/myDb?allowPublicKeyRetrieval\u003dtrue\u0026useSSL\u003dfalse", // Data Source JDBC url
    "driverConfig": {
      "enabled": true, // determines if Driver associated with Data Source is enabled
      "file": "config/MySQLDriver_1.conf", // path to Driver's configuration file
      "name": "MySQLDriver", // name of Driver
      "version": 1 // version of Driver configuration
    },
    "driverClass": "com.mysql.jdbc.Driver", // class of JDBC Driver
    "driverUrl": "/drivers/MySQLDriver/mysql-connector-java-5.1.49.jar", // path to JDBC Driver
    "properties": {
      "user": "root", // database user
      "password": "password", // database password
      "url": "jdbc:mysql://localhost:3306/myDb?allowPublicKeyRetrieval\u003dtrue\u0026useSSL\u003dfalse", // database url
      "testQuery": "SELECT 1"
    },
    "usepool": false, //determines if connection pooling should be used
    "poolProperties": {}, // key-value object of connection pooling properties
    "servers": [ // array of Data Surce Servers configurations associated with Data Source
      {
        "name": "Primary", // server name
        "url": "jdbc:mysql:// localhost:3306/myDb?allowPublicKeyRetrieval=true&useSSL=false", // server JDBC url
        "enabled": true,
        "writeable": true, // true for master, false for slave
        "weight": 1, // relative amount of load server can handle
        "readWeight": 1, // read weight for server
        "writeWeight": 1, // write weight for server
        "active": true,
        "failed": true,
        "maxLag": 2147483647
      }
    ],
    "useLoadBalancing": true,
    "trackClusterChanges": false,
    "desiredWriteCapacity": 1, // desired write capacity for all writeable nodes
    "desiredReadCapacity": 10, // desired read capacity
    "trackReplicationLags": true,
```

```
    "timeWindow": 10000, // safe window for replication lag in ms
    "useMonitoring": false, // determines if server monitoring should be enabled
    "holdTime": 30000, // time to hold connections/operations waiting for valid server to be available
    "obscure": {
      "secretKey": "Odrv3jRI9sliFSZUFXwbynSf68Llbi" // key to obscure password with
    },
    "enabled": true, // determines if Data Source should be enabled
    "file": "config/MySQLDataSource_4.conf", // path to Data Source configuration file
    "name": "MySQLDataSource", // name of Data Source
    "version": 4 // version of Data Source configuration
  }
],
"drivers": [ // array with Drivers associated with VDB
  {
    "jdbcClass": "com.mysql.jdbc.Driver", // JDBC Driver class
    "uploadedDrivers": [
      "/drivers/MySQLDriver/mysql-connector-java-5.1.49.jar" // path to Driver
    ],
    "autoUpdate": true,
    "enabled": true, // determines if Driver should be enabled
    "file": "config/MySQLDriver_1.conf", // path to Driver configuration file
    "name": "MySQLDriver", // name of Driver
    "version": 1 // version of Driver configuration
  }
],
"rules": [ // array with Rules associated with VDB
  {
    "rules": [
      {
        "enabled": true, // determines if Rule should be enabled
        "type": "C", // type of rule, description below
        "patterns": [ // array with regex patterns for rule
          ""
        ],
        "intrans": false, // determines if Rule can be aplied whithin transacion
        "properties": { // key-value parameters of Rule
          "ttl": "5*60000"
        }
      },
      {
        "enabled": true, // determines if Rule should be enabled
        "type": "C", // type of rule, description below
        "patterns": [ // array with regex patterns for rule
          ""
        ],
        "intrans": true, // determines if Rule can be aplied whithin transacion
        "properties": { // key-value parameters of Rule
          "ttl": "5*60000"
        }
      },
      {
        "enabled": true, // determines if Rule should be enabled
        "type": "V", // type of rule, description below
        "patterns": [ // array with regex patterns for rule
          "(?i)^select"
        ],
        "intrans": false, // determines if Rule can be aplied whithin transacion
        "properties": { // key-value parameters of Rule
          "lagignore": "false"
        }
      }
    ],
    "enabled": true, // determines if Rules should be enabled
    "file": "config/RulesForAll_1.conf", // path to Rules configuration file
    "name": "RulesForAll", // name of Rules
    "version": 1 // version of Rules configuration
  }
]
}
```

Rule Types

```
Q -> allow
A -> async execute
C -> cache
G -> call
Z -> debug
D -> drop
O -> extract plan
F -> forward
I -> ignore
P -> learn pattern
L -> log
B -> nocache
M -> pool
V -> reader eligible
N -> result
E -> retry
S -> stack trace
W -> table cache
X -> tag
T -> transform
R -> trigger
```

Example command with default values on local Heimdall instance:

```
curl --user admin:heimdall --form files=@file.json  http://localhost:8087/api/config/import/vdb
```

Export Data Source Servers configuration

- Endpoint URL: **api/config/source/{name}/servers**
- HTTP Method: **GET**
- Parameters:
  - **{name}**: Name of Data Source Servers to export
- Response Body: JSON file with list of Data Source Servers Properties.

This endpoint allows for retrieving Data Source Servers configuration by name.

Usage Example

Provide a name of existing VDB and credentials. Curl pattern:

curl --user {username}:{password} http://{ipAddress}:{portNumber}/api/config/source/{name}/servers

Where:

- {username} -> Heimdall's username
- {password} -> Heimdall's password
- {ipAddress} -> Heimdall's IP or DNS name
- {portNumber} -> Heimdall's server port number
- {name} -> name of existing Data Source

Example command with default values on local Heimdall instance:

```
curl --user admin:heimdall http://localhost:8087/api/config/source/MySQLDataSource/servers
```

Import Data Source Servers configuration

- Endpoint URL: **api/config/source/{name}/servers**
- HTTP Method: **POST**
- Parameters:
  - **{name}**: Name of Data Source to import Data Source Servers configuration to
- Request Body: JSON file with list of Data Source Servers Properties.

This endpoint allows for creating new Data Source Server configuration in existing Data Source.

Usage Example

Provide a name of existing VDB and new Data Source Server properties, to update it. This endpoint is secured, so we have to use authentication. Request must have Header of content-type set to application/json. Curl pattern:

curl --user {username}:{password} --data-binary @{filename} -H "Content-Type: application/json" http://{ipAddress}:{portNumber}/api/config/source/{name}/servers

Where:

- {username} -> Heimdall's username
- {password} -> Heimdall's password
- {filename} -> name of file with DataSourceServer properties
- {ipAddress} -> Heimdall's IP or DNS name
- {portNumber} -> Heimdall's server port number
- {name} -> name of existing DataSource

Format of {filename}:

```
    [
        {
          "name": "Primary", // server name
          "url": "jdbc:mysql:// localhost:3306/myDb?allowPublicKeyRetrieval=true&useSSL=false", // server JDBC url
          "enabled": true,
          "writeable": true, // true for master, false for slave
          "weight": 1, // relative amount of load server can handle
          "readWeight": 1, // read weight for server
          "writeWeight": 1, // write weight for server
          "active": true,
          "failed": true,
          "maxLag": 2147483647
        }
    ]
```

Example command with default values on local Heimdall instance:

```
curl -v --output - --user admin:heimdall --data-binary @file.json -H "Content-Type: application/json"  http://localhost:8087/api/con
```

Heimdall and Open Source Licenses

Heimdall leverages a large number of open source projects and runs on Open source Linux.

The following base OS distributions are currently used, depending on the environment they are started from:

- AWS Marketplace: AWS Linux 2.0
- Azure Marketplace: Ubuntu 18.04 LTS
- Google Cloud Marketplace: Ubuntu 18.04 LTS
- Alibaba Cloud Marketplace: Alibaba Cloud Linux 2
- Docker: Defaults to Ubuntu 18.04, but this can be changed

Heimdall does not use any non-standard Linux packages besides those provided by the distribution, nor does it modify the Linux kernel or environment except through configuration files. All packages installed into the base OS leverage the distribution specific package manager, i.e. Apt for Debian/Ubuntu, or Yum for Centos/Redhat based environments.

Dependencies

In order to ensure all information is correct and updated At compile time, dependency lists are generated in three formats automatically (below). License URL's are as provided via the Maven POM for Java libraries, as submitted by the project maintainer, or in the case a POM is not available or complete, based on the public project website and maintained by Heimdall.

- text license list broken out per dependency w/ unique POM license text included inline
- text license list sorted by license
- CSV license list with project link for source access, and license summary

Open Source Code Access

To the extent any open source subcomponents are licensed under the EPL and/or other similar licenses that require the source code and/or modifications to source code to be made available (as would be noted above), you may obtain a copy of the source code corresponding to the binaries for such open source components and modifications thereto, if any, (the "Source Files"), by downloading the Source Files from the documented project location in the linked files above, or by sending a request, with your name and address to:

```
Heimdall Data, Inc
37070 Newark Blvd., Suite B
Newark, Ca. 94560
```

or email info@heimdalldata.com. All such requests should clearly specify:

```
OPEN SOURCE FILES REQUEST
Attention General Counsel
```

Heimdall shall mail a copy of the Source Files to you on a CD or equivalent physical medium. This offer to obtain a copy of the Source Files is valid for three years from the date you acquired this Software product.

License Text

For reference, the standard formatted version of all licenses referenced by included code is included below. In the linked text above, the exact license including dates (as appropriate) can be found.

Apache License, Version 2.0; Apache 2.0; The Apache License, 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable

by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special,

```
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Go License

```
Copyright (c) 2009 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

   * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following disclaimer
in the documentation and/or other materials provided with the
distribution.
   * Neither the name of Google Inc. nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

HSQLDB License, a BSD open source license

COPYRIGHTS AND LICENSES (based on BSD License)
For work developed by the HSQL Development Group:

Copyright (c) 2001-2020, The HSQL Development Group
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

Neither the name of the HSQL Development Group nor the names of its
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL HSQL DEVELOPMENT GROUP, HSQLDB.ORG,
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

For work originally developed by the Hypersonic SQL Group:

Copyright (c) 1995-2000 by the Hypersonic SQL Group.
All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this
list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

Neither the name of the Hypersonic SQL Group nor the names of its
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE HYPERSONIC SQL GROUP,
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the
Hypersonic SQL Group.

The 3-Clause BSD Licenses

```
Copyright <YEAR> <COPYRIGHT HOLDER>

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following condition

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in

3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived fro

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NO
```

BSD Nuclear clause

```
You acknowledge that this software is not designed or intended for
use in the design, construction, operation or maintenance of any
nuclear facility.
```

The MIT License

```
Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (th

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF M
```

The Postgres License

```
PostgreSQL Database Management System
(formerly known as Postgres, then as Postgres95)

Portions Copyright © 1996-2020, The PostgreSQL Global Development Group

Portions Copyright © 1994, The Regents of the University of California

Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a writ

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DA

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHAN
```

CC0

```
Licensed under Public Domain (CC0)

To the extent possible under law, the person who associated CC0 with
this code has waived all copyright and related or neighboring
rights to this code.

You should have received a copy of the CC0 legalcode along with this
work. If not, see <http://creativecommons.org/publicdomain/zero/1.0/>.
```

Configuration Life-cycle Management

Backup and Restore of VDB configurations

The easiest way to manually backup and restore vdb configurations is to use the menu icon in the upper right of the VDB menu, and use export. This will save a single json file that aggregates the vdb, data source and rules configurations for a given VDB. The same menu can be used to import a VDB (refresh the screen after an import). If there is an existing vdb of the same name on import, the new entry will be named with _1, _2, etc. If the nested configurations (source and rules) exist with the same name on import, they will be reused as part of the imported configuration.

File Based Management

For those wishing to automate backup and restore, the files of the management server configuration can be accessed directly. The configuration files exist in the /opt/heimdall/config directory (accessible only by root) by default. There are several techniques for managing these:

- Use EFS or another mounted filesystem to keep the config directory accessible in a shared location;
- Use chef or puppet to provide the configuration files;
- and; Use the api/config/reload api call to update the configuration in-memory when any change is made.

S3 Bucket Based Load

On the initial startup of a management server, the configuration can be loaded via an S3 bucket using the user data variable "configSource". This is documented in the AWS Cloudformation template.

Recovery

In the event that a proxy has failed in a standalone configuration, it should be sufficient to simply restart the proxy as per the initial deployment. As the proxy will pull the current driver version from the central manager, this will ensure that the proxy matches the central manager version, even if deployed from an older IAM.

In the event of a central manager failure, it typically can be simply restarted. If this fails, it too can be redeployed from the baseline IAM, using the S3 bucket based loadn or shared filesystem to initialize the instance with the proper configuration. If there are any questions on how to do this in your environment, please contact Heimdall support.

Getting Support

Please check our website at https://www.heimdalldata.com/support/ for all information on contacting support, support tiers, SLA's etc.

Accelerating Support

In nearly any case, a summary of the issue being faced will be requested, as well as a copy of the logs. If the logs can be provided on the initial contact with support, this often dramatically accelerates the time to resolution.

When providing screenshots, please use the screenshot tool denoted by the camera icon on the header row (shown below). This tool will capture the current page, and bundle it with the logs to provide to support.

Request Access Overview

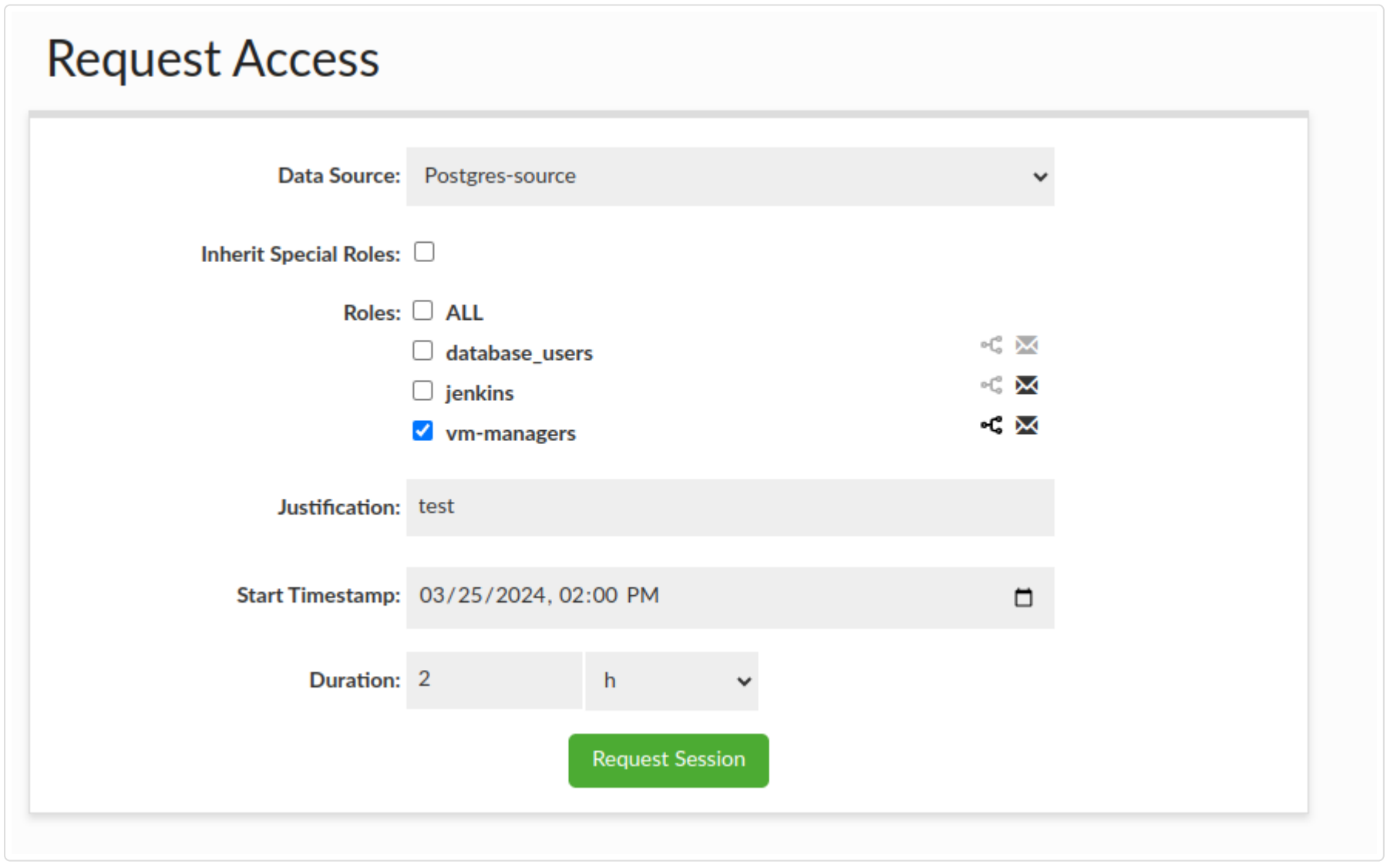
The Request Access tab within the Portal provides the capability to request a session for specific roles in the designated Data Source.

Upon submitting a session request, the relevant approvers will be notified, possessing the authority to approve or deny access to the specified roles.

Only after receiving approval for all roles, the session will be approved, and a temporary user will be created. The requester will then be able to log in to the database using the specified temporary user credentials for a defined period.

There are several possible scenarios during role requests:

1. There may be a situation where a particular role cannot be selected at all. This means that the requesting user is also the only approver for this role and cannot approve themselves.
2. After selecting a role, there might be an alert indicating that the configuration is incomplete. This suggests that there may not be a configuration created in Roles Management for that role or its mapped roles. Additionally, the envelope icon will be grayed out with the information "No approvers found" upon hovering over it.
3. It is also possible that immediately after requesting a specific role in a session, it gets auto-approved. This means that a notification is attached to that role without being associated with any emails. In other words, no approvals are required for such a role.



Fields

- **Data Source**: Data Source in which the user wishes to request access to roles.
- **Inherit special permissions**: If selected, roles will be inherited. This option is available for PostgreSQL, where roles are not inherited by default. By checking this checkbox, it will be possible to inherit special roles such as SUPERUSER, CREATEDB, REPLICATION, BYPASSRLS, LOGIN or CREATEROLE if any of the ordinary roles possesses them.
- **Roles**: Roles that the user wishes to have in the database during the session. The list of roles consists of roles that match the user's LDAP group memberships and roles available in the database. Moreover, for a role to appear on this list, it must be configured in Roles Management, taking into account Group Mappings if they exist for the role.

- **Justification**: Information for approvers justifying why the user is requesting such a session. If the 'Require specific justification format' checkbox is selected in Portal Configuration, justification is required; however, if it does not match the provided regex, an appropriate message will appear.
- **Start Timestamp**: Indicates the time when the approved session is scheduled to start. It can be set to a time in the past.
- **Duration**: Session duration, with the option to choose in various time units. The duration cannot be greater than the set maximum time for the selected role; otherwise, it will not be possible to request a session.

And there are also two icons next to each role. The first one indicates whether there is any mapping to other roles in the Source tab, and the second one shows the emails of approvers associated with that role. The envelope icon is visible only when this option is selected in the Admin tab.

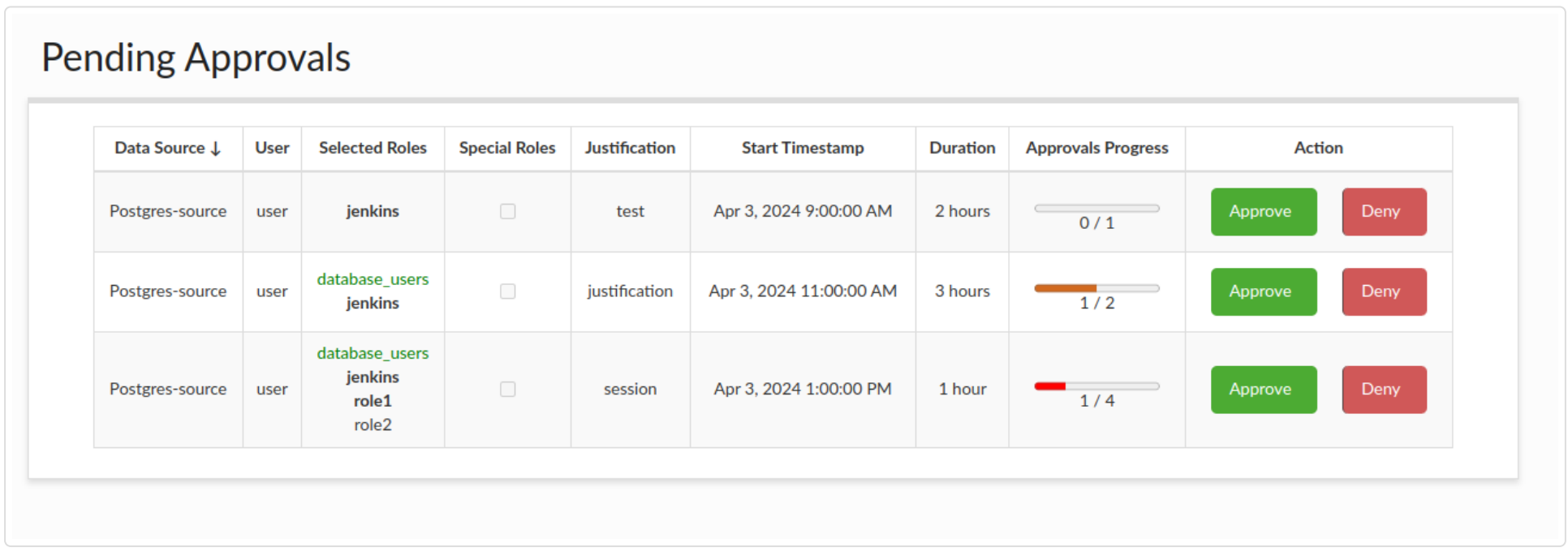
Warning: This feature is **NOT** functional when using MySQL with version < 8.

Pending Approvals Overview

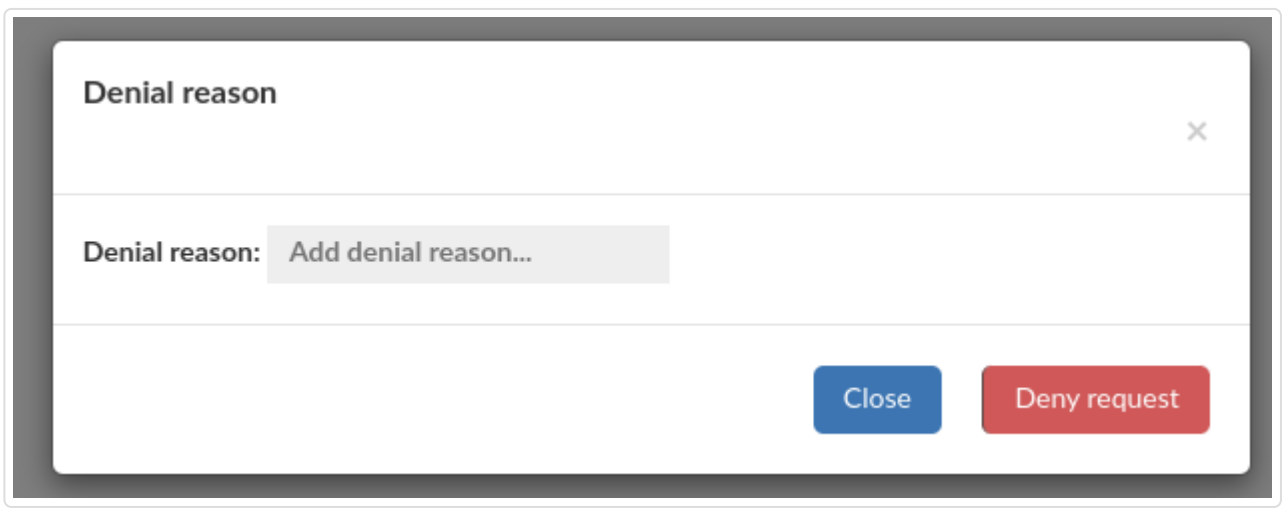
The Pending Approvals tab displays all requested sessions that have not yet received approval or denial and have a status of "CREATED". Only sessions that a particular user (approver) can approve or reject are shown. The relevant columns provide information such as the session request date, data source, user, requested roles, justification, whether the inheritance of special roles is requested, approvals' progress bar and the session duration. The roles that will be approved by clicking the "Approve" button are in bold, while roles that have already been approved are in green.

By clicking "Approve", the status of the corresponding roles that the approver can approve changes to "approved". Only after obtaining approval for all roles in a particular session does the session become activated. In the case of clicking "Deny", the specified roles are denied, and the entire session is rejected, regardless of whether other approvers have approved different roles.

When a session is approved or denied for the user who requested it, a corresponding notification is sent to their email address.



If the approver will decide to deny the session request, the denial reason can be attached to the notification and sent to the requester of the session.

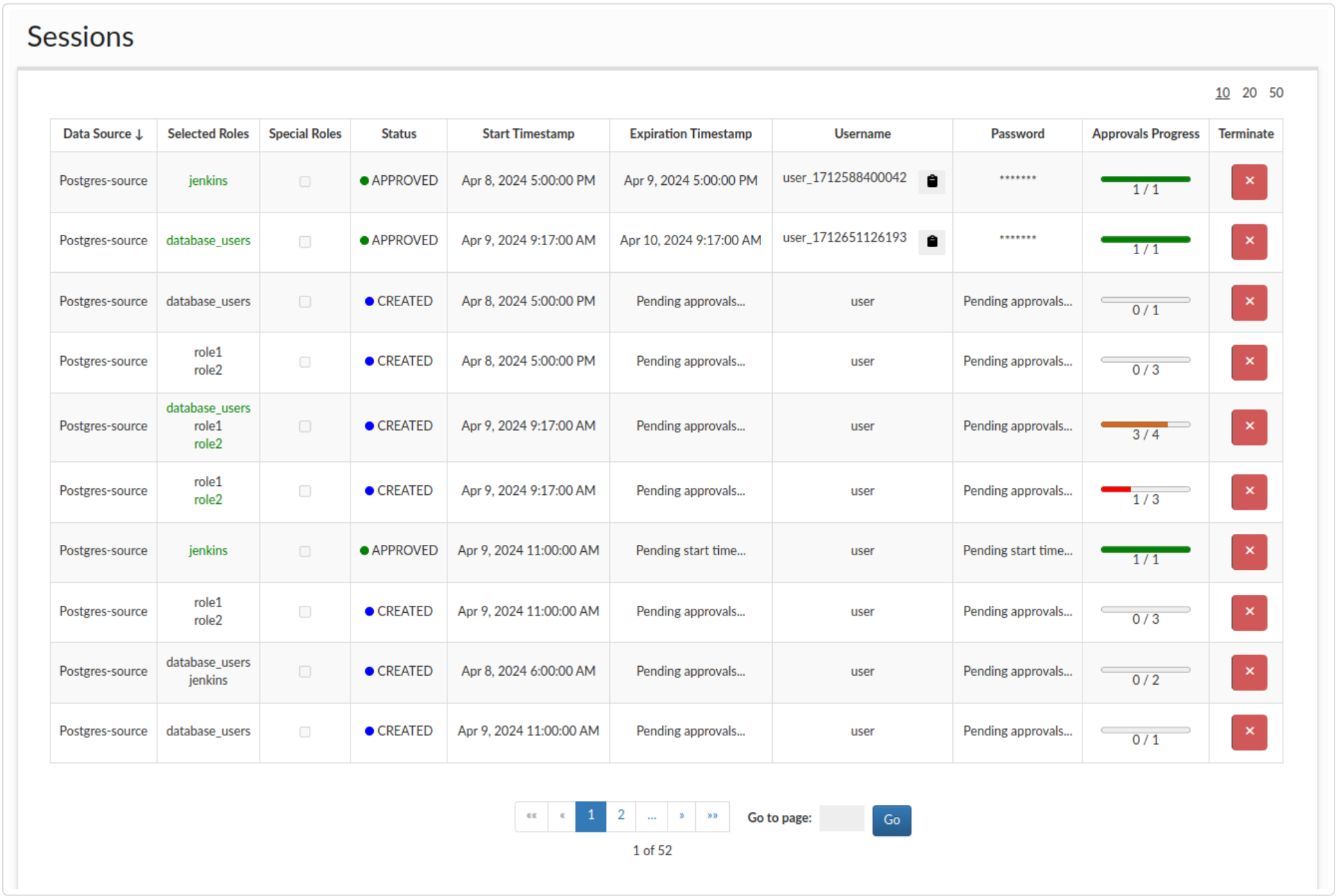


Sessions Overview

The Sessions tab provides the ability to view all sessions that have been given the "APPROVED" status, as well as those with the "CREATED" status. The tabular layout provides information for each session, including the roles requested, session expiration time, approvals' progress bar, information about which roles have already been approved (displayed in green), and user credentials for login. These credentials are only for temporary users with selected roles and are valid for the duration of the session.

The password can be viewed and copied only by the user who requested the session; the admin does not have this permission.

The Sessions tab is accessible to both admins and users who requested the session. There is also an option to terminate a session, whether it has already been approved or is still pending. Admins have the authority to terminate such sessions, while a user who requested their own session can cancel it at any time before the session expires.

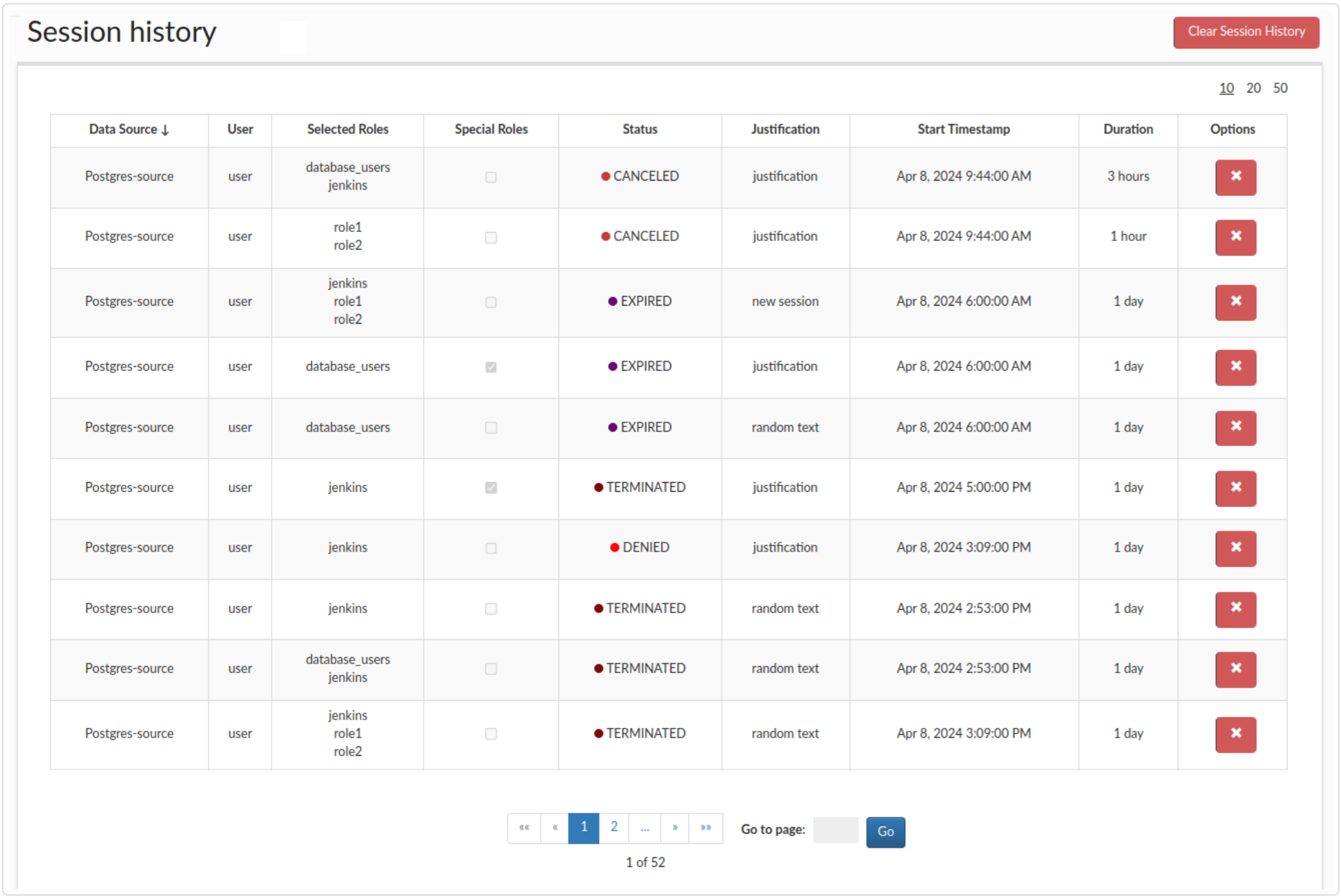


Session history overview

The Session History tab allows users to view their own sessions that are no longer available, and enables admins to access the complete history of all users' sessions. Admins can also delete session data from the database and clear the entire history. Sessions deleted in this way will be removed for all users.

Sessions visible in this tab are not associated with any database user or their credentials. Either they were never assigned, or their credentials were already removed. The "Status" column indicates the manner in which the session ended:

- **DENIED**: The session was denied by one of the approvers, as such credentials were never given.
- **CANCELED**: The session was ended due to cancellation by the requesting user.
- **TERMINATED**: An admin terminated the session.
- **EXPIRED**: The session timed out.



Logging Configuration

Logging Design

The system is designed to allow the proxies/drivers to generate a variety of records to detail the events that are occurring as traffic is processed. These can include:

- SQL statement execution
- Connections opening and closing
- JDBC Method invocation
- Exceptions
- Console/Debug logs

As many of these categories can result in a very large amount of traffic, a variety of mechanisms are provided to control the volume of logs. Here is where each is controlled:

- **Connection open and close**: Only in the VDB tab, under logging
- **JDBC Method invocation**: In the VDB tab, under Debugging
- **SQL Statement Execution**: As an all on/off toggle under the VDB tab, under Logging, OR; via Log rules, with selective throttle controls
- **Exceptions**: Always generated
- **Console Logs**: These are always generated, but where they are sent can be controlled with the "Aggregate Console Logs" option under Logging in the VDB
- **Debug Logs**: These output as console logs, but control if they are generated is under the VDB Debugging tab with the "Verbose Debug Mode" option.

When using a log rule to perform SQL statement logging, there are a variety of options that can be used to limit the rate of logs. These are documented in the rules configuration documentation under "Rate Limiting" and can provide for a maximum log entries per second, or a ratio of logs processed vs. not processed. The rate limiting occurs at the proxy/driver level, and by reducing the rate of logging, this also reduces the amount of processing on the node generating the logs, so is highly recommended in production environments to limit the impact.

Global Log Management

In order to control the total size of the logs globally, there are two options available in the admin->Server properties section:

- **Max Log Age**: This is the number of days that log files and database log entries are allowed to persist. This defaults to 14 days.
- **Reserved Disk Space**: This sets how much space must be free at all times. During log rolling, SQL logs will be deleted until this value is achieved.

When cleaning up disk space, logs are deleted first based on the Max Log Age parameter, then based on the Reserved Disk Space value.

By default, logs are written to /opt/heimdall/log, however this can be configured. By adding into the javaOpts the property "heimdall.logdir", the relative directory can be controlled, i.e. "-Dheimdall.logdir=./log2/" will set the directory to use log2 for the flat file logs.

Analytics Requirements

When analytics are to be used, we log data into an internal database on the management server, which summarizes the data in a concise format necessary for fast performance and minimal space and overhead. The minimum necessary to perform this logging is either:

- Enable the "Log All SQL" option in the VDB Logging section, or;
- Attach an enabled "Log" rule that matches the desired content to be logged.

Flat-file Logging

In order to provide detailed logging, an additional option is available in the VDB tab, to "Write logs to file", which will trigger all query logs for the VDB to not only be registered to our internal database, but to also be logged individually in a CSV formatted file in the $serverhome/log directory, and follow the following format (CSV, and can be opened in a spreadsheet if desired):

- **epoch-ms**: Log timestamp (ms since epoch)

- **type**: Event type
  - **TRACE**: Generated with trace actions
  - **DONE_SQL**: Generated with log actions
  - **CONNECTION_OPEN**: Generated if log connections set
  - **CONNECTION_CLOSE**: Generated if log connections set
  - **METHOD_COMPLETE**: Generated if log methods set
  - **TRANSACTION_START**: Generated if log methods set
  - **TRANSACTION_END**: Generated if log methods set
  - **DEBUG**: Provides additional logs generated in rare cases
  - **SQL_EXCEPTION**: Generated when SQLException is thrown
  - **INTERNAL_EXCEPTION**: Generated when any other exception is thrown
  - **NEW_PATTERN**: Triggered when observing a new query pattern
  - **NOTIFY**: For rules tagged with a notify
  - **CONSOLELOG**: When console log aggregation is enabled, this provides the log data
  - **NOCACHE**: Generated when cache revalidation fails
  - **PROXY_START**: Generated when proxy is started or restarted
  - **AUTHENTICATION**: Generated when user tries to authenticate
  - **ERROR**: Generated when something unexpected or unwanted happened, but no exception was thrown
  - **SYSTEM_STATUS**: Generated for Heap Dumps and Stack Traces
  - **METRICS_DISCOVERY**: Generated for JMX statistics propagation between central manager and proxy
  - For additional info about event types please visit Monitor->Logs Overview.
- **flags**: Status flag
  - **0**: Cache miss
  - **1**: Cache hit
  - **2**: Cache miss (in transaction)
  - **3**: Cache hit (in transaction)
- **User**: JDBC Username used to connect to the db
- **vdb**: VDB name
- **hostname**: Hostname of the source
- **instance-id**: A unique ID for each JVM, to provide uniqueness even on the same host
- **conn-id**: Session id (uniquely assigned from server to identify different users)
- **class**: Log entry source method type:
  - **C**: connection
  - **S**: statement
  - **R**: resultSet
  - **P**: prepared
  - **X**: callable
  - **D**: Driver level
  - **B**: Data Source
  - **U**: Unspecified type
- **obj-id**: Object id (semi-unique identifier for given object in this session)
- **parent-id**: Parent id (id of parent object. E.g. If ResultSet, parent id would be id of statement that created it)
- **runtime-us**: Runtime in microseconds i.e. response time of operation generally. For connection close events is total time connection was open, and for TRANSACTION_END, it is the time since the last TRANSACTION_END or the TRANSACTION_START.
- **db-name**: Data source (may be the name of the LB source)
- **Message**: Event text. For both the log (done_sql) and trace (trace) event types, all of the following fields will be inserted, but for the done_sql, many will be empty values, as the data is not tracked for this type to lower overhead. Other record types include text that represents what was done, i.e. the method, etc.
  - **rows**: number of rows
  - **rows-read**: number of rows not read
  - **result-size**: Estimate of raw byte size of Result
  - **java-size**: Estimate of Java byte size of Result (including object overhead)
  - **proctime-ms**: Overall time (all rows processing time)
  - **result-hash**: Pattern hash used by Heimdall, to associate similar queries
  - **text**: text field based on the record type--this can change between releases fairly often.

SQL statement & Parameters The SQL statement is logged in the format of “Statement~num parameters~param1~param2~”. Each parameter is replaced with a ? in the statement. The target is to simplify the parsing of query parameters, yet insure that if log records are sorted based on this portion of the message, than unique queries will be sorted together. This format also makes stripping potentially sensitive information from the records when needed.

Note: The log fields may change between major revisions of code. Each file contains a header that provides the position of each field, and should be used for parsing of the file to insure that additional or rearranged fields do not break any code depending on the format of the log file.

Example Log Entries

#Fields: epoch-ms, type, flags, user, vdb, hostname, instance-id, conn-id, class, obj-id, parent-id, runtime-us, db-name, rows, rows-read, result-size, java-size, proctime-ms, user, result-hash, text

1523556866884, TRACE, 0, magento, Magento-Demo-vdb, ip-172-31-56-200, 94f7f042-7bd5-41a5-bb49-091877788fde, 0, R, 469576, 11812, 1616, Magento-Demo-source-Master, 1,1,65,425,1740,4BC71F67001332175D27AFF533095BB2,magento,magento.test,SHOW TABLE STATUS LIKE ? ~1~\"'core_resource'\"~,

1523556866892, TRACE, 0, magento, Magento-Demo-vdb, ip-172-31-56-200, 94f7f042-7bd5-41a5-bb49-091877788fde, 0, R, 469577, 11812, 7420, Magento-Demo-source-Reader, 68,68,1925,6005,7584,EBF219E0CA457EC7C9A9255C33A25908,magento,magento.core_resource;magento.test,SELECT `core_resource`.* FROM `core_resource` ~0~,

1523556866894, TRACE, 0, magento, Magento-Demo-vdb, ip-172-31-56-200, 94f7f042-7bd5-41a5-bb49-091877788fde, 0, R, 469578, 11812, 803, Magento-Demo-source-Reader, 9,9,177,357,862,1AF8150173C7C1B330EE8434132A65FB,magento,magento.test;magento.index_process,SELECT `index_process`.`indexer_code` FROM `index_process` ~0~,

Caching Architecture

Heimdall operates via a two-layer cache system. The first layer (L1) is an in-heap cache, the size of which is controlled by the cache size in the VDB settings. The second layer (L2) may either not exist (local cache only), or will leverage any one of the supported cache engines, to provide a distributed cache between nodes.

For caching to be active, the following conditions must apply:

- The VDB defines a base cache configuration
- A rule-set is attached to the vdb
- The rule-set defines one or more cache rules (please see the rules configuration section for the list of cache options)
- the query meets some minimal qualification for caching based on built-in logic, including size limits, etc.

Caching Logic Flow

The (simplified) order of logic processing is as follows:

- Request processing
    - Parse query for tables (this can be displayed with the printTables property on a rule)
    - Rule processing:
        - if (table property) Add to the table list for the query
        - if (cache) set cache ttl to query
        - if (nocache) unset any cache TTL and flag to not cache at all
    - After processing the rule, if a cache TTL is above 0 and caching is set
        - Create a 64 bit hash of the query and all parameters
        - Perform lookup on L1 cache with the hash
        - If no L1 cache hit, then check if L2 cache has the object
        - If a result is returned from the L1 or L2 (grid) cache
            - Check if the result has a creation time later than the last write time for any associated tables, if so delete object from the cache
            - Return the result to the client
        - If no result has been returned, retrieve from the database
    - If the query is not cacheable, check if it is an update
        - If an update, then trigger an invalidation message into the L2 cache pub/sub interface for associated tables
        - After pub/sub message is transmitted, forward the query to the database
- On result processing (from the database)
    - If the query was flagged with a TTL, create a cache object
    - If in proxy mode, return result to the proxy module to serialize
    - If in JDBC mode, attach result to the cache object
    - Trigger the log result
    - After the result-set is closed at the application level
        - queue the result for an ingestion thread
        - Place the cache object (and all metadata) into the L1 cache
        - If necessary, serialize the result for L2 caching and push into the L2 cache

Caching has two fundamental modes of operation, depending on if Heimdall is in proxy mode or operating as a JDBC driver. In proxy mode, the cached object will be the serialized version of the result-set that is pushed on the wire, in order to reduce the overhead of generating a response for following cache hits. The serialized data is also stored in the L2 cache in this same form, as the result is sent to the client the first time, this reduces the overhead of interacting with all layers of the cache. This process also accounts for variables such as character set that may result in different resultset representations depending on the state of the connection to the database.

When in JDBC mode, the local L1 cache stores result-sets in Java object form on the local heap, in order to provide the results in the form the application will use them. When storing into the L2 cache, the results will be serialized into a format that is compatible with all cache engines supported, and only pulled and de-serialized when necessary.

Caching will trigger the query to generate a 64 bit hash of the query, and lookup in the cache index if there is a matching response. In the event there is, request processing is terminated, and the response is processed to the caller. On a cache miss, the processing continues, and on the response side of processing, the response will be stored in the cache store for later use, with a time to live based on the cache rule setting.

When an object is cached, the query that generated it will be hashed to a 64 bit query hash, including all parameters necessary to generate uniqueness (by default vdb, catalog, user). The object will then be pushed into the L1 and L2 cache, if present. Our cache logic is designed to do cache key synchronization as well, so any object pushed into the L2 cache will be visible to all other nodes. The benefit to this is that we will rarely if ever have a cache miss against the L2 cache. As the L2 caches are generally on remote nodes, this minimizes the overhead of caching, and prevents a double round-trip on a cache miss.

In the event that due to a previous cache configuration, an object was cached that would now no longer be cached, due to the order of operations, the old “stale” object will be thrown out after being pulled from the cache. This applies to table exclusion rules, or TTL changes. As such, unlike most systems, a change to the cache rules will not require a purge of the cache to prevent invalid data from being returned.

Each cache object is tagged with the tables the query was associated with, and internally, all tables are tracked with a last modified time, in milliseconds. On a cache lookup, if the object was created prior to the last modified time of any table it is associated with, then the object is flushed from the cache, and repopulated. This helps insure that even if cache policies are changing, the current policies ttl limit will be honored, even if the policy is created or changed after the object enters the cache. Last modified data can come from any one of several sources:

- A DML was observed on the table directly by the driver
- An invalidation record with a newer last modified time was pulled from the grid cache
- A last modified time was pulled from the database via trigger notification
- The first two of these don’t require any additional configuration. The third requires some configuration to setup and configure the database to support this. Currently, sample trigger configurations are available for MySQL and MS-SQL, and can be developed as needed for other databases. Please contact support for assistance with this feature.

Table Caching provides a refinement to any cache policy in order to override a TTL of 0, or reduce the cache TTL on a match. In the case of a cache value of TTL, table cache entries may override the base TTL if and only if all tables have a positive ttl specified. If a table cache entry specifies a ttl of zero, and the table is present in a query, the query will not be cached. Regex patterns do not apply on a table cache, nor are they ignored, even if the “ignore” action is used.

Stored Procedures, Views and Triggers

When using any query that can't be directly parsed, it is possible to still properly enable caching and invalidation. There are a few critical pieces that need to be configured however:

- When tables being written to can't be parsed, add a "table" property, then the fully qualified name of the table, i.e. magento.log_table, and set the property "update" to true, if necessary. On a write, only the table(s) being written to need to be associated with the query, as any reads will be done server-side, and will not come from the cache.
- When tables are being read that can't be parsed, like with a write, use the "table" property to specify the fully qualified table names being read from, and set the property "update" to false. This will insure that writes against the read tables will invalidate the queries being read from.

Example:

In the case of views, simply use the tables:{tablename} syntax to match the view name, then attach the tables that are associated with it. Nested tables can be handled with this as well, as long as the order is correct, i.e. a view using another view is defined before the view it uses. This way, you attach each view's tables to a growing list of tables.

Repopulation

When the heimdall proxy or driver is restarted, if connected to an L2 cache, it will request all the current keys from the L2 cache. This will be used to determine what objects (as they are accessed) can be pulled from the l2 cache layer. If not present, then repopulation will be done from the database itself. No cache data is stored on the disk of the proxy nodes by Heimdall, and all cache objects can be considered "in-flight" or transient.

Invalidation

When a DML operation is detected by Heimdall, and a cache engine is configured, it will use the pub-sub interface of the cache engine in order to exchange information on what tables were written to and when. This is shared across nodes, so that each node knows when other nodes perform a write. In addition, some additional logic is used to reduce the number of invalidation messages:

- If less than 95% of queries are flagged as reads, then a table is deemed non-cachable, and any writes to the table should also trigger a "super-invalidation" to other nodes
- If more than 100 invalidations a second are triggered, then trigger a "super-invalidation" on tables being written to as well

A super-invalidation is a message that when sent to all the other nodes triggers non-cachability of the referenced table for two seconds.

To override invalidation of a table as a result of a DML, the property "invalidate" can be set to false to prevent any eviction action. This will not prevent caching of the object.

To explicitly control if a particular query is treated as an update, the property "update" can be set to true. This will further trigger invalidations against associated tables if parsed.

One final option to control invalidation is to use the "olderthan" property, which can be set to a negative value. In this case, invalidations last into the future, so a write against a table may prevent cached results for a period of time after the write.

Configuring Trigger Based Invalidation

In order to support out-of-band data ingestion into the database, a data source can be configured for "Trigger based invalidation". There are several options available on the data source for this configuration:

- dbTriggerTracking: set to true if this feature is to be enabled
- dbTimeMs: A query to retrieve the current time in MS on the database, to insure timestamps are synchronized
- dbChanges: A query to retrieve a timestamp (as a long) and table name (as a string), for the last invalidation time of that table.

The dbTimeMs call for example defaults to the value of "SELECT heimdall.get_time_ms()", usable on MySQL without any changes. The dbChanges call defaults to {CALL heimdall.changes(@ts)}. This can be a stored procedure. The string "@ts" will be replaced automatically with the last modification (per the DB time) that has been observed, so can be used to pull only the tables that have been modified since the last invalidation observed. The table name returned needs to be fully qualified, as observed with the "printTables" option for the rules, as this is the table name we will be comparing for invalidation purposes.

Note 1: As the trigger table queries are performed as part of LB health checking, the load balancing feature must be active to use this feature Note 2: Please see the database specific documentation for examples on how to configure the trigger based invalidation on the database side

API Based Invalidation

When using the healthcheck-port option (see VDB Properties), an additional feature is enabled, that of HTTP based eviction. To use this, a call can be made on the healthcheck port such as "/invalidate?vdb=&table=". This will trigger the proxy to process an invalidation as if a write had been detected to the table. In a distributed cache scenario, this is propogated to other nodes as well. Debug logging will provide additional verification that the invalidation has occurred. The table name of "all" can be used to clear all tables associated with the VDB on a given proxy node (this is not propogated across nodes however). Table names can also be regular expressions. When a table is invalidated via regular expressions, the tablename will be returned as the response body in a CSV list. Note: If a table has never been invalidated, it will be considered to have a last write time of 0 implicitly, so won't need to be invalidated, and will not show up in the list of tables invalidated.

Healthcheck-port can also be secured with an authorization token if Enable Token Authorization is checked. A field with a generated token will appear where the token can be changed manually or randomly generated by using the “generate new token” button. If a user wants to perform an HTTP call with enabled authorization they must add a token to request parameters.

Example debug logs when using this feature:

curl "http://localhost:8081/invalidate?vdb=Vevo-demo-vdb&table=test&token=securityToken"

```
[2018-12-04 19:28:49,060] 2018-12-04 19:28:49,060 308879 [INFO] [nioEventLoopGroup-9-1] Cache: Invalidated: test vdb: Vevo-demo-vdb 
[2018-12-04 19:28:49,060] 2018-12-04 19:28:49,060 308879 [INFO] [nioEventLoopGroup-9-1] Issuing invalidation on table: test at: 1543
```

Note: Always make sure to specify the table name as it shows as a result of the printTables command or from the expanded query view in the analytics tab, otherwise the invalidation will not be successful.

Auto-Refresh

In many situations, particularly for Analytical and Dashboarding workloads, a small but repetitive set of queries are issued against a database that is only periodically refreshed. The queries against the data-set may be very expensive and time consuming, and if issued only when someone wants to view a report, may result in a long lag before the report is rendered. This feature is for this use-case.

The way auto-refresh works is that first, heimdall will cache a query, typically for an extended TTL, be it one day or longer. Next, when a data load is done, the table the invalidation is cached against. This invalidation will trigger Heimdall to inspect all the last X queries it has processed, and will re-execute the queries, in order to re-populate the cache with the fresh data. This refresh will be done from most frequently used queries to least frequently used in order to make data that is most likely to be needed available as soon as possible.

The net result of this is that immediately after a data load and invalidation, the cache is refreshed, and reports can be rendered quickly when needed without queries being processed on the database.

To enable auto-refresh, in the VDB, set the vdb property "trackQueryDistributionCount" to the number of the most recent queries to track (say 10,000) and then in your cache rules you can set the property autoRefresh. The following conditions must be met for a refresh to occur:

1. The query tracker must have at least two instances of the query being issued. One is not sufficient. If the query tracker is under-sized, then this will impact refresh capabilities.
2. The query must be in the cache at the time of the invalidation;
3. One or more tables associated with the cached query must be invalidate.

Queries will be refreshed based on the most frequently accessed query to least frequently, and refresh queries themselves will not be tracked by the query tracker.

Note: Auto-refresh is not yet functional with the SQL Server proxy

Debugging

One of the most common issues with caching is that the tables extracted from the query or otherwise assigned to a query via a rule do not match between queries and updates. There are a variety of reasons why this can be, including the use of the Postgres search path to search multiple schemas, stored procedures, or invisible updates due to triggers. The easiest way to diagnose this is to create rules that match the update query and the read query, then add the "printTables" property, with the value of true. This will result in log entries such as:

[2018-12-06 15:17:43,924] ip-172-31-14-81.ec2.internal: Query: SELECT ? FROM ir_module_module WHERE name=? AND latest_version=?~3~\"1\"~\"'base'\"~\"'11.0.1.3'\"~ tables: [odoo.public.ir_module_module] isUpdate: false

The table name(s) listed in the tables field must match exactly for invalidation to operate properly.

Additionally, when objects are configured for caching, but are not eligible for caching for some reason, in general, the debug logs will provide additional information on why this is the case. One case that comes up with SQL Server stored procedures often is that they are returning more than one result set. This case is not currently supported by Heimdall.

In order to debug the auto-refresh tracking, the command "heimdall querytracker" can be issued. A table name can be issued following this to narrow down the queries tracked against a particular table.

Load balancing & High Availability

Heimdall provides a multi-tiered approach to supporting High Availability that supports a wider variety of databases vendors and database topologies than any other product on the market. This includes failover logic at the JDBC driver itself, and coordinated failover via the central manager. With it, the following aspects of database behavior can be accounted for and made highly reliable, while accounting for the limits that may be inherent in any database synchronization technology:

Supported Modes

True active+active load balancing One or more write-only masters One or more read-only slaves Hot-standby servers Cold-standby servers Control of pre-emption on restoration of services Scripted orchestration on failures with multi-phase configuration commits Automatic disaster recovery activation *Custom query routing

How to account for each of these will be described, and a variety of configuration scenario examples explained in detail.

Application Side Requirements

Important when using Heimdall in an environment where the proxies are themselves load balanced, a maxAge or similar (maxUsage for Node.js) option should be set so that connections do not live forever. If not set, this can result in one or a few proxy nodes being overwhelmed with traffic while others are not, preventing autoscaling.

Basic Theory

While simple at a high level, high availability of database servers is an extremely complex topic, and the interaction that Heimdall has with the databases is as such also complex topic. Below are a few questions that need to be asked both in setting up the database cluster, and the answers are important in evaluating what should be done with the Heimdall configuration:

1. Should the infrastructure have a shared-nothing approach to the data processing of queries? This implies that all disks and disk infrastructure should be redundant. Solutions such as Oracle RAC leverage a shared-disk infrastructure in order to simplify the task of managing data, when load balancing of the database nodes is performed. While an excellent solution for many environments, this infrastructure is very expensive, and still leaves the risk of a storage corruption resulting in a complete outage.
2. If not sharing the disk infrastructure, will synchronous or asynchronous replication be performed between nodes?
    1. Synchronous replication indicates that between two or more nodes, all nodes must agree that a commit can be performed, and that it completed before a response is provided to the client. This results in a significant overhead in writes, but guarantees that database nodes are in sync, and allows a more flexible active-active approach to load balancing of write traffic. For the purposes of configuration, a shared disk infrastructure will be considered the same as synchronous replication.
    2. Asynchronous replication is the method of executing a complete transaction on a single server, then replicating the changes to a second server. In most cases, this works well, but can have unfortunate side-effects.
    3. Will a single write-only node be acceptable, with reads spread across nodes, or will all activity take place on a single node, with a second node acting as a pure standby node?
3. Will the replication be unidirectional or bidirectional, i.e. will all changes be pushed from one node to one or more, or will all changes on all nodes be pushed to all other nodes.

An excellent article on the issues with Asynchronous replication and load balancing can be found at: http://datacharmer.blogspot.com/2013/03/multi-master-data-conflicts-part-1.html, and covers much of what is necessary to help answer these questions and an overview of the issues related with them.

Once the question of if the server is doing synchronous replication or asynchronous replication and unidirectional vs. bidirectional replication is answered, then the Heimdall configuration can be designed.

Basic Configuration

First, for load balancing to operate, the “enable load balancing” option must be selected. If not, then the only server Heimdall will connect to will be the primary JDBC URL. If load balancing is enabled however, then the jdbc URL specified in the top section of the configuration will be ignored, although used as the initially configured server in the load balancing configuration.

The next option, that of track cluster changes, enables the cluster detection and tracking logic to become active with Heimdall. The purpose of this is to allow Heimdall to reconfigure itself based on the cluster's configuration as it may change. This works for the following cluster types:

1. MySQL (stock) Master/slave replication
2. Galera Cluster Master/Master replication
3. Postgres Master/Read-slave replication (not PG 10 logical replication)
4. SQL Server Always-on availability groups
5. SQL Server Mirroring

In an RDS environment, to support global RDS clusters, the AWS RDS ARN can be filled in, and will then be used to discover the entire global cluster.

The use response metrics option is used in situations where the readers may be distributed, either in multiple availability zones, or even globally distributed. This option will result in each proxy using the response time metrics from health checks to determine the closest or best performing reader, then filtering out any readers that are not within 25% of the best reader's response time. In most cases, this is sufficient enough so that only the readers in the same availability zone will be selected to be read from, although some combinations of AZ's may be paired together for reading due to close proximity.

The Track Replication Lag option is described below, as is the Lag window Buffer.

When enabled, Heimdall will on a failure event and every 30 seconds check the state of the database cluster, and will reconfigure itself as necessary based on the cluster configuration, allowing for automatic role selection. Please see below for the rules in determining what server takes what role (when it is not deterministic) via the target read and write capacity.

The node configurations contain the following:

- **NAME**: A simple name used internally for tracking the individual server, and may be used in various logging operations. When cluster tracking is used, the name will indicate the role and ID of the server, and will be auto-generated.
- **URL**: The URL is the JDBC URL that will be used to access this data source. Note that when load balancing is configured, this JDBC URL overrides the normal JDBC URL. In the case a database server has multiple ports, each can be used as an independent database in this configuration.
- **Enabled**: The Enabled flag specifies if the server should be used or not. If a configuration change is made from enabled to not enabled, then all active connections will be migrated away from the server as soon as transactions on each connection are completed. Enabled (even if not used) also means that the server will be monitored with internal monitoring, so will allow state change scripts to operate properly. Internal to the driver is another value similar to Enabled, called "Active". This may be observed in the logs during failures, with the state being changed automatically by the driver. This value represents if the server is considered healthy or not, i.e. it is passing health checks. This value is managed by the driver itself and can't be set.
- **Writeable**: Specifies if the server should be considered usable for queries that are flagged as writes. If a server is not writable, then if and only if a query matches a "reader eligible" rule or specially flagged forwarded request will the server be used, i.e. read/write split.
- **Weight**: Represents what share of connections each server should receive, as a random share based on the weight. If one server was configured with a weight of 1, and another as a weight of 2, then the second will get 2x as many connections as server 1. This is determined as a random weighted assignment–it does not account for the number of connections that a server already has, or the traffic load, and with a small number of connections may not be accurately represented. In the case where some servers are read-only, the share allocation will only be computed including the read-only servers if the setReadOnly() method is used on a connection. The act of using this method will force a reconnect based on the new weights, even if it was previously connected to another server.

A weight of 0 is considered a special case–it is used to designate a "server of last resort" for a data source, and allows a driver to perform an independent failover without coordinating with the central server, in configurations that allows this behavior. In the case that several servers have a weight of 0, then all servers with a weight of zero will have traffic balanced to them during the failure in a random manner.

When a server of last resort is used, it is ONLY used until at least one server with a weight greater than 0 comes back online. At that point, as connections complete transactions (or immediately if they are ide), the JDBC connections will again revert back to using the weighted server or servers, and the weights will again be used to compute the share they allocate.

Connection Hold Time: If no server of last resort is available, and all valid servers for a connection are in a non-active state, then the overall load balancing property of Connection Hold Time will take effect. This value is specified in milliseconds, and represents how long we should wait until one of two events occurs:

1. A server becomes active again;
2. The data source configuration is changed via the central manager.

When a configuration change takes place or a server state change, all waiting threads will be woken, and will then check to see if a new connection can be made. If not, they will again go to sleep until either of these events occur again until the hold-time has been passed, at which time an exception will be passed to the calling application.

Cluster Role Decision Making

The target read and write capacity is used to help select what server should be used for what role, in particular in the condition of a Galera cluster, as any node in theory can be a write node, but usually, it is desirable to only have one node act as the write node, one or more nodes as read nodes, and other nodes may be hot-standby or used for maintenance or reporting activities. The logic works as follows:

For each node, a write and read capacity will be associated. The online nodes will be ordered for write consideration based on the highest write weight, then the next, etc. The nodes will then be selected for a write role until the target write capacity is achieved. These nodes will also be used as read nodes unconditionally.

For the read roles, IF the write nodes don’t satisfy the desired read weight, then again, the read weights will be ordered from highest to lowest, and additional read-only nodes selected for reading until the target read capacity (including the write nodes) has been achieved.

If any nodes are not selected as write or read nodes, then they will be flagged as disabled, and will not be used by Heimdall until after a failure happens, at which point they will be evaluated for their role again.

The goal of this is to allow a customer to decide which servers in a cluster fill what role, based on their desires.

Database Tables for Cluster Roles

In order to provide greater flexibility in managing database clusters, for any system that supports more than two nodes (MySQL Replication, Galera, Postgres, & SQL Server), when cluster tracking is enabled, a small table will be configured on the database in the Heimdall database/schema called "servers_info" defined as:

CREATE TABLE heimdall.servers_info (private_id VARCHAR(255), public_id VARCHAR(255), read_weight NUMERIC(2), write_weight NUMERIC(2), enabled NUMERIC(1), PRIMARY KEY(private_id));

This will be created on the initial setup of the cluster by Heimdall, and provides an alternative control point to manage the values as they match in the GUI. In addition, this allows you to specify a private Id vs. Public ID, in order to provide mapping due to NAT. As Heimdall polls the cluster for the IP addresses of the nodes in the cluster, Heimdall will often receive a private IP that can't be directly connected to. By setting this private ID in the "private_id" field, and mapping it to the appropriate public IP or name, this allows the cluster logic to account for the actual topology. The read and write weights can also be adjusted, and if a node is to be brought offline, setting the enabled field to "0" will disable the node.

These settings, if adjusted, should take effect in 30 seconds, and will be reflected on the GUI once it is refreshed. Likewise, when used, a change on the GUI will be immediately reflected into the database. This provides a control point for a DB doing maintenance to take offline nodes without even logging into the Heimdall console.

Note: This table is not currently used with AWS Aurora due to the way cluster detection is done.

State Change Scripting

In order to account for the variety of different scenarios that may be necessary, Heimdall provides a generic state change scripting mechanism to allow simple scripts to orchestrate with third party tools as part of a failover. The script should be named statechange-.(.sh | .ps1 | .bat | .py | .pl) and should be located in the install directory for the Heimdall Server component. An example .bat script is provided in the default install package for the “dbdemo-mysql” data source.

The script is provided the state of the data source via the command line, i.e.

statechange-dbdemo-mysql.bat Primary:true,false,true,jdbc,1:mysql://mysql.heimdalldata.com:3307/tpch Secondary:false,true,false,0,jdbc:mysql://mysql.heimdalldata.com:3308/tpch

The values for each server entry are:

- Server name
- Enabled (true or false)
- Healthy (true if healthy, or not monitored)
- Writeable state (true or false)
- Weight (0 or positive integer)
- jdbc URL

The script needs to take these values into account and then print the desired changes to the data source configuration on the standard output of the script. The valid commands that can be issued are:

- debug
- commit
- enabled
- writeable
- weight

The debug option results in all commands being printed as received. Enabled, writeable and weight all adjust the respective options for the specified server. The commit option enacts the new configuration. The changes are made to the stored configuration as the output is returned, and made effective on a commit. Multiple commits may be made in a single script execution, and is generally recommended. First, the failed server should be disabled and a commit made. Any orchestration that will then take time should then be performed, and finally, a new server should be made active.

It is possible to leverage both the scripted failover logic, as well as the cluster change logic. In this case, the scripted failover logic will be executed first when a failover is detected, which can promote a cluster node, then the cluster change logic will execute, which can then detect the changed servers based on the cluster state. To receive notifications of health change events, please insure the notification section in the Heimdall Management Console is configured and use the test button to verify it is working properly.

Replication Lag

One of the problems when routing queries to a database cluster that uses replication between nodes is that most such configurations results in a delay between writing to a table and being able to read the results on the read-only server. As it is a common behavior to write to a table and then immediately use the new information as part of a new query, being able to track how long it takes for this replication to occur can be an important factor in implementing read/write split. With the replication lag detection in place, Heimdall will configure tables in the Heimdall database to write to on the db cluster, and then will use those tables to do performance testing to see how long it actually takes to perform replication. In addition, a static "safety" window can be configured in the "Lag Window Buffer" setting, which will be the baseline for replication lag, to which this is added. If dynamic replication lag detection is not desired, set the Lag window buffer to a sufficiently high value that should always be safe.

With the combination of either or both the "Track Replication Lag" and "Lag Window Buffer", Heimdall will then use the last write time for a table in order to determine if a particular query is safe to read from the read-only server or only from the write master. For example, if the combination of the two results in a value of 10s, then a read of a table that hasn't been read for 11s will be eligible to be sent to the read-only server, but if the table was written 5s ago, then it will be directed to the write server. This logic closely mimics (and leverages) the logic in the cache engine that determines if a particular cached object has been expired, based on the last write time to the table.

There is also an option to set an alert threshold with "Track Replication Lag" enabled. If average lag detected would be greater than this threshold an alert stating so will be generated at most once every 30 seconds. In addition, precise lag value will be logged. To disable alert threshold set its value to 0.

Read/Write Split Overview

Read/Write split is the idea of separating queries to a database cluster to different servers, which operate in different roles, i.e. having multiple nodes that are used for reading only, while a master that replicates to them as a read/write node. While simple in concept, there are many issues that need to be considered when implementing, including:

- **Replication Lag:** The amount of time it takes for a write against the primary node to replicate to a read-only node;
- **Transactional State:** In a transaction, a read may lock records for later modification, so needs to be executed against the primary server instead;
- **Stored Procedures:** A stored procedure call may write to a table, which is not visible at the raw SQL layer;
- **Triggers:** A trigger may be attached to one table to write to another table, which is then read by the application;
- **User Role:** A reporting user may need to be unconditionally routed to a read-only node;

All these factors come together when implementing read/write split in order to decide one question--should a query be routed to the read-only server at this time, or to the read/write server? The more information that goes into this decision, the safer it is to push more queries to the read-only server.

Heimdall helps make this decision easier by tracking the replication lag between nodes, and using it's knowledge of the last write time to a table (from the cache engine) in order to make an intelligent decision on which servers should use a given node to insure that stale data is not read.

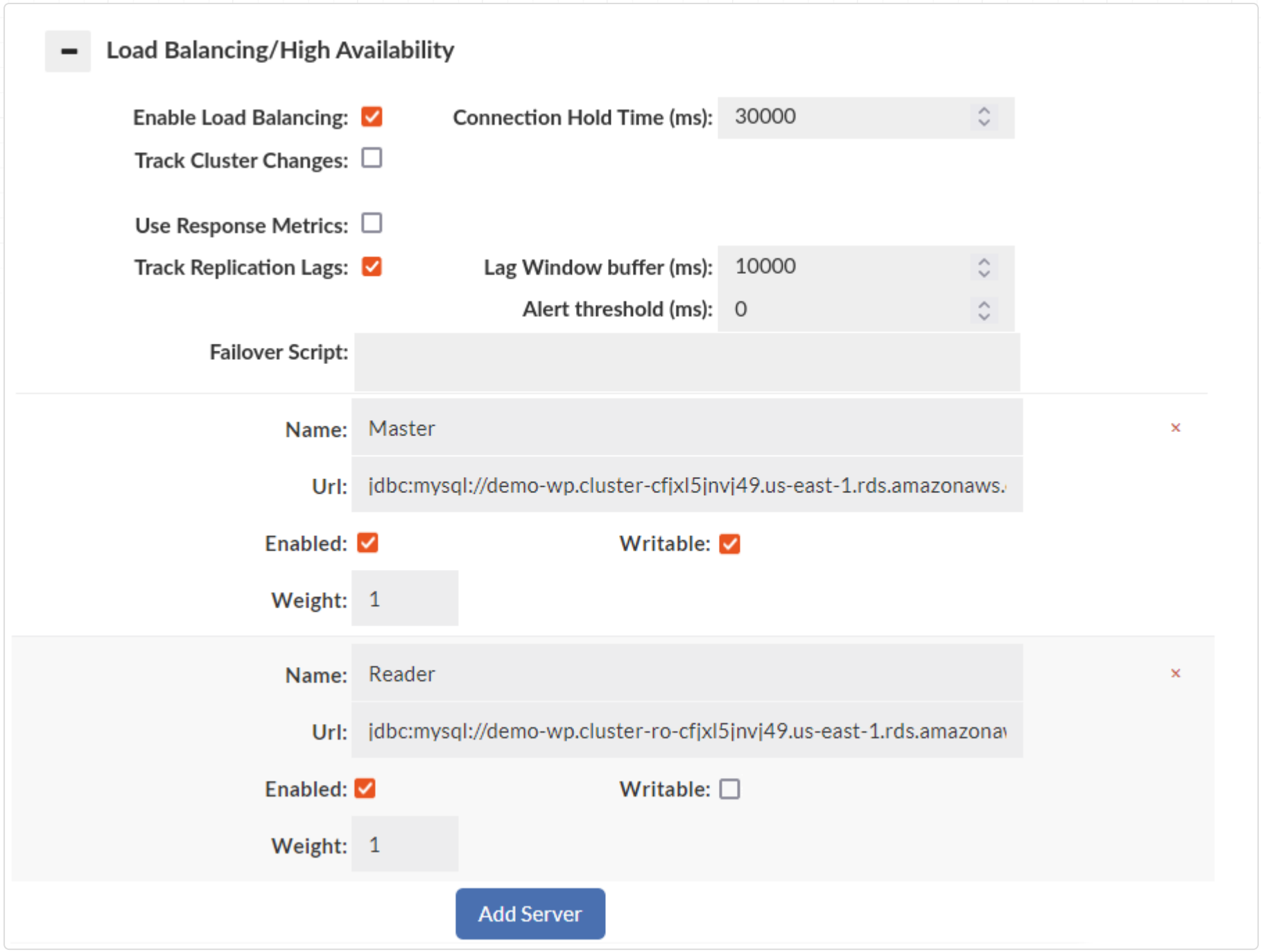
IMPORTANT For read/write split to work properly in a multi-node environment, cache invalidation events must be visible between nodes. This is typically done by setting up a cache engine, but can also be done by using the hdUsePGNotify option for Postgres servers (in source connection properties). A cache engine will need to be enabled for this to work, even with the hdUsePGNotify option.

Configuration

In order to enable read-write split the following is required:

- Load balancing needs to be configured in the data source
- In the LB section at least two sources need to be defined. One with the writeable flag, and one without.
- If replication lag is to be accounted for one of or both the "Track Replication Lag" option needs to be checked, and/or the Lag window buffer set. Both can be used, or just one.
- In the rules tab associated with the VDB, a "reader eligible" rule needs to be defined that dictates what is eligible to be sent to the read-only nodes. Example configuration and status for tracking replication lag:

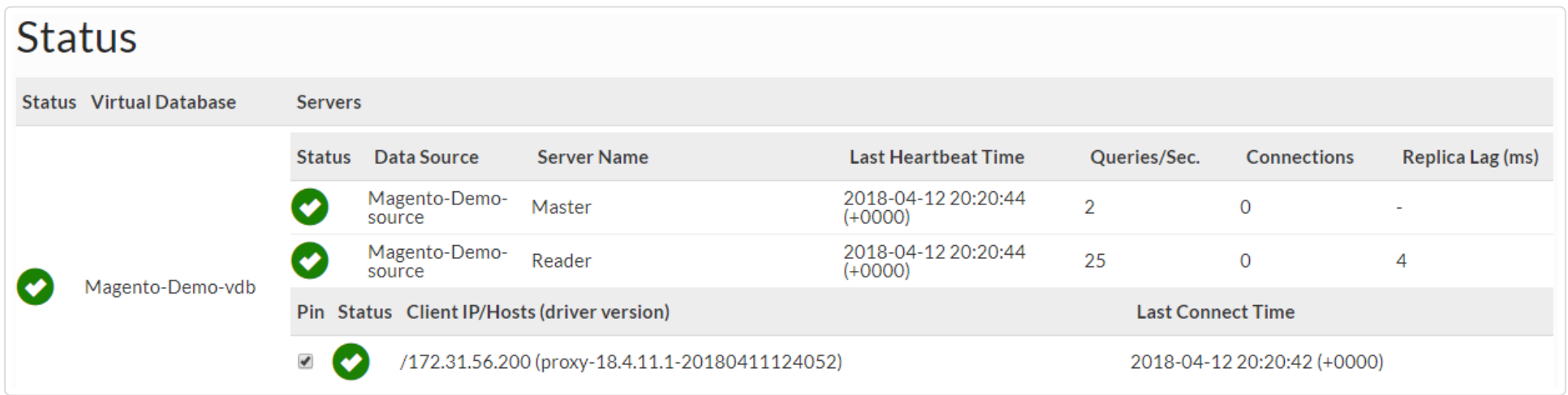
Data Source:



Rules:

Note: The lagignore option can be set to ignore lag for a particular set of queries. This is useful when a particular set of tables is not actually sensitive to lag, but may be reported as being written to often.

Status, showing the replication lag detected, and rate of queries against each role:



In the log section, the actual server selected can be observed for each query processed to validate that the correct actions are being taken, or the raw logs (if being generated) can be inspected.

Proxy Design

Heimdall's proxy is designed to operate in as lightweight a mode as possible. As such, there are a few requirements to meet:

- Disable SSL/TLS from the application to the proxy. When installed on the server as the application, this allows queries to be received and processed without the overhead of the encryption, which isn't necessary when not going over the wire. Instead, set the TLS/SSL requirement in the Heimdall data source to enforce security over the wire
- Specify the proper bind mode for the proxy to provide the maximum amount of security possible. If set to "localhost only" then only local processes can connect to the proxy, insuring that the database isn't exposed via the proxy on the server to outside users
- Insure that if not using localhost, then the user credentials are provided in the configuration, and authentication is enabled.

Asyncronous Execution/DML Batching

Many applications and environments have application nodes that end up performing large amounts of DML operations, which are performed one at a time. Most Databases are very slow at this, with analytics/MPP databases being particuarly slow. The async feature is to help improve the performance and reduce the database load when this type of activity occurs.

Basic Theory

when a DML operation is flagged with an Async execute rule, Heimdall will take the DML operation out of the current connection context, and will put it in a queue for execution on an independent connection. This queue is processed in transaction batches, which has a configurable batch size. By executing the commit across multiple DML operations, much of the overhead of the commit (the most expensive part) is shared between many DML operations, and overall DML performance improves.

There are two main categories of situations where async will help:

- One connection is inserting many DML operations in rapid order
- Connections are inserting DML operations over time from a large pool of connections

Heimdall can provide support for both, but there are situations where the Async batching should not be used.

Basic Configuration

In the Async rule logic, the following options are availabe:

- **asyncsize:** The maximum length of a query (in characters) that can be made asynchronous
- **batchSize:** The maximum batch size of queries to be executed--queries will not be delayed to fill the queue, but this sets the maxiumim size of the batch.
- **source:** The name of the data source to forward the query to for async execute. This source must be specified in the VDB definition.
- **queueName:** The per-source named async queue to process the query from. Defaults to the name of the data source the query is sent to.
- **holdUntil:** When to hold the thread until
- **spoofedResult:** The result the DML will return if the result is spoofed (only for immediate holdUntil values).

The first option, asyncsize, controls the actual size of the SQL that can be made asynchronous. In particular a DML operation may end up doing an insert of a million records as one statement. By using this size option, you can control how large the actual SQL can be for async to activate.

The second option is the size of the batches to perform on the database. This is the maximum value, not the minimum. Consider the case of this setting being set to 100, and the application is first started up. The first DML that is performed will likely be pushed to the database immediately, as only one query came in that instant. The queue has zero items at first, then one. As that one DML is executing on the server however, another four may be received from the application connections. Once the first DML is completed, the next batch will contain the next four in the queue after the first was done. Again, while those four are processing, another eight or more may come in. This process of queue buildup will continue until the batchSize is reached. At that point, even if more than that many queries are in the queue, only this many will be dispatched to the DB as a single transaction.

As a general rule the batchSize should be a reasonable number (64 is a good start) but not so high that if any one DML in the transaction fails and the transaction has to be restarted, then the burden of restarting will be too high (please review the retry logic below). Likewise, the value shouldn't be too low that the overall benefit isn't significant.

The third option is the source option. Normally, this won't be used, but this can be used in certain cluster types to route DML operations for a particular table to alternate data sources.

The queueName options is used to separate different batches of DML operations from each other. By default a single queue will exist for any given data source, based on the data source name. When an explicit queueName is specified however, the DML operations will be handled from a different queue, and will use a distinct database connection for processing. This is used in conjunction with table rules to filter DML for one table into one queue, and DML for another table into another queue. Through this, each table avoid blocking operations on each other, and can be used to improve performance of DML overall. The special queueName value of "${table}" can be used to create a queue per table, and is ideal when expecting rewrites of inserts into a multi-row insert.

Next, the holdUntil option is used to determine the actual async behavior. For each DML, it is often the case that the application is expecting the result of the DML operation, i.e. to tell it how many rows (if any) were modified. This may impact later logic in the application. In the case that the application needs this, we have two options for holdUntil, that of "result" or "commit". We can wait until the result has been sent from the DB, but no commit executed, OR we can wait for a commit to be generated. For best performance, a holdUntil value of none should be used, BUT this requires spoofing the result so that the application can continue processing immediately. When spoofing is needed, the final option of spoofedResult is used, and sets the value that will be returned.

Insert Rewrites

As part of the logic of batching, if a queue contains sequential inserts of the pattern "INSERT INTO ( ) VALUES ( )", and the table and field list are the same between multiple inserts, these inserts will be rewritten into a single insert with multiple rows. For now, ONLY inserts that specify the table and field list will be rewritten, to ensure that exactly the same data is being written in the same columns, and it will ONLY rewrite inserts that follow each other sequentially with no interruption. Only the batch size number of rows will be written at once.

Retry Logic

When a batch is created, there is always a chance that the DML will result in an exception. The way this is handled is that if an exception is generated at the time the sql is executed (but before the commit), then the sql is removed from the batch, the batch is rolled back, the queries are restored into the queue in the proper order, and a new batch is generated. A log entry will then be generated.

In the case of a holdUntil value of none, no exception in this case is returned to the application, since it already received a spoofed result. In such an environment, the logs should be maintained in case a replay is needed.

In the case of any other holdUntil value, the exception will be passed to the client.

The final case is if the exception occurs when the commit is called. This should be rare, but in the event that it happens, then it is impossible to determine which of the batch caused the issue (if it was one SQL in particular). As such, in this case, the entire batch is logged as a failure, and the connection to the DB is torn down and rebuilt, as it is safest to just start with a clean connection.

Blocking

One issue that is likely to come up with many attempts to use Async is the following sequence of events:

select from tableA where id=10; update tableA set something=newvalue where id=10; select from tableA where id=10; // expects the new value here

Without any additional logic, this would likely break the application. With the Async feature of Heimdall however, what will happen is that the update to the table will result in a lock that is associated with the table. When the final select is issued, it will block until the update commits, THEN the select will be issued against the database. This prevents this sequence from returning the incorrect value, and breaking the application. Please note: This locking is only done within a single Heimdall instance--if additional nodes are active, they will not block on the same action, as the synchoronization time to implement this would be excessive. The main objective is to insure that at least for a given connection, the order of operations are maintained.

Initializing the Async Connection

Often, the connection that the Async logic used will need to be initialized with SQL commands that make it behave differently than normal connections. As an example, with DML operations, Greenplum should in general have the MPP optimizer disabled, as it adds excessive latency that doesn't provide a benefit. In order to do this, the connection property of "initSQLAsync" can be set on the data source, with the property of the SQL desired. In the Greenplum example, it would be "set optimizer = off".

Proxy Authentication

Before allowing queries through the proxy, they must first be authenticated by the proxy (in general). There are various configuration modes for this, and what is supported depends on the type of database on the back-end.

Authentication Modes

Heimdall supports four methods:

- None Authenticated (all)
- Proxy Authenticated (all)
- SQL Authenticated (all)
- Passthrough (Not MySQL)
- LDAP Authentication (Not MySQL)

In Proxy Authentication, the username and password to be used is configured directly on the Heimdall management server. The credentials must match what is also configured on the database. All proxies support this method.

Next, SQL Authenticated leverages a table on the database itself that stores details in a format roughly matching the format of the Postgres pg_hba.conf file, although this format supports all supported proxies. See the details below on the creation and use of this table. SQL Authentication also supports setting various attributes for the users on a per-user basis in a way not supported by other methods, and can layer with passthrough authentication as well.

In passthrough, the client is requested to send the credentials in a way so that they can be passed through to the database server, at which point the DB performs the initial authentication. This allows single point of configuration for the password, but is not supported by MySQL.

In none authentication mode, the client is always authenticated no matter what credentials he will send. It's inheriting credentials from default datasource and then authenticate user. It can be used for test purposes, without changing the connection string on the client side.

Finally, for Postgres and SQL Server, LDAP (Active Directory) authentication is supported, which includes active directory group extraction.

Database Authentication Synchronization

"Synchronize DB Authentication" button - specifies if synchronization of users and groups should be performed. Moreover, it allows configuring how often the synchronization is allowed to perform after reconnection (Sync Interval) and how long that data will stay in Authentication Cache (Auth Cache Expiration Time).

When this feature is enabled, it allows the credentials that Heimdall has accepted to be synchronized into the database itself--there is no point to authenticate against the proxy unless the database ALSO allows the user to authenticate. In some cases, Heimdall and the DB can both authenticate against the same source. In other cases, the proxy must be able to synchronize the credentials into the database at runtime. For this to operate, credentials for a privileged user must be provided, along with a command that will allow the users to be synchronized.

Example for Postgres and Postgres derived databases such as Greenplum (excluding Redshift):

synchronization query:

```
select heimdall.sync_user('${user}','${password}','${ldapgroups}');
```

The PL/pgSQL source of the "heimdall.sync_user" is available here.

Example for Redshift:

synchronization query:

```
CALL public.sync_user('${user}','${password}','${ldapgroups}');
```

The PL/pgSQL source (adjusted for Redshift) of the "public.sync_user" is available here.

Example for SQL Server:

synchronization query:

```
EXECUTE dbo.sync_user N'${user}', N'${password}', N'${ldapgroups}'
```

The example of Transact-SQL source of the "dbo.sync_user" is available here.

The source code of these functions/procedures is also stored in config/sync_scripts in heimdall installation directory and can be modified if needed. If any of them are missing, the manager will put default versions of them on startup.

These functions will first ensure that the user's role is created. If it already exists, it will drop other role memberships and reset the password. Next, it will add all existing roles back to the user's role.

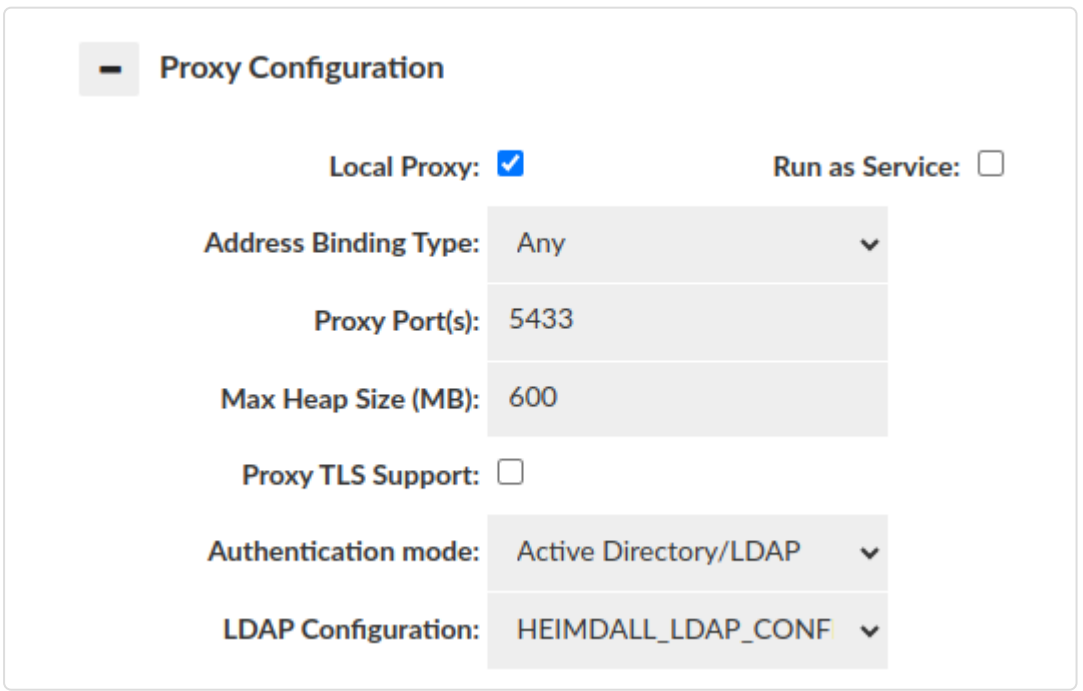
Important: If synchronization query contain the information about '${ldapgroups}' then for LDAP Authentication and LDAP Authentication inside SQL Driven Authentication it will be required to extract at least one group to authenticate the user.

LDAP Authentication

By default, each proxy use the LDAP Configuration of central manager created in Admin tab.

- Bind+Search Mode
- Simple Mode

If there's only one LDAP Configuration, it will be automatically used by default for proxy authentication. However, if multiple LDAP Configurations are established, ensure to designate the appropriate one for each specific VDB within the Proxy Configuration section found in the Virtual DB tab.



SQL Driven Authentication

Simple configuration

The simplest way to configure an authentication by SQL Driven Authentication is to turn on mentioned option in "Proxy Configuration" section of Virtual DB Configuration and provide query as "Authorization Query"

Example authorization query can be declared as below.

```
select password from heimdall.pg_hba where user_name = '${user}';
```

Used authorization query expects declared table like below example in database.

```
CREATE SCHEMA IF NOT EXISTS heimdall;
REVOKE ALL ON SCHEMA heimdall FROM public;

CREATE TABLE heimdall.pg_hba (
    user_name text NULL, -- username (exact)
    "password" varchar NULL -- user's password
);
```

Provided example will provide authentication of given user by password returned as result from Authorization Query.

SQL Query Variables

The authentication query supports the following variables, which are filled in before querying the database:

- ${user}: The username of the connecting user

- ${source}: The client IP of the connecting user, as seen at the socket level (i.e. NAT translation may hide this).

If additional variables are desired, please contact Heimdall support to submit a feature request.

SQL Authentication Caching

In SQL driven authentication credentials retrieved for a given users are cached for one minute.
There is a possibility to clear those credentials caches for a specific proxy. You have to go to into status tab, expand collapse menu button (burger button) and chose "Clear credentials properties" option. More information might be found under this link.

Advanced configuration, with connection pool properties control

To start with advanced configuration, first should be realized that whole process of authentication depends on data table containing rows with information about rules for declared situations of authorization. On side of proxy the only thing which can be configured is query which calls for result from database. That means that whole authentication process can be elastic and is only limited by database authentication table content and by authentication query.

SQL Driven Authentication provides 4 different authentication options, which can be declared in authentication table in column "auth_method".

Authorization option	Option keyword	Description
Server Authentication	scram-sha-256, trust, pam, bsd	Pass the authentication through to the server, and only allow if the server accepts. This is the default if no auth_method is provided or the value is null, or an unknown value is returned
Password authentication option	md5, password	Authentication is performed by comparing password provided by user during authorization and "password" column read from result row matching user.
Reject option	reject	Unconditionally rejects connection made to proxy by user.
LDAP authentication option	ldap	LDAP authentication option enables authentication by using ldap server. To use this authentication, required is to provide information about ldap server inside "options" column of result.

Note: The MySQL proxy only supports the password authentication when SQL authentication is used.

SQL Driven Authentication is based on values inside columns of result set. There are 4 types of columns: filtering columns, authentication columns, connection columns and override columns.

Filtering columns

Filtering columns specify if given row of result matches user connecting to proxy. These include:

Filter column label	Type	Description
connection	string	Value in this column specifies what type of connection was used by a user to connect to a proxy. If column doesn't exist in result or value is NULL or value isn't a specified keyword, then authentication rule applies to all situations.
user_name	array value	Value in this column specifies to which user name rule should be applied. If column doesn't exist in result or value is NULL or value is {all}, then authentication rule applies to all users.
database	array value	Value in this column specifies to which database connection is made. If column doesn't exist in result or value is NULL or value is {all}, then authentication rule applies to all users.
address	string	Value in this column specifies from which IP address user is connecting to proxy. If "netmask" column's value exist, then can be considered as range of IP addresses. If column doesn't exist in result or value is NULL or value is {all}, then authentication rule applies to all users. Works with IP address of IPv4 and IPv6.
netmask	string	Value in this column is only considered if value exist in column "address" of result. Value should be netmask address i.e. "255.255.255.0" or "ffff:ffff:ffff:ffff:ffff:0000:0000:0000". When specified, "address" column specifies range of IP addresses inside of specified net. If not specified, then considered as netmask length 32.

Authentication columns

Authentication columns specifies options used during authentication. There are 3 columns, which are used as authentication columns.

Authentication column label	Type	Description
auth_method	string	Value in this column declares what authentication method should be used. If column doesn't exist in result or is NULL, then default authentication option is password authentication.
password	string	Value in this column contains password which is used in password authentication inside proxy.
options	special formatted string	Values in this columns are used for providing extra options for some authentication methods. Expected value in this column in string containing pairs key="value", where pairs are separated by spaces e.g. 'key1="value1" key2="value2"'. For now, only authentication method using this is LDAP Authentication Method, which is using defined keywords to extract proper values (information about ldap keywords can be found in "LDAP Authentication inside SQL Driven Authentication" section).

Connection columns

Connection columns specify optional properties which will be set after successful authentication. There are 3 columns, which are used as connection columns:

Connection column label	Type	Description
pool:multiplex	bool	Set to override default multiplex setting for user, rules can further override.
pool:maxUserIdle	integer	Set to override default max idle setting for user, rules can further override.
pool:maxUserActive	integer	Set to override default max active setting for user, rules can further override.

In this case, fields are being defined that match to the properties that the "pool" rule types inject into a query's metadata. This allows a user's multiplex, maxUserIdle, and MaxUserActive settings to be set at a default level on login without even matching a rule. Other rule behaviors can be triggered in the same way. To see the exact property that a rule injects, you would use the "debug" logging on the vdb, which will print as part of the logging the exact property name and the value injected.

Note: As these columns match to rule properties, any rule property can actually be set, these are simply examples that are connection oriented and help set the proper behavior for the connection. Use verbose debugging to inspect properties that are used after a rule match, and any such properties can be used as a column header for advanced control.

Override columns

Override columns specify values that should override connection properties after successful authentication on proxy side. There are 3 columns which are used as override columns:

Override column label	Type	Description
db_username	string	Set to override username that is used for connection after successful authentication on proxy side. If no such column exists or value is empty/null then user_name column is used.
db_password	string	Set to override password that is used for connection after successful authentication on proxy side. If no such column exists or value is empty/null then password column is used.
db_database	string	Set to override database name that is used for connection after successful authentication on proxy side. This would mean that we can redirect connection to different database than one supplied in database column. If no such column exists or value/null is empty then database column is used.

These columns can be used to create connection with different values than these used for proxy authentication. For example, we can redirect connection to other database/schema or map user-friendly name to say “000001” as the user on the DB side.

Note: Some PostgreSQL clients will display database column value instead of db_database, event though they are connected to correct one, because they get this information directly from user input and don't query to find which database got connected.

Example usage of advanced configuration

Advanced configuration can be used in below way. On start set authentication query as below in VDB configuration.

```
select * from heimdall.pg_hba where enabled = true order by line_number asc
```

Next declare table used to authentication as below:

```
CREATE SCHEMA IF NOT EXISTS heimdall;
REVOKE ALL ON SCHEMA heimdall FROM public;

CREATE SEQUENCE IF NOT EXISTS pg_hba_seq INCREMENT BY 5 START WITH 10;
CREATE TABLE IF NOT EXISTS heimdall.pg_hba (
    line_number int4 NULL,
    enabled bool NOT NULL DEFAULT true, -- controls if the row is active
    connection text NULL, -- all, host, local, hostssl, or hostnossl
    address text NULL, -- IPv4 or IPv6 subnet or address
    netmask text NULL, -- Netmask for address, defaults to /32
    "database" _text NULL, -- array value, use {any} or null for all databases, or provide the names of the database names (exact ma
    user_name _text NULL, -- array value, use {any} or null for all users, or provide the names of the users (exact match).
    "password" varchar NULL, -- The password to use for the password type, must be provided
    ldapgroups varchar NULL, -- csv group names, as if pulled from ldap for rule processing
    auth_method text NULL, -- trust, reject, ldap or password, null=password
    "options" _text NULL, -- ldap options
    "pool:multiplex" bool NULL, -- set to override default multiplex setting for user, rules can further override
    "pool:maxUserIdle" int4 NULL, -- set to override default max idle setting for user, rules can further override
    "pool:maxUserActive" int4 NULL, -- set to override default max active setting for user, rules can further override
    db_username text NULL, -- set to override username that is used for connection after successful authentication on proxy side
    db_password text NULL, -- set to override password that is used for connection after successful authentication on proxy side
    db_database text NULL -- set to override database name that is used for connection after successful authentication on proxy side
);
```

Next (optional) you can add column comments:

```
COMMENT ON COLUMN heimdall.pg_hba.enabled IS 'controls if the row is active';
COMMENT ON COLUMN heimdall.pg_hba.connection IS 'all, host, local, hostssl, or hostnossl';
COMMENT ON COLUMN heimdall.pg_hba.address IS 'IPv4 or IPv6 subnet or address';
COMMENT ON COLUMN heimdall.pg_hba.netmask IS 'Netmask for address, defaults to /32';
COMMENT ON COLUMN heimdall.pg_hba."database" IS 'array value, use {any} or null for all databases, or provide the names of the databa
COMMENT ON COLUMN heimdall.pg_hba.user_name IS 'array value, use {any} or null for all users, or provide the names of the users (exa
COMMENT ON COLUMN heimdall.pg_hba."password" IS 'The password to use for the password type, must be provided';
COMMENT ON COLUMN heimdall.pg_hba.ldapgroups IS 'csv group names, as if pulled from ldap for rule processing';
COMMENT ON COLUMN heimdall.pg_hba.auth_method IS 'trust, reject, ldap or password, null=password';
COMMENT ON COLUMN heimdall.pg_hba."options" IS 'ldap options';
COMMENT ON COLUMN heimdall.pg_hba."pool:multiplex" IS 'set to override default multiplex setting for user, rules can further overrid
COMMENT ON COLUMN heimdall.pg_hba."pool:maxUserIdle" IS 'set to override default max idle setting for user, rules can further overri
COMMENT ON COLUMN heimdall.pg_hba."pool:maxUserActive" IS 'set to override default max active setting for user, rules can further ov
COMMENT ON COLUMN heimdall.pg_hba.db_username IS 'set to override username that is used for connection after successful authenticati
COMMENT ON COLUMN heimdall.pg_hba.db_password IS 'set to override password that is used for connection after successful authenticati
COMMENT ON COLUMN heimdall.pg_hba.db_database IS 'set to override database name that is used for connection after successful authent
```

Note: the way this is being done, the row numbers do not need to be unique, but will be used to order precedence for authentication.

Helper command to import into public.pg_hba (defined above) from the Postgres pg_hba_files_rules (PG 10+):

```
insert into heimdall.pg_hba (select nextval('pg_hba_seq'), true, "type", address, netmask, "database", user_name, null, null, auth_m
```

If you want to add a new entry, you can do so as follows (including enabling multiplexing for this user). Note, the user field is an array, in the event multiple accounts should use the same configuration:

```
insert into heimdall.pg_hba values (nextval('pg_hba_seq'), true, 'host', null, null, null, array ['user1'], 'newpassword', array ['g
```

Filter columns keywords

Filter columns have specified keywords, which can be used to change behavior of rule declared in the given row of result set.

Connection column keywords

Inside connection column can be user below keywords to change use of set rule.

Keyword	Description
all, host, local	Rule declared in this row will be used for all situations.
hostssl	Rule declared in this row will be used only when connection is made to a proxy with using SSL encryption.
hostnossl	Rule declared in this row will be used only when connection is made to a proxy without using SSL encryption.

User_name column keywords

Inside user_name column can be user below keywords to change use of set rule.

Keyword	Description
all	Rule declared in this row will be used for all users.
`+<role_name>`	Rule declared in this row will be used for users that are directly or indirectly members of given role

Database column keywords

Inside database column can be user below keywords to change use of set rule.

Keyword	Description
all	Rule declared in this row will be used for all database connections.
sameuser	Rule declared in this row will be used for database connections, which requested database name is the same as requested user name.
samerole/ samegroup	Rule declared in this row will be used for database connenctions, which requested user are directly or indirectly member of role with name same as database name.

Address column keywords

Inside address column can be user below keywords to change use of set rule.

Keyword	Description
all	Rule declared in this row will be used for all addresses.
samehost	Rule declared in this row will be used for addresses that matches the server's IP addresses.
samenet	Rule declared in this row will be used for addresses that matches any subnet, which server is directly connected to.
`<host_name>`	Rule declared in this row will be used for addresses matching given host name. Check of this keyword is made in two steps: first is made reverse name resolution of client's IP address and checked if is the same as given host name. In secodn step is performed forward name resolution of given host name and comparison of IP addresses is done. If given host name starts with a dot (i.e. ".server.com") then only reverse name resolution of client's IP address and comparison is done (i.e. for set host name ".server.com", if client IP returns as host name "heimdalldata.server.com", then it matches, if client's host name is "server.com", then comparison returns false).

Override columns example

Let's say we have defined authorization query like this.

```
select * from heimdall.pg_hba
```

And filled our authentication table with following values:

user_name	password	database	db_username	db_password	db_database
user1	password1	db1	root	root	master
user2	password2	db2	`<empty or null>`	`<empty or null>`	`<empty or null>`

For given credentials like:

```
user: user1
password: password1
database: db1
```

After filtering results of query with filter columns we take the first row resulting in successful authentication on proxy side and then the connection to native database will be established with these properties:

```
user: root
password: root
database: master
```

For given credentials like:

```
user: user2
password: password2
database: db2
```

Matched row here will be the second one but credentials won't be overridden because override columns are empty or null.

LDAP Authentication inside SQL Driven Authentication

SQL Driven Authentication enable authentication by using LDAP server. To use it properly two options should be given:

- value in column `auth_method` should be set as `ldap`;

- value in column `options` should contain information about LDAP defined by proper keywords.

General keywords

Keyword	Description
ldapserver	Value assigned to this keyword should be names or IP addresses of LDAP servers, which should be used for authentication. Multiple servers can be specified by separating them with spaces e.g. 'ldapserver="server1.com server2.com"'. If multiple servers are specified then only one of them have to authenticate user.
ldapport	Value assigned to this keyword should be port number on LDAP server, which should be used to make connection. If keyword isn't specified then port 389 is used. E.g. 'ldapport=636'.
ldapscheme	Value assigned to this keyword can define if TLS should be used during connection to LDAP server. TLS can be set by setting value "ldaps" e.g. 'ldapscheme="ldaps"'.
ldaptls	Value assigned to this keyword can define if TLS should be used during connection to LDAP server. TLS can be set by setting value "1" e.g. 'ldaptls="1"'.

Simple Bind Mode keywords

Keyword	Description
ldapprefix	Value assigned to this keyword is string to prepend to the user name when forming the DN to being used for bind in Simple Bind Mode e.g. 'ldapprefix="CN="'.
ldapsuffix	Value assigned to this keyword is string to append to the user name when forming the DN to being used for bind in Simple Bind Mode e.g. 'ldapsuffix=",CN=Users, DC=example, DC=com"'.

Search+Bind Mode keywords

Keyword	Description
ldapbasedn	Value assigned to this keyword should be root DN to start searching for user during Search+Bind Mode e.g. 'ldapbasedn="CN=Users,DC=example,DC=com".'
ldapbinddn	Value assigned to this keyword should specify user to bind to directory to perform the search during Search+Bind Mode e.g. 'ldapbinddn="admin"' or 'ldapbinddn="CN=admin,CN=Users,DC=example,DC=com"'.
ldapsecbinddn	Value assigned to this keyword should specify user to bind to directory to perform the search if the search with ldapbinddn fails.
ldapsearchdn	Value assigned to this keyword should specify the domain in which authenticated user's groups will be searched.
ldapbindpasswd	Value assigned to this keyword should specify password for ldapbinddn user to bind to directory to perform the search during Search+Bind Mode e.g. 'ldapbindpasswd="password"'.
ldapsecbindpasswd	Value assigned to this keyword should specify password for ldapsecbinddn user to bind to directory to perform the search during Seach+Bind Mode.
ldapsearchattribute	Value assigned to this keyword should be attribute to match against user name in the search during Search+Bind mode e.g. 'ldapsearchattribute="sAMAccountName"'.
ldapgroupsearchattribute	Value assigned to this keyword should be an attribute to match against group identifying attribute value, in the Search+Bind mode e.g. 'ldapgroupsearchattribute="cn"'. Setting this keyword makes it required to extract at least one group to be authenticated.
ldapusenestedgroupsfilter	Value assigned to this keyword should describe if nested groups filter should be used during a search for names of groups for an user. Accepted values are "true" and "false". If this value isn't set, by default it is set to "false".
ldapsearchfilter	Value assigned to this keyword should be search filter used instead "ldapsearchattribute". All occurrences of "$username" will be replaced by user name. E.g. 'ldapsearchfilter="(sAMAccountName=$username)"'.
ldapurl	Value assigned to this keyword allows to declare some of other LDAP options in more compact form. For information about proper using of this keyword, check LdapUrl Keyword section.
ldapservertype	Value assigned to this keyword must be one of the following: ACTIVE_DIRECTORY, REDHAT_IDM, OPENLDAP, JUMPCLOUD, OKTA, OTHER.

LdapUrl Keyword

Keyword "ldapurl" is used to declare multiple LDAP options in single declaration. However, it requires special value format to declare options properly. Format is:

```
ldap[s]://host[:port]/basedn[?[attribute][?[scope][?[filter]]]]
```

The possible options specified in the format are:

- ldap/ldaps - have to be `ldap` or `ldaps`, set `ldap` to make connection to LDAP server without TLS, `ldaps` for connection with TLS;
- host - required value, works similar to `ldapserver`, but only one server can be declared
- port - optional value, works as `ldapport`
- basedn - required value, works as `ldapbasedn`
- attribute - required if `filter` isn't declared, works as `ldapsearchattribute`
- scope - optional value, declare to specify scope used during searching for user, possible values are: `base` for baseObject scope, `one` for singleLevel scope, `sub` for wholeSubtree scope. If not defined, then default is `base` scope.
- filter - required if `attribute` isn't declared, works as `ldapsearchfilter`.

Examples

Let's assume that we want to Simple Bind authenticate to LDAP server with below options: * LDAP server url - `ldap://server.example.com:389` * LDAP prefix - `CN=` * LDAP suffix - `,CN=Users, DC=example, DC=com`

Then value in `options` column should look like:

```
ldapserver="server.example.com" ldapprefix="CN=" ldapsuffix=",CN=Users, DC=example, DC=com"
```

Let's assume that we want to Search+Bind authenticate to LDAP server with below options:

- LDAP server url - `ldap://server.example.com:456`
- LDAP base DN - `CN=Users,DC=example,DC=com`
- LDAP search attribute - `sAMAccountName`
- LDAP bind DN - `binduser`
- LDAP bind password - `examplepassword`

A value in `options` column should look like:

```
ldapserver="server.example.com" ldapport=456 ldapbasedn="CN=Users,DC=example,DC=com",ldapsearchattribute="sAMAccountName" ldapbinddn=
```

If we would want to specify LDAP bind DN as `CN=binduser,CN=Users,DC=example,DC=com` then value in `options` columns could look like:

```
ldapserver="server.example.com" ldapport=456 ldapbasedn="CN=Users,DC=example,DC=com",ldapsearchattribute="sAMAccountName" ldapbinddn=
```

If we would want to use `ldapurl` keyword instead, then value in `options` column could look like:

```
ldapurl="ldap://server.example.com:456/CN=Users,DC=example,DC=com?sAMAccountName?sub" ldapbinddn="CN=binduser,CN=Users,DC=example,DC=
```

Let's assume that we want to Search+Bind authenticate to LDAP server with below options: * LDAP server url - `ldaps://server.example.com:678` * LDAP base DN - `CN=Users,DC=example,DC=com` * LDAP search filter - `(|(sAMAccountName=$username)(uid=$username))` * LDAP bind DN - `binduser` * LDAP bind password - `examplepassword`

A value in `options` column should look like:

```
ldapserver="server.example.com" ldapport=678 ldapscheme="ldaps" ldapbasedn="CN=Users,DC=example,DC=com" ldapsearchfilter="(|(sAMAcco
```

or:

```
ldapserver="server.example.com" ldapport=678 ldaptls=1 ldapbasedn="CN=Users,DC=example,DC=com" ldapsearchfilter="(|(sAMAccountName=$
```

If we would want to use `ldapurl` keyword instead, then value in `options` column could look like:

```
ldapurl="ldaps://server.example.com:678/CN=Users,DC=example,DC=com??sub?(|(sAMAccountName=$username)(uid=$username))" ldapbinddn="bi
```

If we would want to change or scope to `base`, then value in `options` column should look like:

```
ldapurl="ldaps://server.example.com:678/CN=Users,DC=example,DC=com??base?(|(sAMAccountName=$username)(uid=$username))" ldapbinddn="b
```

or:

```
ldapurl="ldaps://server.example.com:678/CN=Users,DC=example,DC=com???(|(sAMAccountName=$username)(uid=$username))" ldapbinddn="bindu
```

If we would want to change or scope to `one`, then value in `options` column should look like:

```
ldapurl="ldaps://server.example.com:678/CN=Users,DC=example,DC=com??one?(|(sAMAccountName=$username)(uid=$username))" ldapbinddn="bi
```

Extra information about advanced configuration

There are some information which may be helpful before preparing authentication table and authentication query:

- if there is more than one row which matching to user properties, then first is taken from result set (that's why sorting of result may change authentication process);
- if there is no row matching to user properties, then authentication is rejected;
- there are three characters which can cause a problem if they are in value inside array: '{' and '}' as they are removed; '"' as quotes are used to protect commas in value (required for providing ldap options);
- if you don't want to use or can't use array type of column then can be used text column, but value inside should match pattern of array value i.e. {"user1", "user2"};
- if using text column instead of array column, recommended is to use array pattern like '{"user1","user2"}', but possible is to use without curly brackets like '"user1","user2"' and even without quotes like 'user1, user2';
- keywords inside array value columns can be joined i.e. in column `user_name` can be declared array value like `'{"user1", "+role1"}'`, what means, that this rule will be used for requested user `user1` or requested users who are members of role `role1`.

Kerberos/GSSAPI Authentication

Currently, Heimdall supports Kerberos authentication for PostgreSQL and SQLServer.

Once we set the authentication mode to Kerberos/GSSAPI, we will only need to fill in one field, which is the Keytab location. We will use that Keytab to identify as a service in the Kerberos realm. The service principal is as follows:

```
For PostgreSQL -> postgres/hostname
For SQLServer -> MSSQLSvc/hostname & MSSQLSvc/hostname:port
```

Heimdall proxy will automatically retrieve the hostname from the system. Please ensure that the "hostname" command returns the fully qualified domain name of the host.

In case of using multiple proxies, the Keytab location will be common for all proxies.

Instructions on how to add a Heimdall proxy instance to Active Directory:

One way to do it is:

1. Add our instance to the DNS server.
2. Join the instance to Active Directory.
3. Instructions for Amazon Linux: https://docs.aws.amazon.com/directoryservice/latest/admin-guide/join_linux_instance.html
4. Set the service principal for the instance.
5. Go to Active Directory Users and Computers.
6. Find your instance in the Computers folder.
7. Right-click on it and go to properties.
8. Go to the Attribute Editor and find the attribute servicePrincipalName, and edit it.
9. Add 2 values: e.g.
    - MSSQLSvc/heimdall-sqlserver-kerberos.ad.heimdalldata.com:1433
    - MSSQLSvc/heimdall-sqlserver-kerberos.ad.heimdalldata.com
10. Apply the changes.
11. In order to update the keytab file, we need to leave our instance from the realm and join it again via "realm leave domainName" and "realm join domainName"
12. After rejoining our instance into the realm, we should have our MSSQLSvc service in our keytab file. We can verify it using the command "klist -k keytab path". By default, the keytab is placed in /etc/krb5.keytab.
13. After that, we can authenticate using Kerberos or Windows authentication.

You can also enable synchronization if desired.

Load Balancing

Windows Active Directory

To demonstrate the correct setup for load balancing in an Active Directory (AD) environment with Kerberos authentication, let's use the following scenario as an example.

In this setup, we'll be operating with two proxy servers. The first proxy is hosted on **hd-lb-proxy3.ad.heimdalldata.com**, and the second on **hd-lb-proxy4.ad.heimdalldata.com**. These proxies are managed by an active load balancer designed to direct connection attempts to **hd-lb-win.ad.heimdalldata.com** towards one of these proxies, based on the current load balancing algorithm.

Configuring this environment to support Kerberos authentication involves a sequence of five essential steps:

1. Creation of an AD User for Proxy Service
2. Assignment of a Service Principal Name (SPN) to the New User
3. Generation of a Keytab for the SPN
4. Distribution of the Keytab Across Hosts
5. Configuration of SPN and Keytab in Heimdall

IMPORTANT !!!

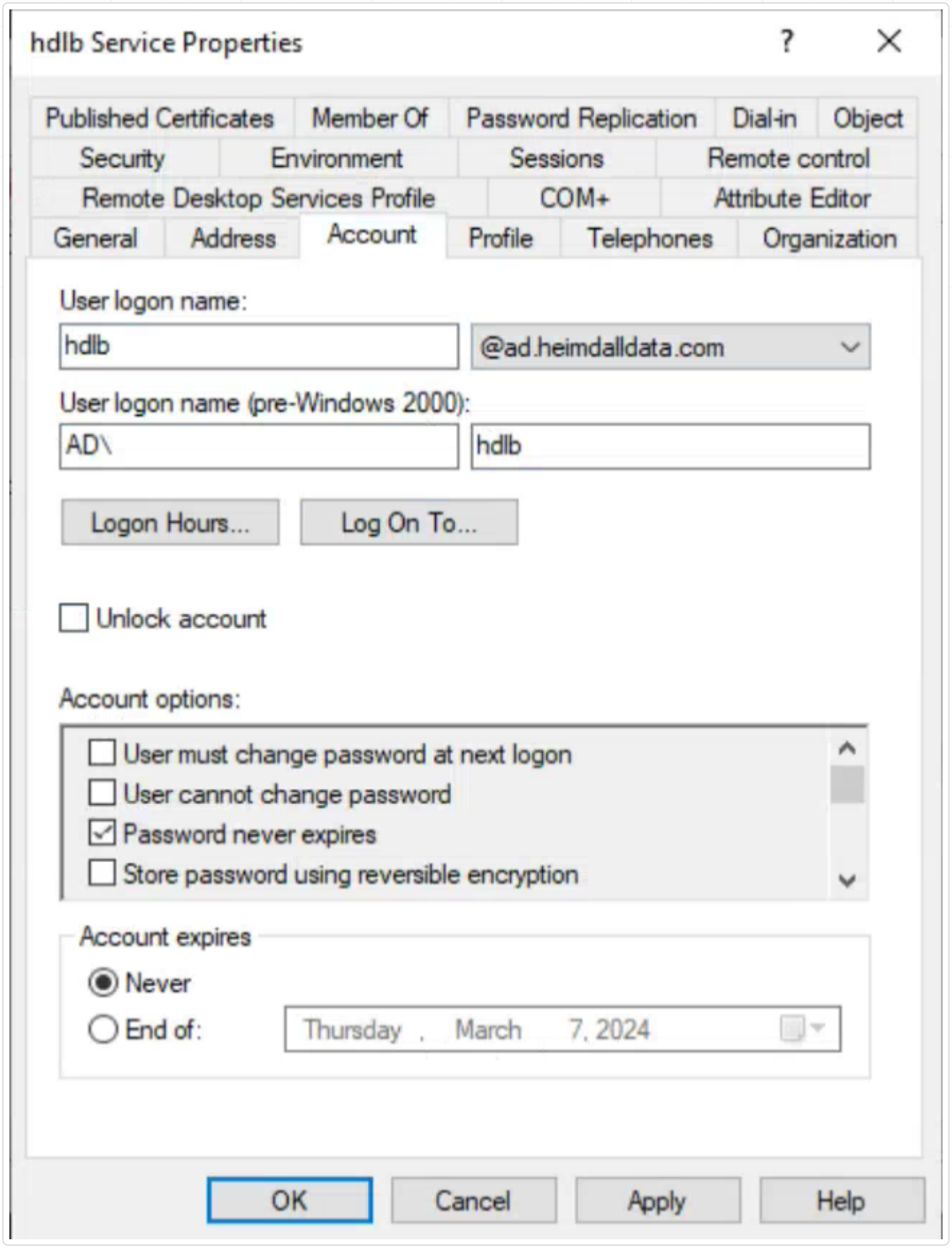
- In every command shown, the realm should be in uppercase, e.g., AD.HEIMDALLDATA.COM
- If you have a CNAME for a load balancer, never use the alias name for the service principal name; use the canonical name instead. Connections using the alias will still work.

Creation of an AD User for Proxy Service

Initiate by establishing an AD user account to symbolize our proxy service. This can be achieved either manually through the Active Directory Users and Computers interface or via the PowerShell command New-ADUser. Ensure the user’s password is set to never expire and Kerberos encryption type AES256 is supported.

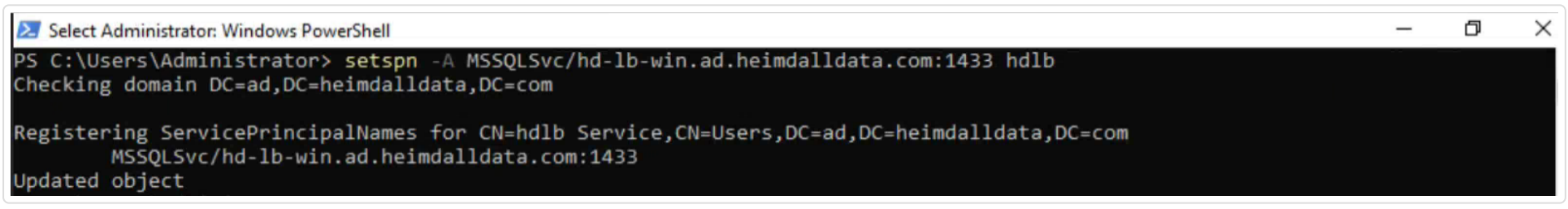
Command: `New-ADUser -Name "user_name" -UserPrincipalName "user_principal_name" -AccountPassword (ConvertTo-SecureString "user_password" -AsPlainText -Force) -Enabled $true -PasswordNeverExpires $true -KerberosEncryptionType AES256`

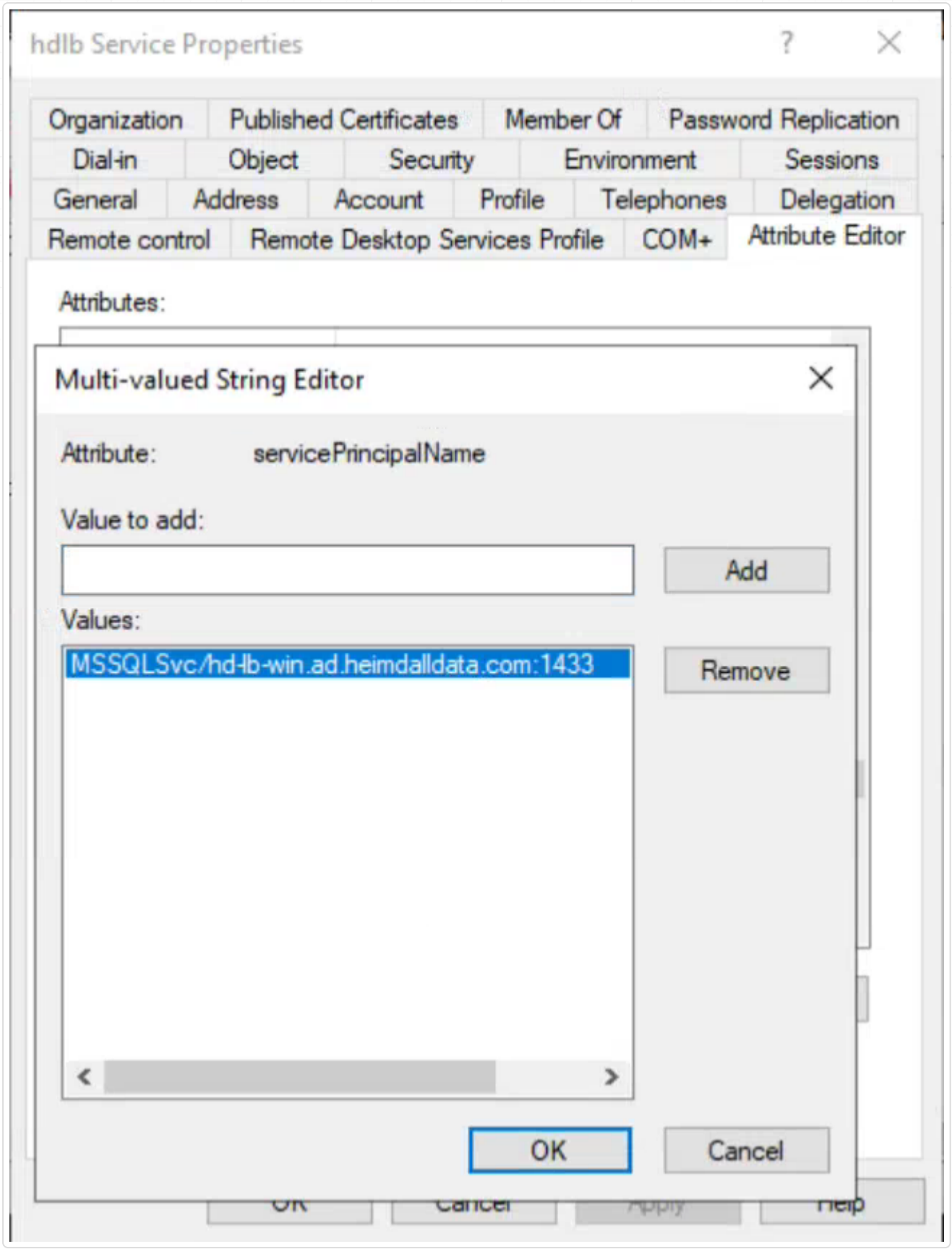
Example: `New-ADUser -Name "hdlb" -UserPrincipalName "hdlb@AD.HEIMDALLDATA.COM" -AccountPassword (ConvertTo-SecureString "user_password" -AsPlainText -Force) -Enabled $true -PasswordNeverExpires $true -KerberosEncryptionType AES256`



Assignment of a Service Principal Name (SPN) to the New User

The user created is **hdlb@ad.heimdalldata.com**. The next step involves assigning a SPN to this user: **MSSQLSvc/hd-lb-win.ad.heimdalldata.com:1433**, indicating the service name, load-balanced domain, and target port, respectively. SPNs can be added manually via the "Attribute Editor" in the user's settings or through the setspn command.





Generation of a Keytab for the SPN

After the user and SPN have been established, generate a keytab file containing this SPN using the ktpass command.

Command `ktpass -princ service/host:port@realm +rndPass -crypto ALL -ptype KRB5_NT_PRINCIPAL -mapuser mapped_user_principal -out keytab_path`

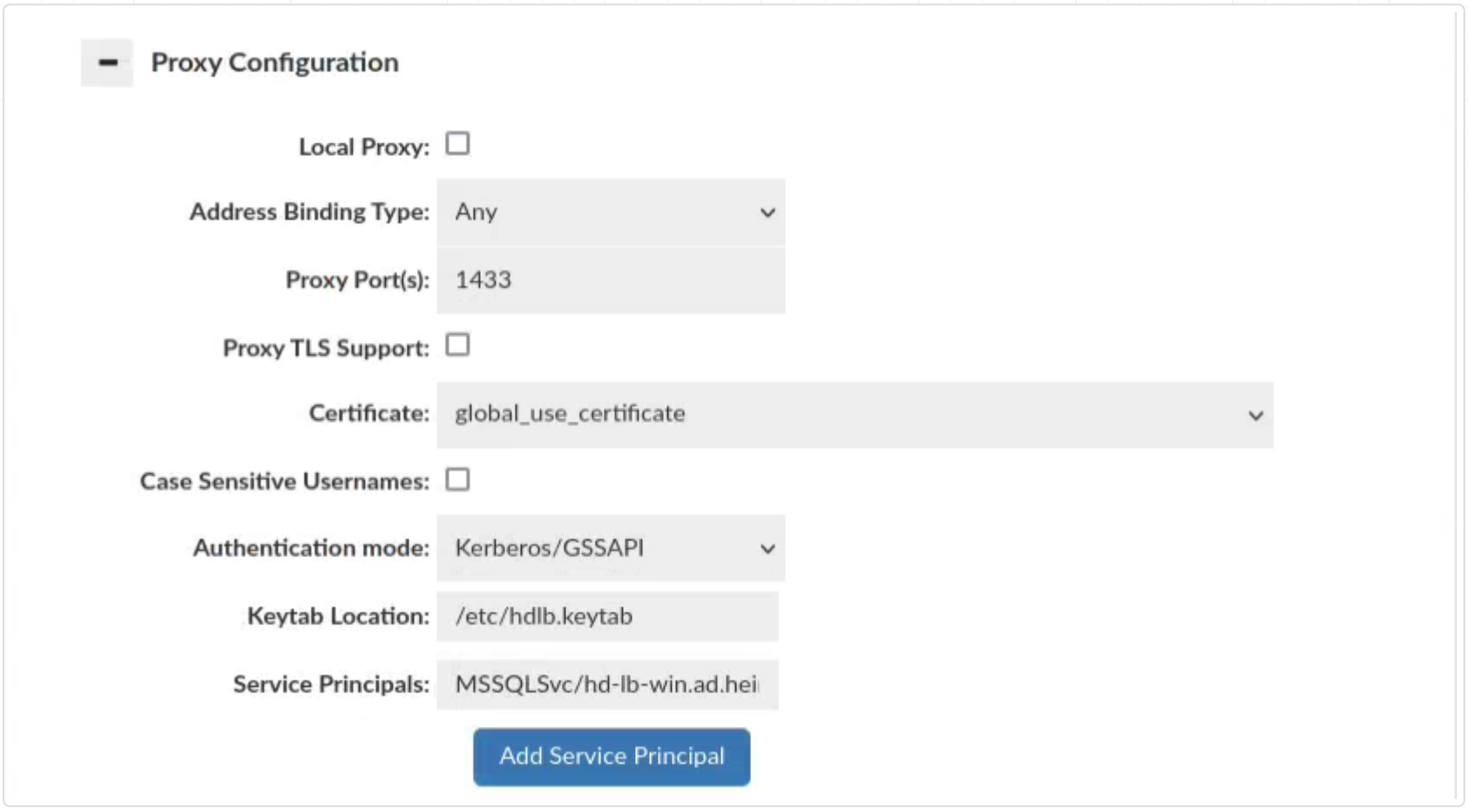
Example `ktpass -princ MSSQLSvc/hd-lb-win.ad.heimdalldata.com:1433@AD.HEIMDALLDATA.COM +rndPass -crypto ALL -ptype KRB5_NT_PRINCIPAL -mapuser hdlb@AD.HEIMDALLDATA.COM -out hdlb.keytab`

Distribution of the Keytab Across Hosts

The generated keytab file must be distributed to every host that will run the proxy service to ensure seamless Kerberos authentication. The distribution method depends on the system's architecture and requirements. A simple method, for example, is to transfer the keytab using the SCP command. However, this approach may be challenging in an autoscaling environment. In such cases, for instance, within an AWS environment, the file can be stored in AWS Secrets and downloaded to a specified path when a new instance is created. This scenario is merely illustrative. It is crucial to ensure that the keytab file includes our SPN and is placed in a location specified in the Heimdall configuration **(Virtual Database/Our VDB/Proxy Configuration/Keytab Location)**.

Configuration of SPN and Keytab in Heimdall

Once the keytab file is distributed, it's crucial to update the Heimdall configuration. This includes specifying the Service Principal Name (SPN) we're using and the location of the keytab file. In our case, we've placed the keytab files at **/etc/hdlb.keytab**, and the SPN designated for our setup is **MSSQLSvc/hd-lb-win.ad.heimdalldata.com:1433**. Below is a screenshot of the Heimdall configuration reflecting these settings:



Dual Authentication

The feature, available only in SQLServer Proxy, facilitates the use of Kerberos Authentication alongside passthrough authentication. However, there's a condition: users must adhere to a single authentication method and cannot switch between them interchangeably.

Dual Authentication can be enabled in VDB -> Proxy Configuration -> Dual Authentication Mode

Authentication Testing

Variety of authentication options ofter requires confirmation that actual settings are working as expected. For checking if set authorization option would work correctly, with expected as correct credentials, should be used Authentication Test tab, which is located below settings of proxy authentication.

Authentication Test tab provides all required credentials inputs to test if configured proxy authentication would work as expected. Worth mentioning is that configured authentication doesn't have to be committed to perform an authentication test. **Warning:** VDB configuration doesn't have to be committed, but all data sources, used by VDB, have to be committed. Any change in Data Source configuration should be followed by committing the change and refreshing page, before using Authentication Test. Only after these actions, Authentication Test results will be reliable.

Authentication testing is an option for checking if only proxy authentication confirms users credentials. Results returned by authentication testing doesn't confirm if given credentials match credentials in a database behind the proxy. If authentication testing returns failure as a result, then proper alert will be shown with description about reason of rejecting credentials.

Basic testing

Basic routine of authentication mainly expects two crucial inputs:

Input	Input name	Type	Description
username	Test User	String	Input for a name of the user
password	Test Password	String	Input for a password of the user

Authentication methods, which only based on these credentials, are: - Passthrough - Proxy Authenticated - LDAP Authentication

For testing with these methods only credentials inputs: Test User and Test Password have to be set. Other test data inputs can be ignored.

Examples

Passthrough examples

Passthrough by default passes all credentials, so every credential combination should return successful result.

Proxy Authenticated examples

Let's assume that we have two users declared for Proxy Authenticated method: `user1` with password `password1`, and `user2` with password `password2`. In this situation below inputs in Authentication Test should be returned as successful.

```
Test User: user1
Test Password: password1
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: <empty or random>
```

```
Test User: user2
Test Password: password2
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: <empty or random>
```

Other combinations of credentials will be returned as rejected by the proxy. Returned alert will contain information that provided credentials didn't match any of declared users, or given user password didn't match to declared password of the user.

LDAP examples

Let's assume that our proxy configuration has been set properly (see section about LDAP Authentication for information about correct configuring) and configuration is pointing to a ldap server to read users only from `GROUP1`, which contains `user1` with password `password1` and `user2` with password `password2`. However, in the ldap server we have declared `GROUP2`, where we have `user3` with password `password3`, but that user shouldn't be used for authentication by the proxy. In this situation below inputs in Authentication Test should be returned as successful.

```
Test User: user1
Test Password: password1
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: <empty or random>
```

```
Test User: user2
Test Password: password2
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: <empty or random>
```

Below credentials and others should be returned as rejected for the assumed situation.

```
Test User: user1
Test Password: password1
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: <empty or random>
```

Extended testing

Some proxy authentication methods requires more data than only username and password to authenticate a user. For actual authentication methods were added below data inputs:

Input	Input name	Type	Description
address	Test IP Address	String	Input for an address from which testing user is connecting to the proxy. Used for testing with SQL Driven Authentication for rules with specified address range.
database	Test Database	String	Input for a name of a database/schema name to which testing user wants to connect. Used for testing with SQL Driven Authentication for rules with specified database name.
ssl	UseSSL	Boolean	Input for a set if authentication process is made in a connection using SSL encryption. Used for testing with SQL Driven Authentication for rules with set connection type as "hostssl" or "hostnossl".

Proxy authentication methods, which uses more than basic data as credentials to authenticate users, are : - SQL Driven Authentication

SQL Driven Authentication Examples

Let's assume that we have defined a single rule for SQL Driven Authentication with below values:

```
connection type: hostssl
auth_method: trust
```

For assumed rule only below input for Authentication Test will pass. As can be seen, rule requires using SSL so all credentials with set using SSL as true will pass through proxy authentication.

```
Test User: <empty or random>
Test Password: <empty or random>
Test Address IP: <empty or random>
Test Database: <empty or random>
UseSSL: true
```

Let's assume that we have defined a single rule like:

```
address: 123.123.123.123
auth_method:trust
```

For assumed rule only below input for Authentication Test will pass. As can be seen, rule requires connection from address 123.123.123.123 so all credentials with set IP address as 123.123.123.123 will pass through proxy authentication.

```
Test User: <empty or random>
Test Password: <empty or random>
Test Address IP: 123.123.123.123
Test Database: <empty or random>
UseSSL: <empty or random>
```

Let's assume that we have defined a single rule like:

```
address: 123.123.123.0
netmask: 255.255.255.0
auth_method:trust
```

For assumed rule only below input for Authentication Test will pass. As can be seen, rule requires connection from address from network 123.123.123.0/24 so all credentials with set IP address from that network will pass through proxy authentication.

```
Test User: <empty or random>
Test Password: <empty or random>
Test Address IP: <any address from range 123.123.123.0-255>
Test Database: <empty or random>
UseSSL: <empty or random>
```

Let's assume that we have defined a single rule like:

```
database: {testdb1}
auth_method:trust
```

For assumed rule only below input for Authentication Test will pass. As can be seen, rule requires connection to database named `testdb1` so all credentials with set connection to database `testdb1` will pass through proxy authentication.

```
Test User: <empty or random>
Test Password: <empty or random>
Test Address IP: <empty or random>
Test Database: testdb1
UseSSL: <empty or random>
```

Let's assume that we have defined a single rule like:

```
connection type: hostssl
user_name: ldapuser1
database: testdb1
address: 127.0.0.0
netmask: 255.255.255.254
auth_method: ldap
options: ldapserver="ldap.example.com" ldapport=389 ldapprefix="CN=" ldapsuffix=",CN=Users, DC=example, DC=com"
```

Additionally, let's assume that we have declared `ldapuser1` with password `ldappassword1` in `CN=Users, DC=example, DC=com` on LDAP server `ldap.example.com:389`. For assumed rule only below input for Authentication Test will pass. Only below credentials will pass, because of strict rule definition and password stored on LDAP server.

```
Test User: ldapuser1
Test Password: ldappassword1
Test Address IP: <127.0.0.0 or 127.0.0.1>
Test Database: testdb1
UseSSL: true
```

Database Orchestration

Different database environments have a variety of tools to provide ways to promote and manage the databases, such as the MySQL Orchestrator. In order to integrate with these tools better, Heimdall has a simplified API call that allows the server state for a data source to be downloaded, adjusted, and uploaded. Examples:

To download the server configuration for a given data source:

```
wget --http-user=admin --http-password=<password> http://demoa.heimdalldata.com:8087/api/config/source/<source name>/servers
```

This will provide a file named "servers" with a format as follows (in json):

```
[
  {
    "name": "Primary",
    "url": "jdbc:postgresql://pg.cfjxl5jnvj49.us-east-1.rds.amazonaws.com:5433/${catalog}",
    "enabled": true,
    "writeable": true,
    "weight": 1,
    "readWeight": 0,
    "writeWeight": 0,
    "active": true,
    "failed": false,
    "maxLag": 2147483647
  },
  {
    "name": "Secondary test",
    "url": "jdbc:postgresql://pg.cfjxl5jnvj49.us-east-1.rds.amazonaws.com:5433/${catalog}?readOnly\u003dtrue",
    "enabled": true,
    "writeable": false,
    "weight": 8,
    "readWeight": 0,
    "writeWeight": 0,
    "active": true,
    "failed": false,
    "maxLag": 2147483647
  }
]
```

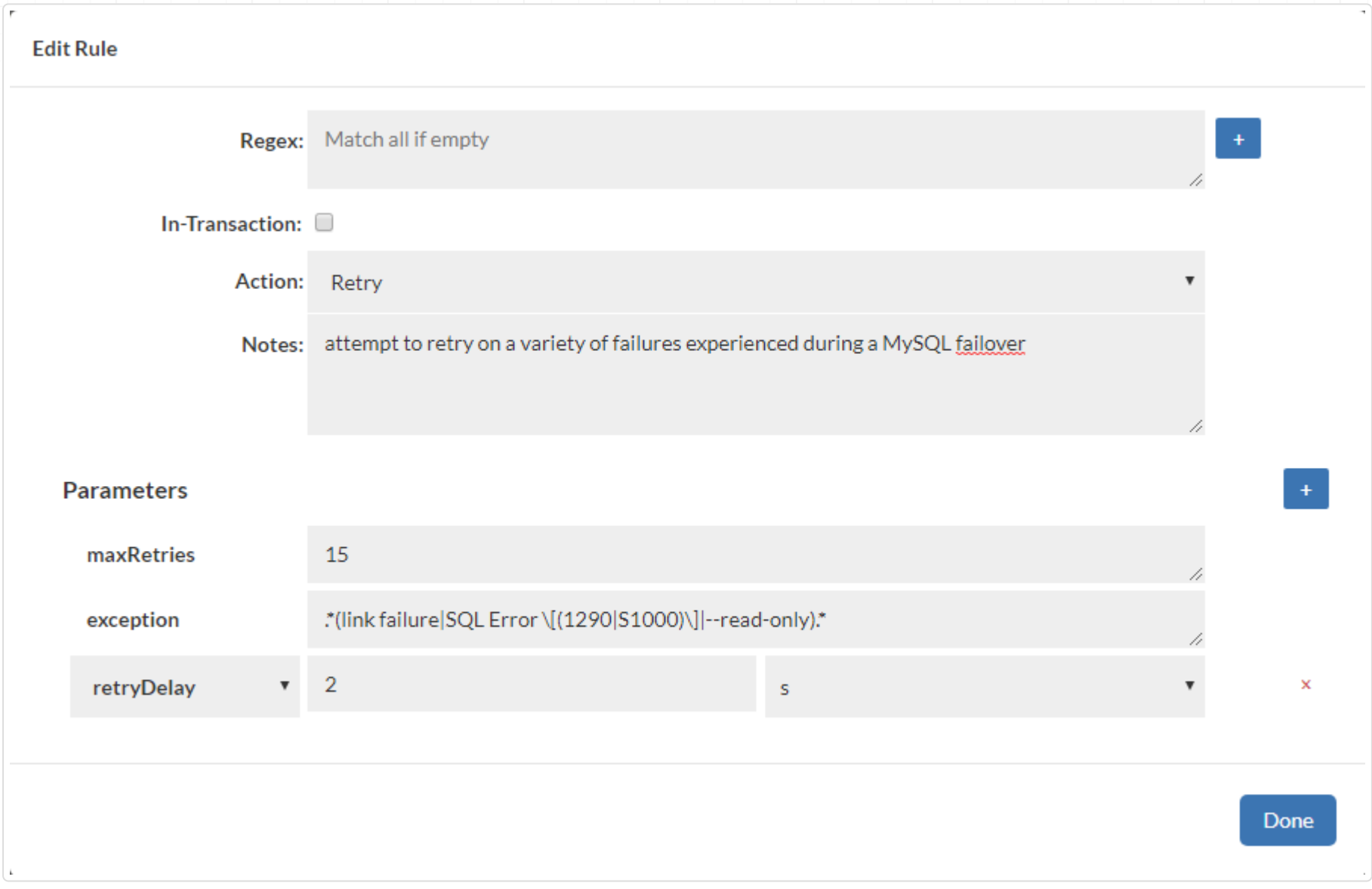
This can then be edited as needed in the context of the events that have happened, and uploaded back to Heimdall with the following:

```
wget --http-user=admin --http-password=<password> --post-file=servers http://demoa.heimdalldata.com:8087/api/config/source/<source n
```

Note: The "1" at the end of the URL represents a configuration version number, not a server ID, and should remain 1 in general.

Query Retry on Exception

Heimdall supports capturing and retrying queries based on the exceptions returned by the underlying JDBC driver. This allows a variety of conditions that would otherwise break an application's processing flow, and allow retry logic to kick in, potentially resolving the issue. This is most useful for transient errors, such as during a failover, a deadlock situation, etc. For example, in order to support a graceful failover of a MySQL cluster (including Aurora), the following rule can be used:



This retry logic is not foolproof however--if in a transaction, retries should not be automatically performed, but should be resolved at the application level. It is impossible to cleanly retry all queries in a multi-query transaction, as one query can result from state that has changed since the transaction started. In auto-commit situations however, this retry logic can be useful to minimize disruption of the application, and to prevent connections from being dropped.

PostgreSQL Specific Information

PostgreSQL Proxy Restrictions

- PostgreSQL allows the user to send multiple statements in one query. Such query is treated as a transaction: if any of statements failed the query is roll backed entirely even in auto commit mode. The PostgreSQL proxy splits such queries to single statement queries and execute them separately via JDBC. This leads to the fact that if one of statements fails and auto commit mode is enabled all statements before it won't be roll backed. To use such a syntax, please use a true transaction start and commit to wrap the queries to avoid any issues.
- The vdb property of **suppressNoResult**=true is available. With Postgres, if an update query is executed via the executeQuery result, it will generate an exception on return saying "No results were returned by the query". In some frameworks, this is detected and suppressed when using the native Postgres driver, but not with the Heimdall driver. In order to work around this behavior, this option can trigger us returning a null instead of a resultset, which appears to allow the calling code to work fine. This applies when using Heimdall as a JDBC driver, but should not apply when as a proxy.

Prepared Statements

Postgres (and other databases) have the option to use prepared statements. When used however, this triggers "state" on the connection and this state will potentially result in sub-optimal behavior of Heimdall or other proxies. In particular, it will break multiplexing behavior, which is needed to dramatically reduce connection counts. In the case of Java applications, there is a parameter that can be set, "preferQueryMode". If set to "simple" it will effectively "de-prepare" the statements in a safe way, and allow multiplexing to work. One caution should be noted however: Many applications rely on an undocumented behavior of Postres with prepared statements where typecasts that are otherwise required may not be required when prepared statements are used. If setting preferQueryMode to simple results in a typecast error, then this is effectively exposing a bug in the application code (or the framework generating the query).

Search_path support

When search paths are used, it is suggested that the search path be configured on the application user instead of dynamically changing the path at runtime. This ensures that a consistent path is used. If inconsistent search paths are used, this can break cache consistency and read/write split handling. To set the search path on a user, use the command:

```
alter user user1 set search_path='my_schema, "$user", public';
```

If this is not sufficient, or if overlapping table names are used across schemas, another resolution can be done to ensure read/write split and multiplexing works. In this case, you can add a JDBC URL parameter of currentSchema=${schema} to the URLs. This will fill in any value from the set search_path into the currentSchema value, and when we pull connections from the connection pools, we will use a connection connected with the proper value. You can then inspect the connections made with the "show pools" command.

Schema Assist for Dependencies

When using Postgres or a Postgres derived database, parsing is performed to detect view dependencies, in order to automate invalidation. This is only performed at proxy start at this time.

PG Notify Support

Postgres supports notification support, i.e. it provides a built-in pub/sub interface. When Postgres is used, this option can be explicitly controlled with a connection property of "hdUsePGNotify" and a value of true or false. To specify the database(s) to listen to, use "hdNotifyDBs" with a comma separated list of database names. Also, * can be used as a wildcard, with -dbname to remove from the list pulled from the database, i.e. "*,-template0,-template1" would be a solid starting point for configuring this feature. The catalogs are cached for up to one minute between attempts to poll, so if new catalogs are added, we will start listening for invalidations within a minute.

When used, this will override the need to have a grid cache for invalidations, although it does not automatically enable multi-primary Heimdall management nodes as a grid cache does. This also enables the below trigger invalidation that leverages the pg_notify interface. The purpose of this option is to allow a simplified deployment without the use of an external cache, but still maintain cache coherency between multiple proxies.

Clear pool option

There is an option to clear a user pool even when it is impossible to get a connection, e.g. when all connections from the pool are busy, they cannot be closed and the maximum number of connections is reached. Two parameters can be set to issue a clear pool before making a connection.

- Application name - setting "ApplicationName" to "clear_pool" when attempting to connect will clear this users user pool of all previous connections. To do so supply the connection URL adding at its end the following parameters: "?ApplicationName=clear_pool&assumeMinServerVersion=VERSION", where VERSION is the version of Postgres Server currently in use. VERSION has to be at least 9.0 for the JDBC driver to recognize custom application name parameters. Alternatively, use psql and provide a custom application name as "application_name=clear_pool" when attempting a connection.

- Custom option - provide clear_pool=true as an option by adding the following to the connection URL: "?options=clear_pool=true". The clear behavior is the same as it is when setting the application name to "clear_pool".

It's recommended to close those connections after connecting as reconnecting may occur which will clear the user's user pool again.

Trigger Invalidation with Postgres

Postgres provides a listen/notify interface that can be used to push invalidation messages directly from the database to the proxy. In order to configure this, use the following:

Connection property of hdUsePGNotify, value of true, then configure the following trigger function and trigger on each table:

```
CREATE OR REPLACE FUNCTION heimdall.changed()
RETURNS TRIGGER AS $$
BEGIN
  PERFORM pg_notify('heimdall', 'invalidate-trigger,'||current_database()||'.'||TG_TABLE_SCHEMA||'.'||TG_TABLE_NAME||','||((date_par
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- for each table on the system, perform the following:
CREATE TRIGGER trigger_table_changed
  BEFORE INSERT OR UPDATE OR DELETE ON test
  EXECUTE PROCEDURE heimdall.changed();
```

The result of this is near instant invalidation of tables, even if a write is done to PostgreSQL without going through Heimdall. Log messages with debug mode will indicate when this is functional.

When this is in place, this adjusts a few other behaviors. When active, invalidations will be issued through the DB notify interface vs. the grid cache as well. This enables local only cache configurations to effectively work without a grid cache. Additionally, if a table is invalidated via the grid cache with the above configuration, then additional notification messages will not be generated for that table by Heimdall, but other tables that have not been invalidated by the trigger will continue to have invalidations from Heimdall. This allows a mixed setup, so that frequently updated tables may have the trigger, while infrequently updated tables (or new tables) are invalidated directly. This will reduce the overhead of maintaining the triggers as the DB schema changes, while optimizing the number of invalidation messages needed.

This also supports invalidation of tables that are changed via stored procedures, other triggers, or are modified in other ways. It is still a requirement that stored procedures be tagged with the tables they are reading, but writes will be automatically accounted for.

Alternate Trigger Invalidation with Postgres (via polling)

This technique requires polling against the database, which adds load and latency to the invalidation process, but is documented for completion.

Connection properties:

- dbTriggerTracking=true
- dbChanges=select ts,name from heimdall.tableupdates where ts > @ts
- dbTimeMs=select (date_part('epoch', now())*1000)::bigint

Note: The following example is drafted for use with older Postgres, so doesn't use the ON CONFLICT syntax available in PG 9.5+

```
CREATE SCHEMA heimdall;

CREATE TABLE heimdall.tableupdates (
name text PRIMARY KEY,
ts bigint not null );

CREATE OR REPLACE FUNCTION heimdall.changed()
RETURNS TRIGGER AS $$
BEGIN
  LOOP
    UPDATE heimdall.tableupdates SET ts=(date_part('epoch', now())*1000)::bigint, name=current_database()||'.'||TG_TABLE_SCHEMA||'.'
    IF found THEN
      RETURN NEW;
    END IF;
    BEGIN
       INSERT INTO heimdall.tableupdates VALUES ((date_part('epoch', now())*1000)::bigint, current_database()||'.'||TG_TABLE_SCHEMA|
       RETURN NEW;
    EXCEPTION WHEN unique_violation THEN
    END;
  END LOOP;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER trigger_table_changed
  BEFORE INSERT OR UPDATE OR DELETE ON test
  FOR EACH ROW
  EXECUTE PROCEDURE heimdall.changed();
```

Active Directory Password & Group Synchronization

```
CREATE OR REPLACE FUNCTION heimdall.sync_user(username text, password text, ldapgroups text)
 RETURNS void
 LANGUAGE plpgsql
AS $function$
DECLARE
    sql       text;
    grouplist text[];
    g         text;
    groupsToRevoke text[];
    lookupTableName text;
    lookupTableKey text; -- ldap_group
    lookupTableValue text; -- role
    lookupLdapGroups text[];
    lookupRoles text[];
    lg text;
    current_schema_name text;
    already_processed_group boolean;
    create_roles_if_missing boolean;
BEGIN
    current_schema_name := current_schema();
    create_roles_if_missing := TRUE; -- set to false to prevent role creation
    lookupTableName := 'lookup_table';
    lookupTableKey := 'ldap_group';
    lookupTableValue := 'role';

-- prevent syncing special users (if necessary)
IF username IN ('api') THEN
    RAISE NOTICE '%.sync_user: skip syncing for special user "%"', current_schema_name, username;
    RETURN;
END IF;

-- do not synchronize any user that is flagged as superuser
IF EXISTS(SELECT usesuper FROM pg_user WHERE user = pg_catalog.quote_literal(username)) THEN
    RAISE NOTICE '%.sync_user: User % is superuser, not synchronizing', current_schema_name, username;
    RETURN;
END IF;

IF EXISTS (
    SELECT 1
    FROM information_schema.tables
    WHERE table_schema = current_schema_name AND table_name = lookupTableName
) THEN
    -- get lookupLdapGroups
    EXECUTE 'SELECT array_agg(lt.' || lookupTableKey || ') FROM ' || current_schema_name || '.' || lookupTableName || ' AS lt'
    INTO lookupLdapGroups;
    RAISE NOTICE '%.sync_user: Lookup table loaded: %', current_schema_name, lookupLdapGroups;
END IF;

-- sync user and password
BEGIN
    sql := 'CREATE USER ' || quote_ident(username) || ' WITH PASSWORD ''' || password || ''';';
    EXECUTE sql;
    sql := 'CREATE USER ' || quote_ident(username) || ' WITH PASSWORD ''' || 'REDACTED' || ''';';
    RAISE NOTICE '%.sync_user: % created', current_schema_name, sql;
 EXCEPTION
    WHEN Duplicate_Object THEN -- alter password when user exists
        sql := 'ALTER USER ' || quote_ident(username) || ' WITH PASSWORD ''' || password || ''';';
        BEGIN
            EXECUTE sql;
            sql := 'ALTER USER ' || quote_ident(username) || ' WITH PASSWORD ''' || 'REDACTED' || ''';';
            RAISE NOTICE '%.sync_user: % updated', current_schema_name, sql;
        EXCEPTION
            WHEN SQLSTATE 'XX000' THEN
                IF SQLERRM LIKE '%tuple concurrently updated%' THEN
                    -- ignore, may occur as we use connection pooling for synchronization
                ELSE
                    RAISE;
                END IF;
        END;
 END;

-- sync group/role membership
 grouplist := string_to_array(ldapgroups, ',');
 groupsToRevoke := array(SELECT b.rolname
                         FROM (SELECT rolname, oid
                               FROM pg_roles
                               WHERE rolname != ALL(grouplist) EXCEPT SELECT usename, usesysid
                                                                     FROM pg_user) b
                                                                     WHERE pg_has_role(username, b.oid, 'member'));
```

```
IF array_length(groupsToRevoke, 1) > 0 THEN
    FOR i IN array_lower(groupsToRevoke, 1) .. array_upper(groupsToRevoke, 1) LOOP
        g := groupsToRevoke[i];
        IF position('=' in g) > 0 THEN -- capture the name if ldapgroups is passed in as cn=groupname
            g := trim(TRAILING '"' FROM split_part(groupsToRevoke[i], '=', 2));
        ELSE
            g := trim(BOTH '"' FROM trim(BOTH FROM groupsToRevoke[i]));
        END IF;

        -- now revoke membership of the role from the user
        sql := 'REVOKE ' || quote_ident(g) || ' FROM ' || quote_ident(username) || ';';
        RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
        IF sql IS NOT NULL THEN
            BEGIN
                EXECUTE sql;
            EXCEPTION
                WHEN SQLSTATE 'XX000' THEN
                    IF SQLERRM LIKE '%tuple concurrently updated%' THEN
                        -- ignore, may occur as we use connection pooling for synchronization
                    ELSE
                        RAISE;
                    END IF;
            END;
        END IF;
    END LOOP;
END IF;

IF array_length(grouplist, 1) > 0 then
    FOR i IN array_lower(grouplist, 1) .. array_upper(grouplist, 1) LOOP
        g := grouplist[i];
        already_processed_group := false;
        RAISE NOTICE '%.sync_user: processing group: %', current_schema_name, g;
        IF position('=' in g) > 0 THEN -- capture the name if ldapgroups is passed in as cn=groupname
            g := trim(TRAILING '"' FROM split_part(grouplist[i], '=', 2));
        ELSE
            g := trim(BOTH '"' FROM trim(BOTH FROM grouplist[i]));
        END IF;

       -- handle additional ldap_group -> role(s) aliases
       IF array_length(lookupLdapGroups, 1) > 0 then
            FOR lg_i IN array_lower(lookupLdapGroups, 1) .. array_upper(lookupLdapGroups, 1) LOOP
            lg := lookupLdapGroups[lg_i];
            RAISE NOTICE '%.sync_user: processing group: % vs. lookupGroup: %', current_schema_name, g, lg;
                IF lg = g THEN
                    -- get all roles mapped to given ldap_group
                    EXECUTE 'SELECT array_agg(lt.' || lookupTableValue || ') FROM ' || current_schema_name || '.' || lookupTable
                    INTO lookupRoles;

                    RAISE NOTICE '%.sync_user: ldap_group % mappings: %', current_schema_name, lg, lookupRoles;

                -- if role is an empty string then no role is mapped for given ldap_group
                IF array_length(lookupRoles, 1) > 0 THEN
                    -- process all roles mapped to given ldap_group
                    FOR r_i IN array_lower(lookupRoles, 1) .. array_upper(lookupRoles, 1) LOOP
                        g := lookupRoles[r_i];
                    IF g IS NOT NULL AND g <> '' THEN
                        IF create_roles_if_missing THEN
                            BEGIN
                                sql := 'CREATE ROLE ' || quote_ident(g);
                                EXECUTE sql;
                                RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
                            EXCEPTION
                                WHEN Duplicate_Object THEN
                                    RAISE NOTICE '%.sync_user: role % already exists', current_schema_name, g;
                            END;
                        END IF;
                        -- now grant membership of the role to the user
                        BEGIN
                            sql := 'GRANT ' || quote_ident(g) || ' TO ' || quote_ident(username) || ';';
                            EXECUTE sql;
                            RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
                        EXCEPTION
                            WHEN SQLSTATE '42704' THEN
                            RAISE NOTICE '%.sync_user: role % does not exist', current_schema_name, g;
                        END;
                        already_processed_group := true;
                    END IF;
                    END LOOP;
                END IF;
```

```
        END IF;
    END LOOP;
END IF;

--lookup_table does not exist
IF g IS NOT NULL AND g <> '' AND already_processed_group = FALSE THEN
    -- make sure groups are created
    IF create_roles_if_missing THEN
        BEGIN
            sql := 'CREATE ROLE ' || quote_ident(g);
            EXECUTE sql;
            RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
        EXCEPTION
            WHEN Duplicate_Object THEN
                RAISE NOTICE '%.sync_user: role % already exists', current_schema_name, g;
        END;
    END IF;

    BEGIN
        -- now grant membership of the role to the user
        sql := 'GRANT ' || quote_ident(g) || ' TO ' || quote_ident(username) || ';';
        EXECUTE sql;
        RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
    EXCEPTION
        WHEN SQLSTATE '42704' THEN
                RAISE NOTICE '%.sync_user: role % does not exist', current_schema_name, g;
            END;
        END IF;

    END LOOP;
END IF;
RETURN;
END;
$function$
;
```

Active Directory Password & Group Synchronization extra options

By default, a role with the same name is created in the database for each extracted ldap group. By creating the table below for each key found (ldap_group), a role with the specified name in the value (role) will be created instead. Composite Primary Key allows to map more than one role to particular ldap_group; If 'role' is empty then no role is created and granted to user for this particular ldap_group.

```
CREATE TABLE IF NOT EXISTS heimdall.lookup_table (
        ldap_group varchar(255),
        role varchar(64),
        PRIMARY KEY (ldap_group, role)
    )
```

MySQL Specific Information

Common Issues

When testing with the mysql commandline client, use -A to connect. This avoids an issue where the client is itself using a deprecated feature the MySQL protocol that can trigger issues in a proxy environment.

One of the most common issues with MySQL is the character-set being used by the database. As JDBC attempts to negotiate with the server, if the server is set to anything other than UTF8MB4 (along with the tables) this can cause issues with certain characters not being handled properly. Here is an example configuration for the my.cnf that can help avoid any such issues:

```
[mysqld]
character-set-server = utf8mb4
collation-server = utf8mb4_unicode_ci

[mysql]
default-character-set = utf8mb4
```

In addition to this however, it is also important to set the character set of the database itself to utfmb4:

```
ALTER DATABASE <dbname> DEFAULT COLLATE utf8_unicode_ci;
```

If all of these are not in alignment, then various character encoding issues may occur.

One further impact--if a database is set to utf8 instead of utf8mb4, most functions may appear to work properly, BUT it can impact performance. If an index is done on a column that is coded in utf8, but the query is received in utf8mb4, then the index may not be used as part of the optimization. This can be a very subtle and difficult issue to track down, but using "explain" on the query will show the difference in the execution plan with and without heimdall, but it will not say why the index is not used.

Several MySQL specific behaviors are included and detected by Heimdall, such as:

SELECT SQL_CALC_FOUND_ROWS: queries with this pattern will be treated as updates.

Using the "getBoolean()" method on a string type will look at the first character and match per the logic used in the MySQL driver, i.e. any string starting with "y" will test as "yes" or true. Same with "t".

Known Incompatibility: There is a rare but known issue when using non-prepared statements when inserting binary data using an insert command. In this configuration, the insert uses inline binary data in what is otherwise a UTF-8 string. When received, the entire string is processed as UTF-8, and results in the binary data being modified when transmitted from the proxy to the database. This problem is NOT observed if using true server-side prepared statements for the insert. If you observe issues where binary data appears corrupted, and you believe you are using a prepared statement, please check if your driver is implicitly converting prepared statements into "client" side prepared statements, as most DO do this for performance reasons. For example, with the MySQL JDBC driver, you will need to add the option "useServerPrepStmts=true". Please see dev.mysql.com for more information.

sql_mode

If a custom sql mode is needed, it can be set in the data source properties with the key of "sessionVariables" and a value of “sql_mode='whatever'“.

MariaDB

Due to changes in the version string and how the Oracle MySQL driver interacts with the MySQL server, when using MariaDB, it is advised to set it to use a version string such as "5.7.19-10.2.22-MariaDB". The first part is the MySQL compatibility version--if this string is higher than either 5.6.19, or 8.0.3, then this may trigger an error about an invalid counter of "transaction_isolation". Please see the MariaDB documentation for information on how to change this setting.

Trigger Invalidation with MySQL

Connection properties:

- dbTriggerTracking=true
- dbChanges=select ts,name from heimdall.table_change where ts > @ts
- dbTimeMs=select get_time_ms()

Script to configure MySQL for Trigger expiry:

```
#!/bin/bash

database=tpch
host=127.0.0.1
port=3306
user=root
password=secret
command="mysql -N --force --host=$host --port=$port --user=$user --password=$password"

(
# setup the heimdall table for all Heimdall activities
echo "DROP DATABASE IF EXISTS heimdall;
CREATE DATABASE heimdall;
USE heimdall;

DELIMITER \$\$

#----------code for MySQL (including pre-5.6 versions) to get the current time to the ms level---------
DROP FUNCTION IF EXISTS get_time_ms\$\$
CREATE DEFINER=root@localhost FUNCTION get_time_ms()
  RETURNS BIGINT
  READS SQL DATA
  NOT DETERMINISTIC
BEGIN
 DECLARE time_ms BIGINT;
 SELECT conv(
  concat(
   substring(uid,16,3),
   substring(uid,10,4),
   substring(uid,1,8)),16,10) div 10000 - (141427 * 24 * 60 * 60 * 1000)
  into time_ms FROM (select uuid() uid) as alias;
 RETURN time_ms;
END\$\$

#---------code for trigger based expiry--------
CREATE TABLE table_change (
   ts BIGINT DEFAULT 0,
   name VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_bin,
   updates INT DEFAULT 0,
   inserts INT DEFAULT 0,
   deletes INT DEFAULT 0,
   PRIMARY KEY (name)) ENGINE = MEMORY\$\$

DROP PROCEDURE IF EXISTS changes\$\$
CREATE DEFINER=root@localhost PROCEDURE changes(time BIGINT)
BEGIN
    SELECT ts,name,updates,inserts,deletes from table_change where ts >= time;
END\$\$

DROP PROCEDURE IF EXISTS track_changes\$\$
CREATE DEFINER=root@localhost PROCEDURE track_changes(table_name VARCHAR(64), type VARCHAR(16))
BEGIN
    CASE type
    WHEN 'update' THEN INSERT INTO heimdall.table_change VALUES (get_time_ms(), table_name, 1, 0, 0) ON DUPLICATE KEY
        UPDATE updates=updates+1,ts=get_time_ms();
    WHEN 'insert' THEN INSERT INTO heimdall.table_change VALUES (get_time_ms(), table_name, 0, 1, 0) ON DUPLICATE KEY
        UPDATE inserts=inserts+1,ts=get_time_ms();
    WHEN 'delete' THEN INSERT INTO heimdall.table_change VALUES (get_time_ms(), table_name, 0, 0, 1) ON DUPLICATE KEY
        UPDATE deletes=deletes+1,ts=get_time_ms();
    END CASE;
END\$\$
USE $database\$\$
DROP USER 'heimdall'@'%'\$\$
CREATE USER 'heimdall'@'%' IDENTIFIED BY 'heimdalldemo'\$\$
GRANT EXECUTE on heimdall.* to 'heimdall'@'%'\$\$
GRANT SELECT on tpch.* to 'heimdall'@'%'\$\$
FLUSH PRIVILEGES\$\$
"

echo "show tables" | $command $database | while read table; do

if [ "$table" != "table_change" -a "n$table" != "n" ]; then
    echo "INSERT INTO heimdall.table_change (ts,name) values (UNIX_TIMESTAMP()*1000,'${database}.${table}')\$\$"

        echo "DROP TRIGGER IF EXISTS ${database}.${table}_update\$\$
        CREATE TRIGGER ${database}.${table}_update AFTER UPDATE ON ${database}.${table} FOR EACH ROW
            CALL heimdall.track_changes('${database}.${table}', 'update')\$\$"

        echo "DROP TRIGGER IF EXISTS ${database}.${table}_insert\$\$
```

```
            CREATE TRIGGER ${database}.${table}_insert AFTER INSERT ON ${database}.${table} FOR EACH ROW
            CALL heimdall.track_changes('${database}.${table}', 'insert')\$\$"

        echo "DROP TRIGGER IF EXISTS ${database}.${table}_delete\$\$
            CREATE TRIGGER ${database}.${table}_delete AFTER DELETE ON ${database}.${table} FOR EACH ROW
            CALL track_changes('${database}.${table}', 'delete')\$\$"
fi

done )| $command
```

Microsoft SQL Server Specific Information

- Currently, the SQL Server proxy only supports SQL Server logins. If the proxy is running on Windows, it is possible to configure internal authentication on the SQL Server JDBC driver. Please contact support for more details if this is necessary.
- When using the Microsoft Azure Database, the setCatalog() function is not usable to change the catalog on a connection. To support an application selecting the database on the initial connection, use the string ${database} in the JDBC URL to set the database, i.e. ";databaseName=${database}" at the end of the url. The value ${database} will be extracted from the client login to populate this value.
- When performing explain extraction, the option checkTableMismatch can be used (any value) to trigger testing and printing if a the tables extracted from the explain match what is used internally by Heimdall. This can be used to track down table extraction issues.
- If using Mirroring (not Always-On) for redundancy, transactions will fail. This can apply if using SQL Server in AWS for versions prior to 2016, or older instances spun up before May 2019 with failover. Please use Always-On, as transaction handling with mirroring is not easily supported.
- Applications that depend on session recovery are likely to not work properly, in particular if session data is used as part of record/table locking mechanisms, such as the SP "applock_mode" with the lock owner of "session". One example of a library that uses this is "Hangfire".
- Applications that require MARS (multiple active result sets) may not work properly. The proxy will actively negotiate this option to be off with clients, which normally is sufficient to allow the applications to work. In some rare cases however, code needs to be modified to read one full result-set into memory before opening a new result-set.

Schema Assist for Dependencies

When using SQL Server, parsing is performed to detect view dependencies along with fixed (not dynamic) dependencies in stored procedures, in order to automate invalidation. This is only performed at proxy start at this time.

Trigger Invalidation with SQL Server

Connection properties:

- dbTriggerTracking=true
- dbChanges=select ts,name from heimdall.table_change where lastupdate > @ts
- dbTimeMs=select heimdall.get_time_ms()

```
CREATE DATABASE [heimdall]
GO

/*** create the trigger tracking table ***/
USE [heimdall]
CREATE TABLE [dbo].[table_change](
    [ts] [bigint] NULL,
    [name] [varchar](64) NOT NULL,
    CONSTRAINT [PK_table_change] PRIMARY KEY ([name]));

CREATE FUNCTION [dbo].[get_time_ms]() RETURNS BIGINT
AS
BEGIN
DECLARE @timems BIGINT
set @timems= (SELECT cast(DATEDIFF(s, '1970-01-01 00:00:00', GETUTCDATE()) as BIGINT)*1000+cast(DATEDIFF(ms,CONVERT(VARCHAR,GETUTCDA
return @timems
END

/*** Create the tracking procedure, which will be used by the triggers ***/
CREATE OR ALTER PROCEDURE [heimdall].[dbo].[track_changes]
    @tablename NVARCHAR(128)
as
    SET NOCOUNT OFF
    UPDATE [heimdall].[dbo].[table_change] SET ts=[heimdall].[dbo].get_time_ms() WHERE name = @tablename
    IF @@ROWCOUNT != 0
        RETURN
    INSERT INTO [heimdall].[dbo].[table_change] VALUES ( dbo.get_time_ms(), @tablename )
GO

/*** trigger to apply to each table that needs trigger based eviction ***/
/**** for table vevo.dbo.Product ***/
USE [vevo2]; CREATE OR ALTER TRIGGER [dbo].[Product_changed]
    ON [vevo2].[dbo].[Product]
    AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    exec [heimdall].[dbo].[track_changes] 'vevo2.dbo.Product'
END
GO
```

Active Directory Password & Group Synchronization

```
CREATE OR ALTER PROCEDURE heimdall.sync_user
    (
    @username nvarchar(max),
    @password nvarchar(max),
    @ldapgroups nvarchar(max)
    )
AS
BEGIN
    --Logic variables
    DECLARE @sql nvarchar(max);
    DECLARE @roleName nvarchar(max);
    DECLARE @currentRow INT = 1;
    DECLARE @rowCount INT;
    CREATE TABLE #groupList (id INT IDENTITY(1,1), g varchar(max)); --table variable can't be used in dynamic SQL
    CREATE TABLE #groupsToRevoke (id INT IDENTITY(1,1), g varchar(max));
    CREATE TABLE #oneToManyMappings (id INT IDENTITY(1,1), g varchar(max));

    --Configurable variables
    DECLARE @currentSchemaName varchar(max) = 'heimdall';
    DECLARE @lookupTableName varchar(max) = 'lookup_table';
    DECLARE @lookupTableKey varchar(max) = 'ldap_group';
    DECLARE @lookupTableValue varchar(max) = 'role_name';
    DECLARE @createRolesIfMissing BIT = 1; -- if set to 1 roles are created and membership is added to the user

    -- Put ldapgroups to table
    INSERT INTO #groupList (g)
    SELECT LTRIM(RTRIM(value))
    FROM STRING_SPLIT(@ldapgroups, ',');

    -- Check if lookup_table exists
    IF (EXISTS (SELECT *
                FROM INFORMATION_SCHEMA.TABLES
                WHERE TABLE_SCHEMA = @currentSchemaName
                AND  TABLE_NAME = @lookupTableName))
    BEGIN
        -- One to many mappings of ldap_groups
        SET @sql = CONCAT(
        'INSERT INTO #oneToManyMappings (g)
        SELECT lt.', QUOTENAME(@lookupTableKey),
        ' FROM ', QUOTENAME(@lookupTableName), ' lt
        INNER JOIN #groupList gl ON lt.', QUOTENAME(@lookupTableKey), ' = gl.g
        GROUP BY lt.', QUOTENAME(@lookupTableKey),
        ' HAVING COUNT(1) > 1;');
        EXEC sp_executesql @sql;
        PRINT @sql;

        -- Delete groups which has one to many mapping
        SET @sql = 'DELETE t1 FROM #groupList t1 JOIN #oneToManyMappings t2 ON t1.g = t2.g'
        EXEC sp_executesql @sql;
        PRINT @sql;

        -- Insert all one to many mappings (corresponding roles)
        SET @sql = CONCAT(
       'INSERT INTO #groupList
        SELECT ', QUOTENAME(@lookupTableValue),
        ' FROM ', QUOTENAME(@lookupTableName),
        ' lt INNER JOIN #oneToManyMappings otmm ON lt.', QUOTENAME(@lookupTableKey), ' = otmm.g;');
        EXEC sp_executesql @sql;
        PRINT @sql

        -- Replace ldap groups with it's corresponding role names from lookup_table
        SET @sql = CONCAT(
        'UPDATE gl SET gl.g = lt.', QUOTENAME(@lookupTableValue),
        ' FROM #groupList gl JOIN ',
        QUOTENAME(@currentSchemaName), '.', QUOTENAME(@lookupTableName), ' lt ON gl.g = lt.', QUOTENAME(@lookupTableKey));
        EXEC sp_executesql @sql;
        PRINT @sql;
    END

    -- Create login if not exists, alter otherwise
    IF NOT EXISTS
      (SELECT name
       FROM master.sys.server_principals
       WHERE name = @username)
    BEGIN
        SET @sql = CONCAT('CREATE LOGIN ', QUOTENAME(@username), ' WITH PASSWORD = ', QUOTENAME(@password, ''''));
        EXEC sp_executesql @sql;
        PRINT @sql;
    END
```

```
ELSE
BEGIN
    SET @sql = CONCAT('ALTER LOGIN ', QUOTENAME(@username), ' WITH PASSWORD = ', QUOTENAME(@password, ''''));
    EXEC sp_executesql @sql;
    PRINT @sql;
END

-- Create user if not exists, alter otherwise
IF NOT EXISTS
   (SELECT 1
    FROM sys.sysusers
    WHERE name = @username)
BEGIN
    SET @sql = CONCAT('CREATE USER ', QUOTENAME(@username), ' FOR LOGIN ', QUOTENAME(@username));
    EXEC sp_executesql @sql;
    PRINT @sql;
END
ELSE
BEGIN
    SET @sql = CONCAT('ALTER USER ', QUOTENAME(@username), ' WITH LOGIN = ', QUOTENAME(@username));
    EXEC sp_executesql @sql;
    PRINT @sql;
END

-- Get roles which user is not a member of anymore
INSERT INTO #groupsToRevoke (g)
(
    SELECT r.name
    FROM sys.database_role_members drm
             JOIN sys.database_principals r ON drm.role_principal_id = r.principal_id
             JOIN sys.database_principals u ON drm.member_principal_id = u.principal_id
    WHERE u.name = @username
    EXCEPT
    SELECT g
    FROM #groupList
);

SET @rowCount = (SELECT COUNT(*) FROM #groupsToRevoke);

-- Drop membership of roles which user is not a member of anymore
WHILE @currentRow <= @rowCount
BEGIN
    SET @roleName = (SELECT g FROM #groupsToRevoke WHERE id = @currentRow);
    SET @sql = CONCAT('ALTER ROLE ', QUOTENAME(@roleName), ' DROP MEMBER ', QUOTENAME(@username));
    EXEC sp_executesql @sql;
    PRINT @sql;

    SET @currentRow += 1;
END

SET @currentRow = 1;
SET @rowCount = (SELECT MAX(id) FROM #groupList)
-- Iterate over groupList and create roles
WHILE @currentRow <= @rowCount
BEGIN
    SET @roleName = (SELECT g FROM #groupList WHERE id = @currentRow);

    IF @roleName IS NOT NULL AND @roleName <> '' AND @createRolesIfMissing = 1 -- If 'roleName' is empty then no role is created
    BEGIN
        -- Create role if not exists
        IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name = @roleName AND type = 'R')
            BEGIN
                SET @sql = CONCAT('CREATE ROLE ', QUOTENAME(@roleName));
                EXEC sp_executesql @sql;
                PRINT @sql;
            END

            -- Grant role to the user
            SET @sql = CONCAT('ALTER ROLE ', QUOTENAME(@roleName), ' ADD MEMBER ', QUOTENAME(@username));
            EXEC sp_executesql @sql; -- Won't throw an exception if user is already a member of the given role
            PRINT @sql;
    END

    SET @currentRow += 1;
END
-- Make sure we drop these to prevent reuse issues on the same connection
DROP TABLE #groupList;
DROP TABLE #groupsToRevoke;
END
```

Active Directory Password & Group Synchronization extra options

By default, a role with the same name is created in the database for each extracted ldap group. By creating the table below for each key found (ldap_group), a role with the specified name in the value (role) will be created instead. Composite Primary Key allows to map more than one role to particular ldap_group; If 'role' is empty then no role is created and granted to user for this particular ldap_group.

```
CREATE TABLE dbo.lookup_table (
    ldap_group varchar(255),
    role_name varchar(64),
    PRIMARY KEY (ldap_group, role_name)
)
```

Amazon RDS for SQLServer

Adjustments to the 'master' database are presently limited within Amazon RDS for SQL Server. When the 'master' database is configured in the datasource or left unspecified, certain functionalities, such as LDAP synchronization, may not work properly. To resolve this issue, clients are recommended to specify the default database in the Datasource tab of the JDBC URL. A representative JDBC URL format is demonstrated as follows: "jdbc:sqlserver://{server_name}:1433;databaseName=${database}", then i properties define defaultCatalog with created database name, in our case [testDB].

If we would to set up a proxy with a new user on Amazon RDS for SQLServer, we can use the following example: Connect with admin user to database master. Then execute queries similar to those:

```
CREATE DATABASE [testDB];
USE [testDB];
CREATE LOGIN [test_user] WITH PASSWORD = 'test_user',  DEFAULT_DATABASE=[testDB];
CREATE USER [test_user] FOR LOGIN [test_user];
CREATE SCHEMA heimdall;
GRANT ALTER, CONTROL, EXECUTE, SELECT, INSERT, UPDATE, DELETE
    TO [test_user];
GRANT CREATE TABLE TO [test_user];
GRANT CREATE PROCEDURE TO [test_user];
```

While using ldap in VDB the command for syncing user will look like this:
EXECUTE testDB.heimdall.sync_user N'${user}', N'${password}', N'${ldapgroups}'

For using load-balancing features we will need to run:

```
GRANT VIEW DATABASE STATE TO [test_user];
GRANT VIEW SERVER STATE TO [test_user]; -- has to be run on master database
```

Database name, and username should be adjusted to the clients need. At the end, created user should be placed in Datasource tab, and JDBC URL should point into defined database.

AWS Redshift

Redshift Drivers

In some situations, you may find that the Heimdall management console can test successfully to Redshift, but the proxy does not work properly. If you inspect the logs, you may see the console is reporting a 403 exception. This is due to the proxy trying to load additional jar files that normally are contained from the Redshift zip file.

To resolve this, first extract all the jar files from the Redshift zip file, which contains all the jar dependencies for the Redshift driver, including the AWS SDK. Next, in the driver configuration, add all the jar files to the driver config. This may be over ten files. Once added, make sure the proxy restarts, and at this point you should see this error resolved.

Redshift and SSL

Redshift by default connects via TLS and requires TLS including CA validation. When connecting to the Heimdall proxy for Redshift, it is likely that initially, the self-signed certificate will be presented, AND TLS will need to be enabled. To resolve this, make sure the Redshift driver is updated, and use the sslmode option to "prefer" for testing. For production, it is advised that a correct and valid TLS certificate be loaded via the admin->certificate tab, and configured in the proxy tls settings. Older Redshift drivers will not accept the prefer setting.

Redshift and Heimdall

The Heimdall Database proxy is as of June 26, 2023, the only "Redshift Ready" proxy supported as a Redshift Ready product, and we will continue working with the Redshift team to improve compatibility and performance.

Active Directory Password & Group Synchronization (DRAFT)

```
CREATE OR REPLACE PROCEDURE heimdall.sync_user(username VARCHAR, password VARCHAR, ldapgroups VARCHAR(max))
    LANGUAGE plpgsql
AS $$

DECLARE
    -- configuration variables
    current_schema_name TEXT;
    create_roles_if_missing BOOLEAN;
    lookupTableName TEXT;
    lookupTableKey TEXT; -- ldap_group
    lookupTableValue TEXT; -- role

    -- logic variables
    sql VARCHAR(max);
    groupname RECORD;
    has_rows BOOLEAN;
    lookup_table_exists BOOLEAN;
    lookup_table_has_rows BOOLEAN;
    user_exists BOOLEAN;
    group_exists BOOLEAN;

BEGIN
    -- set these variables to match your config
    current_schema_name := 'heimdall';   -- where the sync_user and lookup table live
    create_roles_if_missing := false; -- set to true to create roles when they don't exist
    lookupTableName := 'lookup_table';
    lookupTableKey := 'ldap_group';
    lookupTableValue := 'db_group';

    EXECUTE 'CREATE TEMPORARY TABLE grouplist (g varchar(max))';
    EXECUTE 'CREATE TEMPORARY TABLE finalgroups (g varchar(max))';

    CREATE TEMPORARY TABLE numbers AS (
      SELECT ROW_NUMBER() OVER() AS n
      FROM
      (
        SELECT 1
        FROM pg_catalog.pg_attribute
        LIMIT 100 -- you can adjust this value to fit the maximum number of elements in your CSV strings
      ) a
    );

    INSERT INTO grouplist
    SELECT TRIM('"' FROM TRIM(SPLIT_PART(ldapgroups, ',', n::integer)))
    FROM numbers
    WHERE n <= REGEXP_COUNT(ldapgroups, ',') + 1;

    -- find out if we have a lookup table to map groups via
    SELECT EXISTS (SELECT 1 FROM pg_tables WHERE schemaname = current_schema_name AND tablename=lookupTableName) INTO lookup_table_e

    IF lookup_table_exists THEN
       -- check whether lookup_table has any rows. If not there's no reason to process queries related to it.
        EXECUTE 'SELECT COUNT(*) FROM ' || current_schema_name || '.' || lookupTableName || ' lt LIMIT 1' INTO lookup_table_has_rows
    END IF;

    IF lookup_table_exists AND lookup_table_has_rows THEN
        RAISE NOTICE '%.sync_user: lookup table found', current_schema_name;
        -- we have a lookup table, use it to map the inbound groups to local groups
        sql := 'insert into finalgroups select ' || lookupTableValue || ' from ' || current_schema_name || '.' || lookupTableName ||
        EXECUTE sql;
        -- and find the unmapped groups
        sql := 'insert into finalgroups select g from ' || current_schema_name || '.' || lookupTableName || ' lt, grouplist gl where
        EXECUTE sql;
    ELSE
    -- no lookup group, so just copy the grouplist into the final groups table
        INSERT INTO finalgroups SELECT g FROM grouplist;
    END IF;

    -- at this point, finalgroups should have the complete grouplist of what groups we want for the user
        SELECT EXISTS (SELECT 1 FROM pg_user WHERE usename = username) INTO user_exists;

        -- sync user and password
    IF user_exists THEN
        sql := 'ALTER USER ' || quote_ident(username) || ' WITH PASSWORD ''' || password || ''';';
        EXECUTE sql;
        sql := 'ALTER USER ' || quote_ident(username) || ' WITH PASSWORD ''' || 'REDACTED' || ''';';
        RAISE NOTICE '%.sync_user: % updated', current_schema_name, sql;
          -- now find user's groups that are not in the finalgroup table
        FOR groupname IN SELECT groname FROM pg_group JOIN pg_user ON usesysid = ANY(grolist)
```

```
    WHERE usename = username LOOP
        -- this select is not yet working properly
        SELECT EXISTS (SELECT 1 FROM finalgroups WHERE groupname.groname = ANY(SELECT g FROM finalgroups)) INTO has_rows;
        IF NOT has_rows THEN
            RAISE NOTICE '%.sync_user: dropping % from user %', current_schema_name, groupname.groname, username;
            -- now revoke membership of the group from the user
            sql = 'ALTER GROUP '|| groupname.groname ||' DROP USER '|| username || ';';
            EXECUTE sql;
            RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
        END IF;
    END LOOP;
ELSE
    sql := 'CREATE USER ' || quote_ident(username) || ' WITH PASSWORD ''' || password || ''';';
    EXECUTE sql;
    sql := 'CREATE USER ' || quote_ident(username) || ' WITH PASSWORD ''' || 'REDACTED' || ''';';
    RAISE NOTICE '%.sync_user: % created', current_schema_name, sql;
    END IF;

    FOR groupname IN SELECT g FROM finalgroups LOOP
        SELECT EXISTS (SELECT groname FROM pg_group WHERE groname ILIKE groupname.g) INTO group_exists;
        RAISE NOTICE 'group: %, group_exists: %', groupname.g, group_exists;
    IF (NOT group_exists) AND create_roles_if_missing THEN
        sql := 'CREATE GROUP ' || groupname.g;
        EXECUTE sql;
        RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
        END IF;
        -- add group to user
        IF group_exists OR create_roles_if_missing THEN
            sql = 'ALTER GROUP '|| groupname.g ||' ADD USER '|| username || ';';
            EXECUTE sql;
            RAISE NOTICE '%.sync_user: %', current_schema_name, sql;
        END IF;
    END LOOP;

    -- make sure we drop these to prevent reuse issues on the same connection
    DROP TABLE numbers;
    DROP TABLE grouplist;
    DROP TABLE finalgroups;

RETURN;
END;

$$
;
```

Active Directory Password & Group Synchronization extra options

By default, a group with the same name is created in the database for each extracted ldap group. By creating the table below for each key found (ldap_group), a role with the specified name in the value (role) will be created instead. Composite Primary Key allows to map more than one role to particular ldap_group; If 'role' is empty then no role is created and granted to user for this particular ldap_group.

```
CREATE TABLE IF NOT EXISTS heimdall.lookup_table (
    ldap_group varchar(255),
    db_group varchar(64),
    PRIMARY KEY (ldap_group, db_group)
)
```

Pivotal Greenplum Specific Information

- When using the Async feature, set a connection property of "initSQLAsync" to the value of "set optimizer = off". This will disable the GPOrca optimizer when async is used, which can reduce the overhead per insert significantly.
- If using Postgres offload, ensure that both Greenplum and Postgres use identical schema and user configurations.
- If using foreign data wrappers from Postgres to Greenplum, add a rule as follows:
    - Regex: SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
    - In-Trans: Checked
    - Action: Transform
    - Parameter: target
    - Value: SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

This will allow the repeatable read transaction isolation level (not supported by GP) to be translated to serializeable, which will enable the Postgres FDW to operate properly

- Ensure that the proper modules for Postgres are loaded in the admin->modules tab.
- When configuring load balancing, specify the failover script (provided), and set the secondary node as being offline. Since Greenplum doesn't allow monitoring of the secondary node, it is up to the script to trigger promotion and enabling the node in the LB configuration.

PG Notify support

This is not currently supported with Greenplum vs. current Postgres sofware versions.

Trigger Invalidation with Greenplum

While Greenplum supports triggers in a limited sense, the triggers do not work when writing to a table from the data segments. As such, trigger invalidation is not yet working with Greenplum (see below for an alternative).

Stored Procedure based Invalidation with Greenplum

As a replacement for trigger based invalidation, a stored procedure can be used that leverages the "Trigger invalidation" mechanism to provide invalidation for Greenplum. When used, the stored procedure can be called after each data load to notifiy Heimdall of new data.

Connection properties to use:

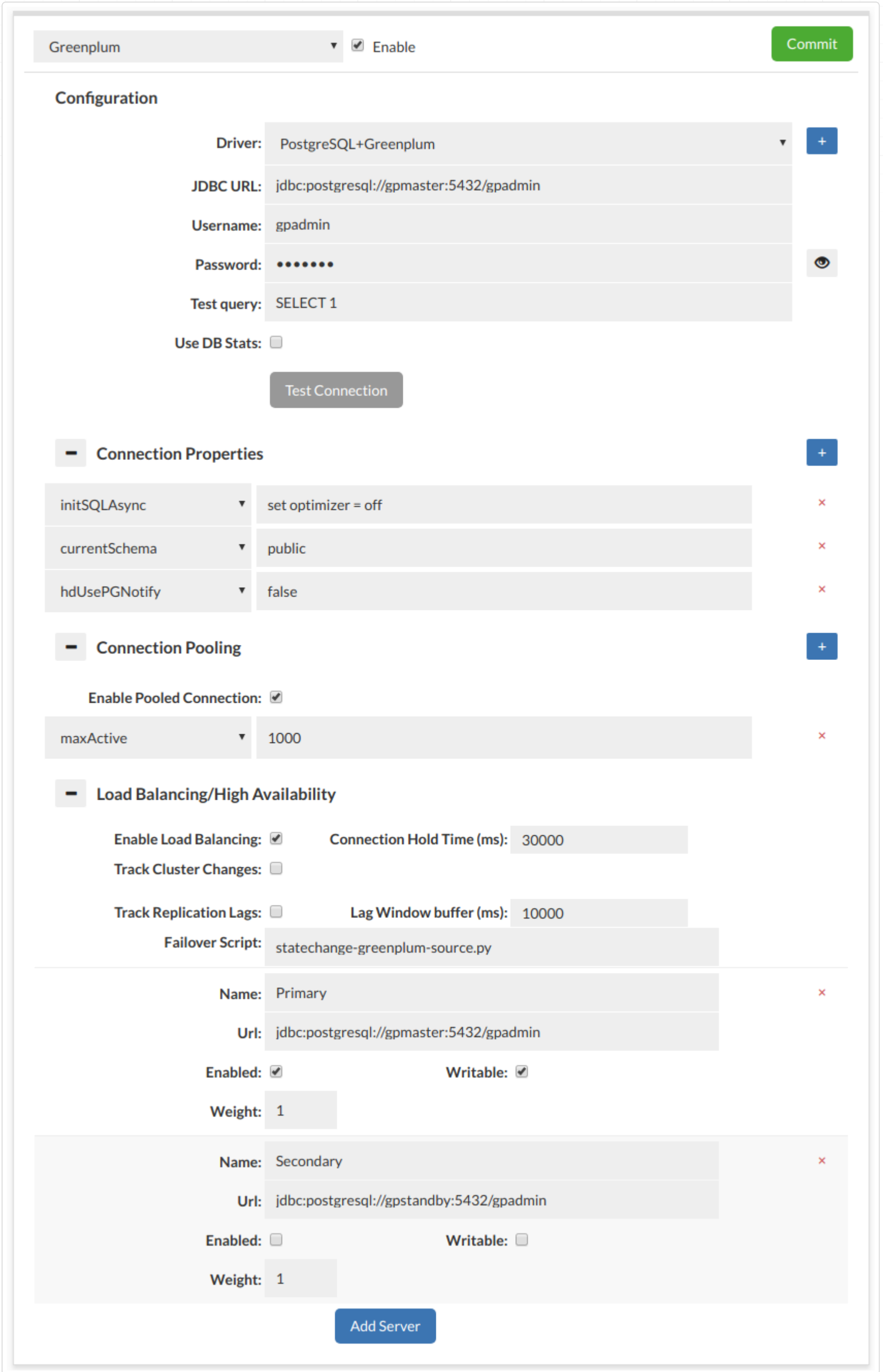
- dbTriggerTracking=true
- dbChanges=select ts,name from heimdall.tableupdates where ts > @ts
- dbTimeMs=select (date_part('epoch', now())*1000)::bigint

Greenplum configuration:

```
CREATE SCHEMA heimdall;

CREATE TABLE heimdall.tableupdates (
name text PRIMARY KEY,
ts bigint not null );

CREATE OR REPLACE FUNCTION heimdall.changed(text) returns void AS $$
BEGIN
  LOOP
    UPDATE heimdall.tableupdates SET ts=(date_part('epoch', now())*1000)::bigint where name=$1;
    IF found THEN
      RETURN;
    END IF;
    BEGIN
       INSERT INTO heimdall.tableupdates VALUES ($1, (date_part('epoch', now())*1000)::bigint);
       RETURN;
    EXCEPTION WHEN unique_violation THEN
    END;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE plpgsql;
```

Source Configuration

Below is a screenshot of the default recommended data source configuration for a baseline Greenplum cluster. Details are explained below:



- Connection Properties
    - **initSQLAsync**: set optimizer = off This is to disable GPOrca optimization when using the Async insert, as this slows down processing for most inserts
    - **currentSchema**: public This can be adjusted, but sets the default schema used when connecting
    - **hdUsePGNotify**: false Postgres supports
- Pooled Connections
    - **Enabled** to enable connection pooling
    - **maxActive** to set the maximum number of connections allowed in the pool
- Load Balancing/High Availablility
    - **Enabled**
    - **Failover Script**: statechange-greenplum-source.py

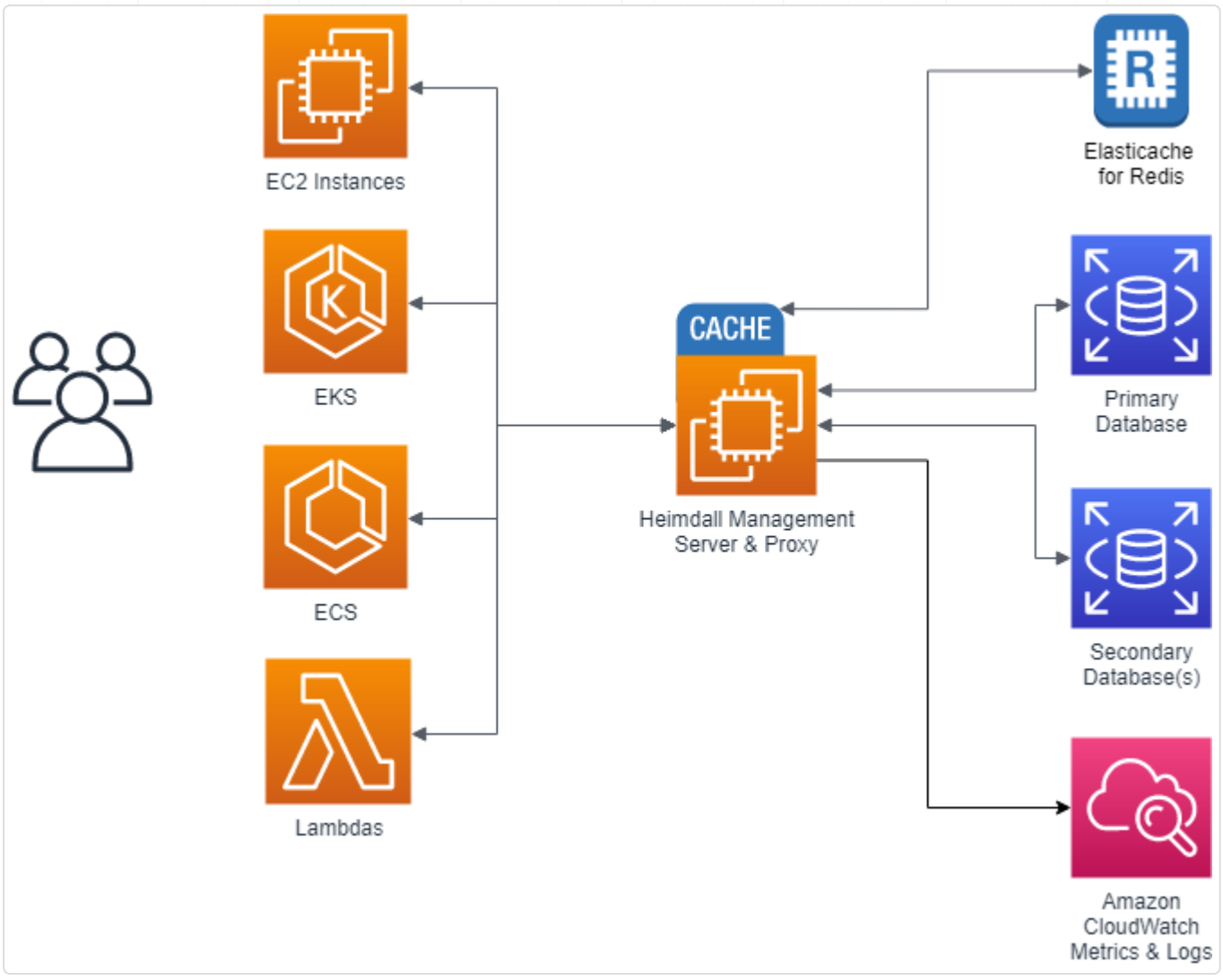
  Set your master node as enabled and writeable, and the standby as disabled, and not writeable. After a failover, please ensure that the standby is reset to be in sync and available as a standby per: https://gpdb.docs.pivotal.io/580/admin_guide/highavail/topics/g-recovering-a-failed-master.html

Amazon AWS Specific Information

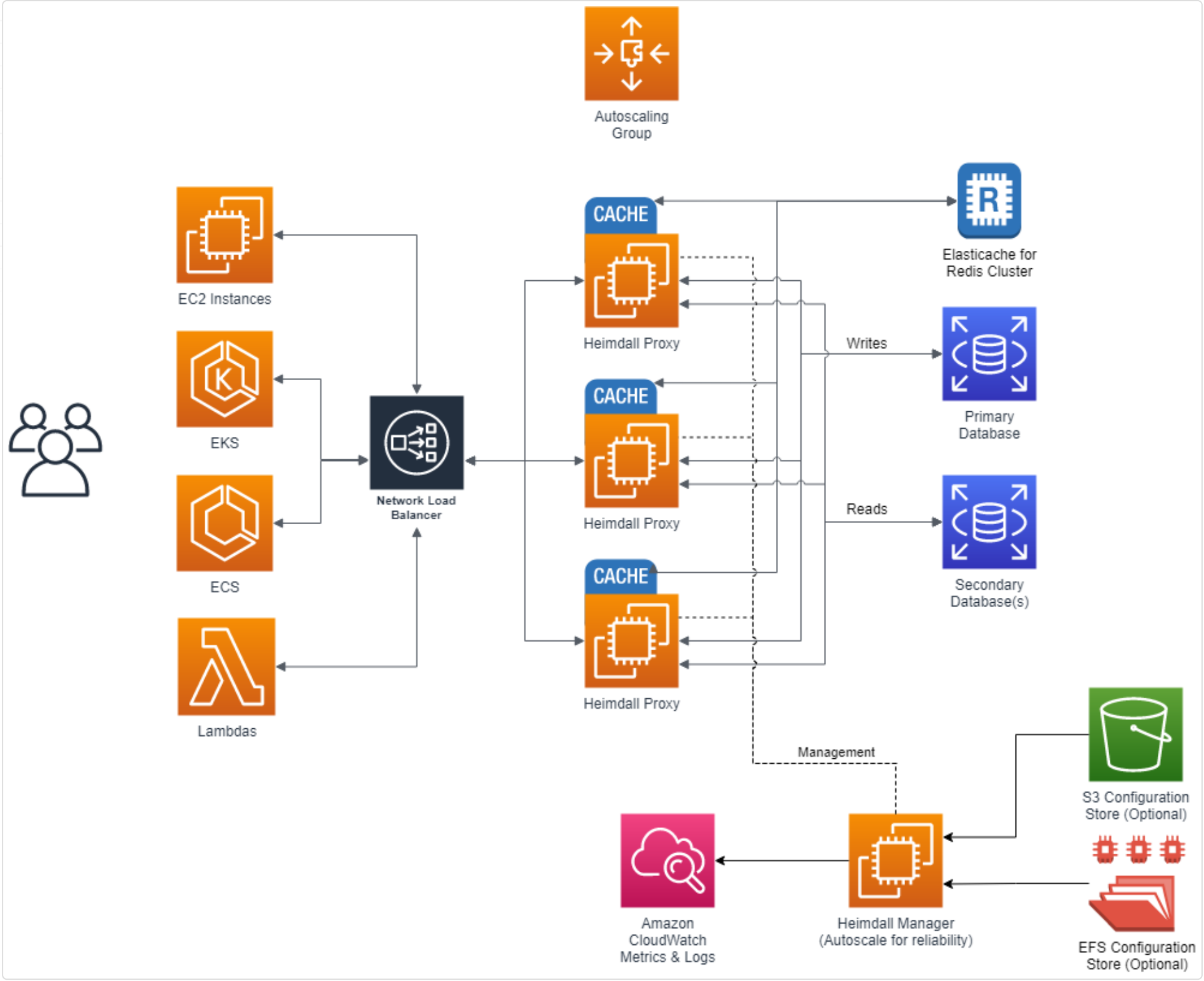
Please note--this section is organized in the form of a quick-start guide. If all the steps performed are done in-order, it will help to ensure success for an AWS deployment.

Typical Deployment

In most situations, customers go through two stages of deployment for Heimdall, a single server test deployment for POC installs and QA:



Here, Heimdall exists only on one server, and both the management server and the proxy exist together. Obviously, this results in a single point of failure, but this allows easy and fast proof of concept installs to occur. Once the idea of using Heimdall is validated, a more robust configuration is deployed:



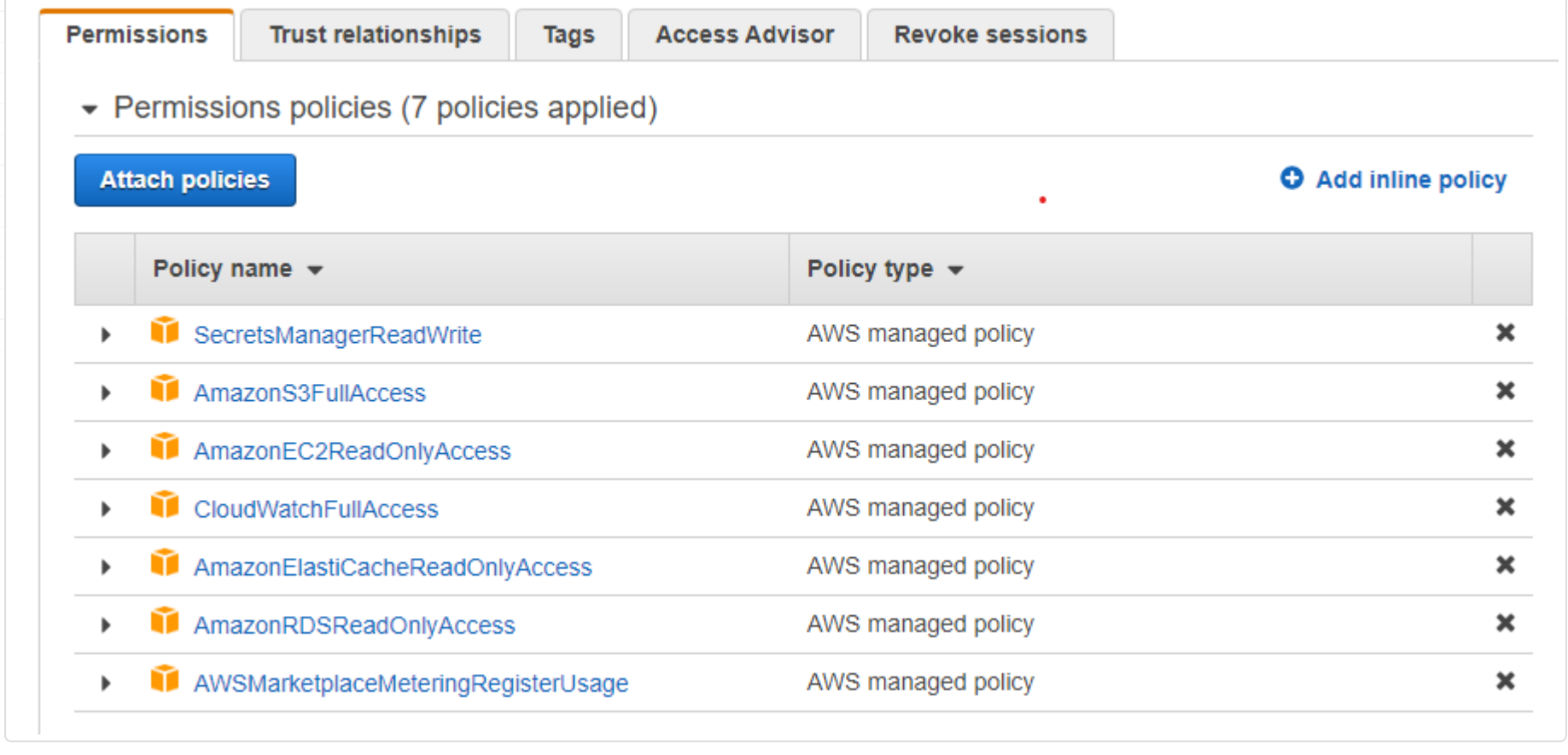
Here, the proxy nodes are deployed into one or more Availability zones (to match the applicaton server AZs) with autoscaling, and the central manager exists on it's own server. As the proxies operate as an independent data plane, failure of the management server is non-disruptive, so most customers choose to simply deploy a single management server. If a redundant management server is desired however, this can be done as well, with the obvious extra cost. In the single management server model, the management server can sit behind an autoscaling group, and keep it's configuration files on EFS, or pre-populate the configurations from S3, as desired.

RDS Support

Heimdall provides support for all MySQL, Postgres, and SQL Server RDS types, including Aurora and Aurora for MySQL, including all actively supported versions of those engines.

Creating an IAM Role

In order to simplify the configuration for Amazon Web Services, the system supports reading the RDS and Elasticache configuration directly. In order to use this feature, one must configure an IAM role with access to the appropriate rights, as shown, then the option for "AWS Configuration" will become available in the Wizard. Without the IAM credentials, a manual configuration is still possible.



The permissions are for:

- **AmazonEC2ReadOnlyAccess**: To detect the local region being run in, and the security groups to validate proxy configurations
- **AmazonRDSReadOnlyAccess**: To populate database configurations and to enhance auto-reconfiguration of clusters on a configuration change or failover
- **AmazonElastiCacheReadOnlyAccess**: To populate the available caches
- **CloudWatchFullAccess**: When using cloudwatch monitoring, to build log groups and to report data (recommended, but not required)
- **SecretsManagerReadWrite**: If the hdSecretKey option is to be used, this permission enables the manager to pull and push configuration information into a secret.

Note: In a production environment, it's recommended to follow the principle of least privilege and grant only the specific permissions needed by the manager or the associated entity to perform their required tasks. This can be achieved by creating custom IAM policies with more granular permissions based on the specific resources needed and actions required. It is often easier to let Heimdall create resources such as secrets and cloudwatch log groups, and then lock down access to those needed vs. create them upfront.

For more information on IAM and IAM best practices please see Security best practices in IAM.

Note, for any given API access to work, in addition to the IAM role, the api endpoint needs to be accessible as well. To test if the api endpoints are accessible from a given client, you can use the following:

```
wget https://ec2.us-east-1.amazonaws.com/
wget https://rds.us-east-1.amazonaws.com/
wget https://elasticache.us-east-1.amazonaws.com/ (will return a 404, this is normal)
wget https://monitoring.us-east-1.amazonaws.com/ (will return a 404, this is normal)
wget https://secretsmanager.us-east-1.amazonaws.com/ (will return a 404, this is normal)
```

If these return anything other than a timeout, they are likely accessible. Otherwise, the api endpoint likely needs to be enabled in the VPC, due to routing restrictions.

Configuring an Elasticache for Redis Instance

If Elasticache will be used, it is recommended that the instance be setup before configuring Heimdall, as this allows the Elasticache configuration to be auto-detected. When configuring Elasticache, make sure at least two nodes are configured for redundancy if desired (clustered or not). For initial configuration on a POC, it is generally recommended that the following options be selected:

1. Cluster mode disabled (but cluster mode is supported);
2. Multi-AZ enabled

3. Size of m6g.large similar size instance. If load is low, then t4g.medium instances would be reasonable--the size can be adjusted if needed due to excessive bandwidth use after the inital install;
4. At least one replica for redundancy;
5. AZ selection should map to the availability zones that are expected to be used by the proxy layer and/or application servers.
6. Security options such as enabling encryption in transit and access control is optional--both are supported, but need to be set manually after the wizard is run to create the initial virtual database.

After configuring the Elasticace for Redis install, it is important to configure a parameter group, and set the parameter "notify-keyspace-events" to the value "AE". This will allow the system to track objects that are added and removed from the cache automatically, which helps prevent L2 cache misses. In other Redis types, this parameter can be set dynamically at runtime, but in ElastiCache, this can only be set via the parameter group. Failure to do this will simply reduce the performance of the system when there are cache misses. The configuration should appear as below:

Parameter Group: redis-keyspace-events

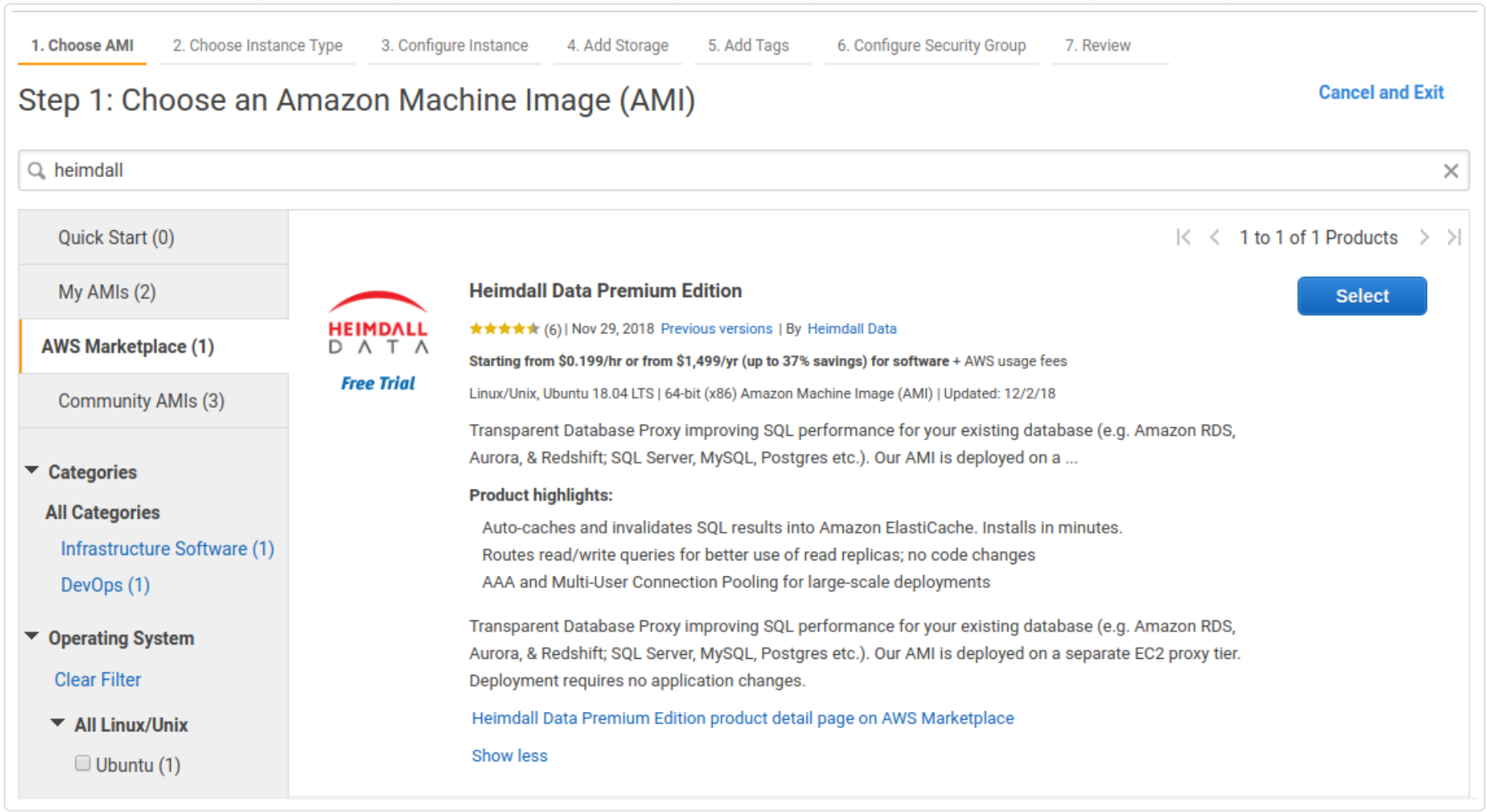
Viewing: All Parameters | Edit Parameters | 21 to 30 of 134 parameters

Name	Allowed Values	Is Modifiable	Instance Class	Value	Source	Type	Change Type	Description
maxclients	1-65000	No	ALL	65000	system	integer	requires-reboot	The maximum number of Redis clients.
maxmemory-policy	volatile-lru,allkeys-lru,volatile-random,allkeys-random,volatile-ttl,noeviction	Yes	ALL	volatile-lru	user	string	immediate	Max memory policy.
maxmemory-samples	1-	Yes	ALL	3	user	integer	immediate	Max memory samples.
min-slaves-max-lag	0-	Yes	ALL	10	user	integer	immediate	Maximum number of seconds within which the master must receive a ping from a slave to take writes. Use this parameter together with min-slaves-to-write to regulate when the master stops accepting writes. Setting this value to 0 means the master always takes writes.
min-slaves-to-write	0-	Yes	ALL	0	user	integer	immediate	Number of slaves that must be connected in order for master to take writes. Use this parameter together with min-slaves-max-lag to regulate when the master stops accepting writes. Setting this to 0 means the master always takes writes.
notify-keyspace-events		Yes	ALL	AE	user	string	immediate	The keyspace events for Redis to notify Pub/Sub clients about. By default all notifications are disabled

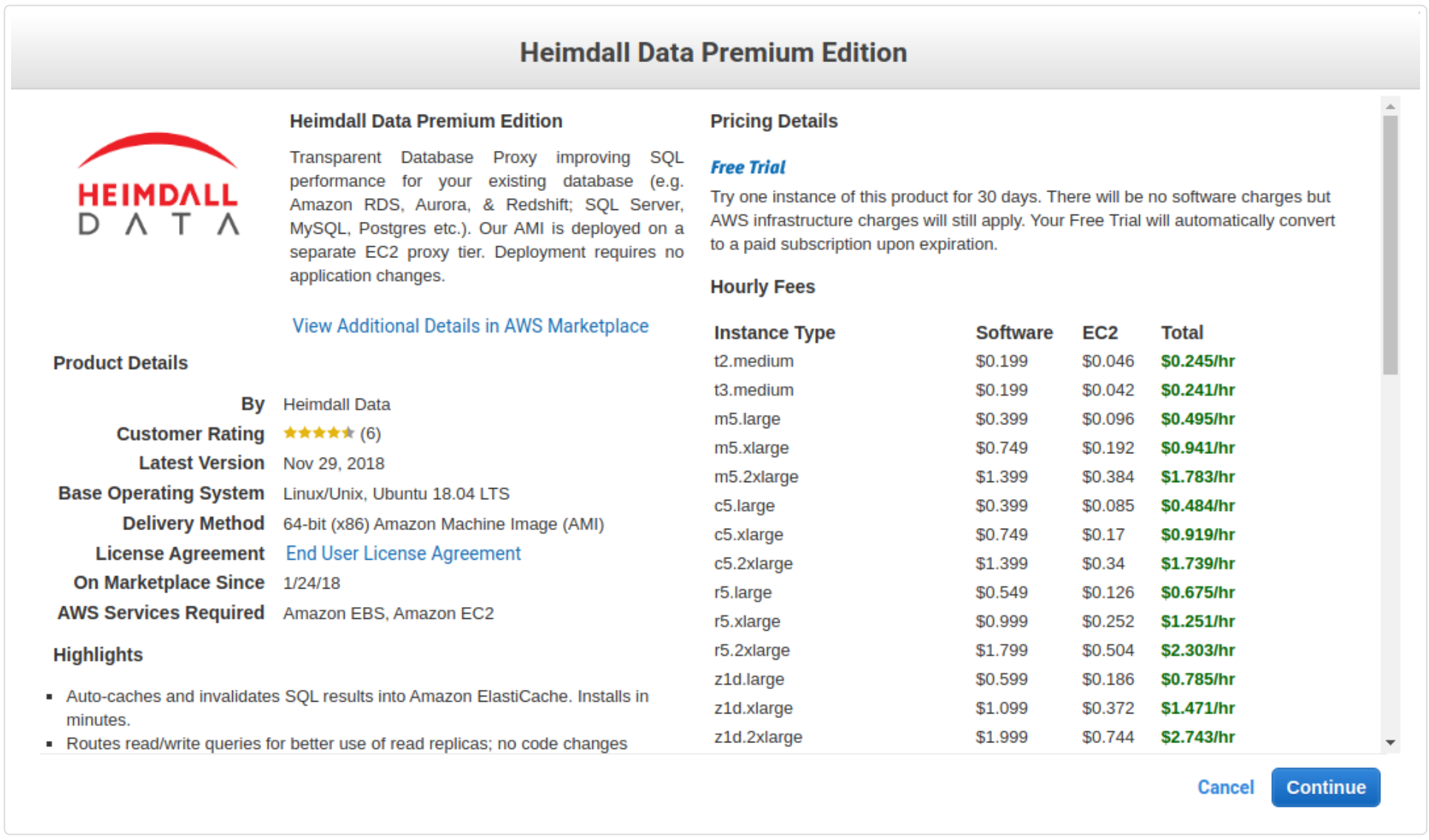
AWS Marketplace Install

Heimdall can be be easily started using the AWS marketplace. During this startup, the image will download the newest release version of Heimdall. If the security groups are such that the download can not complete, then an older version of Heimdall will be used (from the instance creation time).

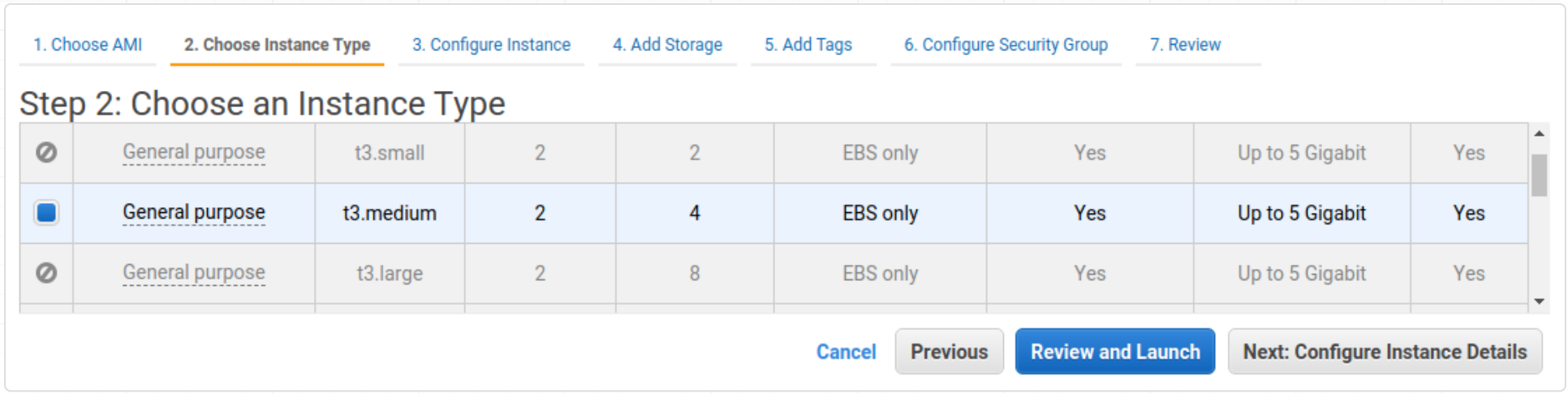
First, in the EC2, select launch instance, select the aws marketplace option on the left, and search for "Heimdall Standard Edition" or "Heimdall Enterprise Edition" with 24/7 support, then select:



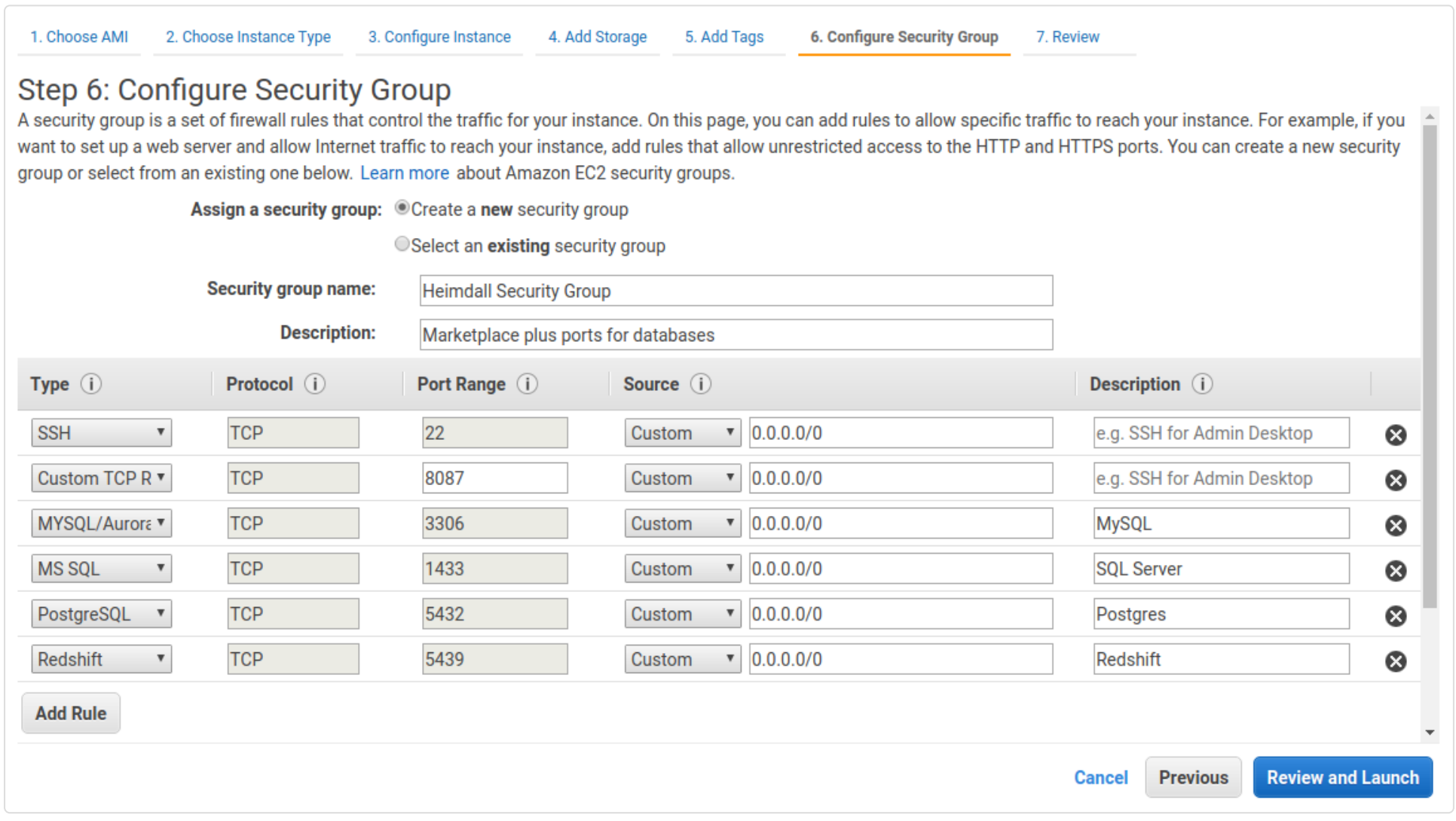
Then, continue with Heimdall:



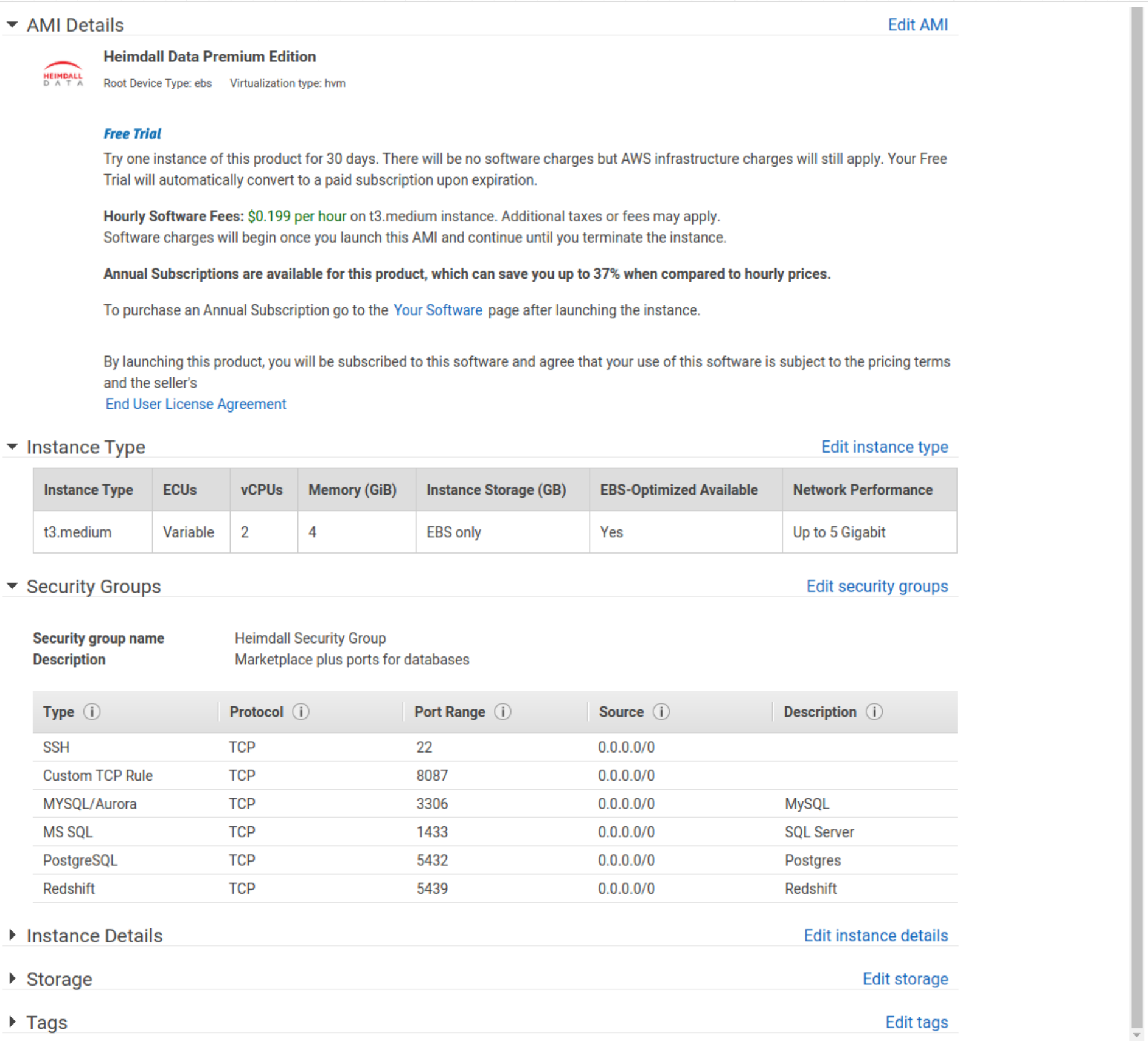
Select the desired instance types--The marketplace offering supports a variety of appropriate instances, with larger core counts supported in the Enterprise edition. Note on sizing: We generally recommend in initial core count for proxy capacity to be 1/4 that of the database behind Heimdall, so if the database has 8 cores, we would recommend starting with a 2 core proxy. In the event the cloud formation template was used, then you can split these cores across multiple proxies and use auto-scaling to size from there. In general, the c5 or Graviton 2 instances are recommended. For very high cache hit rates (over 90%), the c5n instance type may be appropriate, as the network bandwidth will be saturated before the CPU will be. Sizes can be adjusted once testing is complete and a better idea of the load ratio is understood.



Continue through the screens, ensuring that the security group configuration opens ports as needed for proxy ports (please update the source subnets to match your local networks, it is best practice to never open ports to the internet in general):



Finally, review and launch:



Next, with the EC2 instance online, you can bind the IAM Role created above to the EC2 instance, by right mouse-clicking on the instance, and under security, select modify IAM role. The following screen will allow you to select the role created. This enables autodetection of RDS and Elasticache resources, and other AWS integrations.

Once the instance is online and the IAM role is configured, connect to the instance on port 8087--there is a login help page providing instructions for the initial login. Please configure the instance using the wizard for the best results.. Nearly every manual configuration will have a fault, often resulting in support calls.

Advanced Marketplace Install

As Heimdall can run on multiple servers, at initialization, an instance can accept as user-data configuration options that will control how the instance runs, and if it should operate as a proxy or server (onl`y). When set to such a mode, the instance will attempt to tune itself for the role in question, i.e. use the entire instance's memory vs. allowing memory to be used. To provide these options, the user-data should be a script that generates a file "/etc/heimdall.conf". For more details see heimdall.conf configuration.

Please refer to AWS CloudFormation Template page for more information on using a template to create an autoscaling group, which automates the process of creating proxy-only instances, and provides failover resiliency as well.

CloudWatch Metrics and Logs

In each VDB, under logging, an option is available for AWS CloudWatch. If enabled and CloudWatch access is enabled in the system's IAM role, it will start logging a variety of metrics into CloudWatch under the "Heimdall" namespace, and the vdb logs will also be logged into CloudWatch. The metrics include:

- DB Query Rate
- DB Query Time
- Avg Response Time
- DB Read Percent
- DB Transaction Percent
- Cache Hit Percent

Please note that additional charges may be incurred due to metrics and logging, in particular if debug logs under high volume are logged into CloudWatch.

When using Heimdall on private IP ranges, please see https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/cloudwatch-logs-and-interface-VPC.html for more information to enable cloudwatch within the vpc.

TLS certificates for RDS

In the new builds starting in July 2021, the file rds-combined-ca-bundle.pem will be installed in /etc/ssl/certs and can be used to validate RDS CA certificates as appropriate and needed.

Aurora/RDS Load Balancing Behaviors

In an AWS RDS (including Aurora) environment, Heimdall has taken a flexible approach to allowing a cluster to be defined and configured. When used with RDS and the proper IAM role is attached to the management server(s), the primary URL hostname defined in the data source (outside of the LB) is used to identify a particular cluster. The cluster nodes are then probed in order to build the LB configuration. This definition is re-done at the initial cluster definition (if cluster tracking is enabled) every 30 seconds during normal processing and every 1 second when a node is in a "failure" mode, and also every time the data source configuration is committed.

When a cluster definition poll is done, the driver specified for the data source is used to create a new set of nodes. The nodes can be defined as "listener" (SQL Server only), "writer" node, "replica" for a read replica, or "reader" which includes both replicas and writers. The reader and writer URL's are then used to populate the nodes as appropriate based on the actual cluster state.

Please see the driver configuration page for information on each variable.

The goal of the Heimdall configuration is to bypass DNS resolution when possible for endpoints, which can slow down the failover process, while allowing the underlying drivers to support automatic detection and failover of the nodes that are acceptable for reader vs. writer roles. Through the default configurations provided by Heimdall, sub-second failovers can be achieved in most cases, and via the templates, custom configurations can be created to adjust the actual behavior desired.

For example, with the Postgres drivers, the default writerUrl template is:

```
jdbc:postgresql://${readers}/${database}?targetServerType=master
```

Here, the ${readers} is used, as the Postgres driver supports an ordered list, and the writer nodes will be included first in this list. With the targetServerType of master included, it will connect only to a master in the list, but on a failure, it will fail as quickly as possible to a new writer as soon as it is promoted. This allows the endpoint hostname resolution for Aurora to be bypassed, while providing the best possible availability.

And for the reader it is:

```
jdbc:postgresql://${readers}/${database}?targetServerType=preferSecondary&loadBalanceHosts=true&readOnly=true
```

Again, in this case, the driver's capability to load balance is being used to create a single pool of read connections, which will avoid using the writer node if possible.

An alternate writer URL could be used of:

```
jdbc:postgresql://${writer}/${database}
```

This wouldn't leverage the built-in failover capability, but would rely on Heimdall to detect a failure, and poll the AWS API's to find the new writer.

If each host in a number of readers is desired to have it's own node entry, along with graphs on the dashboard, and the write nodes should not be used for reading, the following alternate URL could be used:

```
jdbc:postgresql://${reader}/${database}?readOnly=true
```

This bypasses the postgres driver built-in LB capability, and will create a node for LB in Heimdall for each reader.

All built-in database types include a pre-defined set of reader and writer URL's except for Oracle, due to the complex nature of Oracle URL's for many environments. Older installs prior to this template function will have defaults included when updated to the code that supports this functionality.

Notes on behavior:

- ${writeEndpoint} and ${readEndpoint} can be used for Aurora definitions, and will point to the writer (primary) and reader endpoints as appropriate. This allows use of the AWS managed DNS targets if desired. DNS resolution on these will be cached for five seconds, and due to potential update timings, this could result in a connection intended for a writer to point to a reader instead. This is a side-effect of connection pooling, DNS caching, and the update timing of the DNS entry. This behavior can occur with other applications not using Heimdall as well.
- ${reader} will include the nodes that are expanded from ${writers}, but will set the weight of 1 vs. 10 for any pure reader node.
- ${readers} can be used in the writer URL definition to expand to the entire list of writers and readers for drivers that will use an ordered list to identify what is a writer and what is a reader. Other variables can only be used in the proper context, i.e. ${writer} for a writeUrl, or ${reader} for a readUrl.
- ${listener} is used only for SQL Server, and will resolve to the listener IP for a particular always-on cluster for the writerUrl, and will map to the writer instance in the cluster.

Things AWS doesn't tell you (or hides) that you need to know!

- Each EC2 instance type has a documented burstable bandwidth level, but it has an undocumented steady-state bandwidth level that often significantly lower than the burstable limit. One website has prepared a cheat-sheet that provides the numbers for many instance types.
- ELB including NLB (despite documentation to the contrary) may require pre-warming if you expect highly burstable traffic.
- RDS (but not Aurora) instances may get alerts of "Write query was sent to read-only server or transaction. Please see log for details." This error may be happening during your daily backup window, as the database may transition from writable to read-only during this window for a period of time. Aurora instances do not freeze this way due to the way they handle block IO.

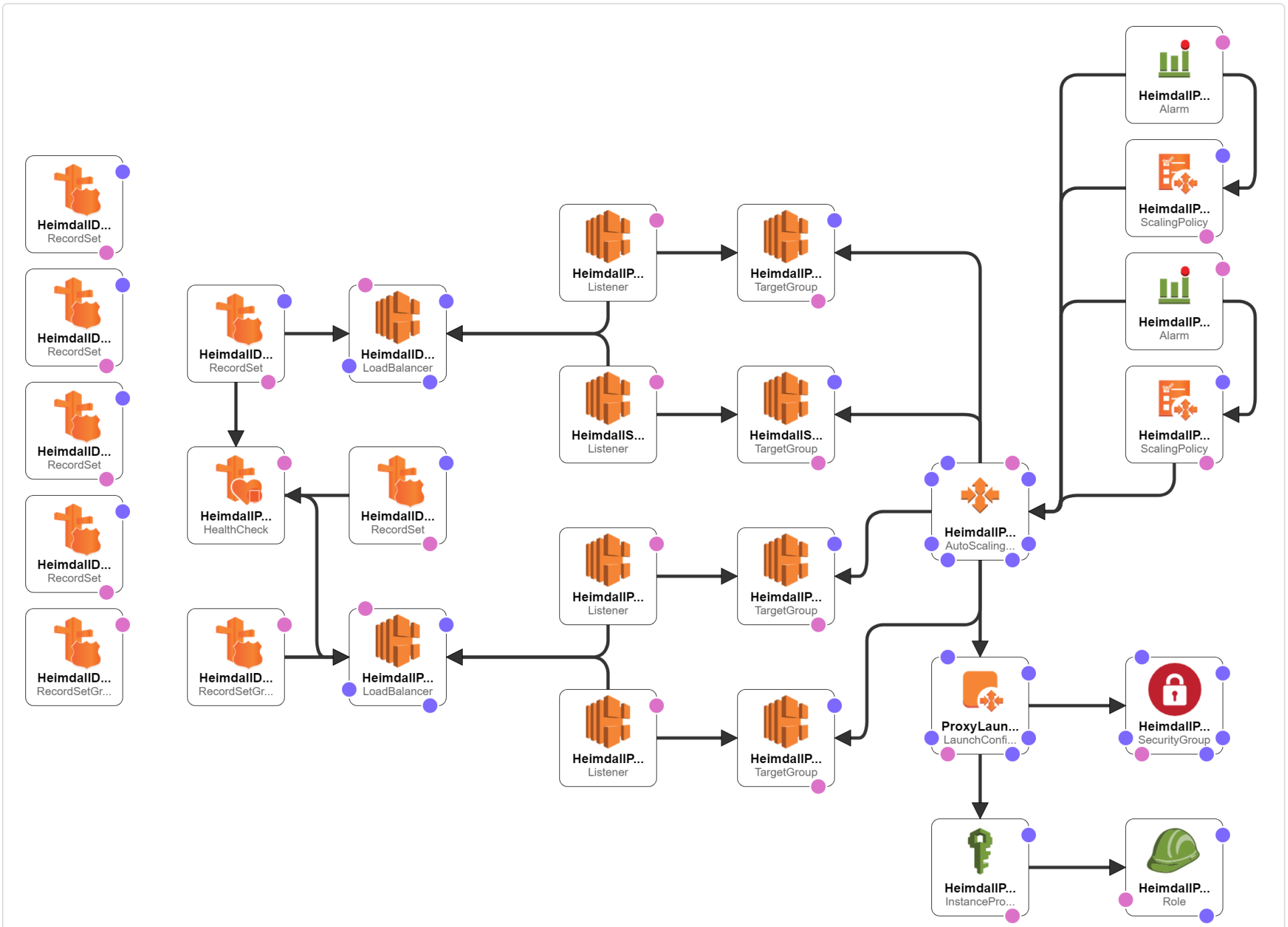
Amazon AWS CloudFormation Template

In order to provide a redundant proxy configuration, the following guide can be followed.

- Deploy a Heimdall Central Manager via the AWS Marketplace. This instance doesn't need to be particularly large, many customers use a t3.medium;
- Ensure the application works appropriately through a proxy instance hosted on the management server;
- Once the application is validated, you can un-check the proxy option "Local proxy" which will terminate the proxy on this management server;
- To fully dedicate the memory of the management server to management activities, follow the heimdall.conf instructions here;
- Deploy a cluster of proxy-only nodes using the template located at: Template.

Note: Support for this template is exclusively provided for Heimdall Enterprise Edition customers as customers that require this level of redundancy also typically require 24/7 support. Heimdall offers private offer pricing to customers on a negotiated basis if needed. The template is designed to pull the newest Enterprise (ARM) image in any region automatically. If a custom AMI ID is desired, please create a SSM parameter, and provide the AMI ID desired as the value of the parameter. Use the parameter name as the AMI Alias in the CF template, and on deployment, it will resolve the AMI ID per the SSM parameter.

The resources created by this template are:



- One or Two Network Load Balancers with Listener billable Note: We support both database protocol load balancing as well as distributing traffic via DNS queries. In the DNS case, the NLB costs will be significantly less, but overall traffic distribution may not be as even across nodes.
- Autoscaling configuration with Target Group
- Scale up and down policies with alarms
- Launch Configuration with instance policy and IAM role
- Security group for proxy EC2 instances

- (Optional) Route53 records to monitor the proxies and provide easy to use hostnames to access the proxies, including with geolocation capability. billable

In the case of the DNS based NLB setup, the template will also configure: * A record pointing to the NLB target hostname * An NS record that points to the A record pointing to the NLB target hostname

It depends on: * Existing VPC & Subnets * An existing Heimdall Central Manager * SSH Key

All resources will be named based on the stack name as appropriate, and their creation can be reversed by deleting the stack.

Heimdall assumes no responsibility for the charges that may be charged by AWS through the use of this template, as in particular, outside of the EC2 instances, the network load balancer may incur charges, as explained "here".

Template Options

For all template options, the following are required:

- ProxyMinimum and ProxyMaximum set to numeric values
- HeimdallProxyVPC configured, and HeimdallProxyAvailablityZones and HeimdallPublicProxySubnetList configured, with all parameters in alignment
- ProxyPublicIpAddress can be set to true if public routing will be used, but for single region deployments, this is typically set to false
- HeimdallProxyAMIID is set to the AMI alias for the Heimdall Enterprise (ARM) edition. Minor editing could be done to use Intel VM's however if need.
- InstanceTypeParameterProxy is set to an appropriately sized instance size
- Keyname for the SSH key to install on the system
- vdbName is configured
- hdHost is configured
- hdUser is configured. This could be the admin user OR the VDB secret key
- hdPassword is configured. This could be the admin password, or the VDB secret value
- ProxyPort is configured.

Note: The HeimdallProxyAMIId defaults to: /aws/service/marketplace/prod-6k3x2q5j7vg32/latest for the ARM Enterprise Edition. Other valid options are:

- /aws/service/marketplace/prod-upicfbvjlnfqo/latest for the ARM Enterprise Plus Edition
- /aws/service/marketplace/prod-ahm64zezggfhg/latest for the Intel Enterprise Edition
- /aws/service/marketplace/prod-qomddleky3adi/latest for the Intel Standard Edition

Please note that in production, we only support the Enterprise edition options for larger installs with autoscaling.

Single Region NLB Balanced Cluster

The first and simplest configuration that the template supports is one where NLB is used to load balance an autoscaling group of proxies, and the load balancer name is used to access the proxies in a single region. In the template parameters, to configure this, the following are needed:

LBType=nlb

The following options should be blanked to avoid unnecessary route53 configurations for this simple setup:

HostedZoneID, DNSZoneName, dbEndpointName

Multiple Region NLB Balanced Cluster

Like single region, but enable the route53 configuration items of "HostedZoneID" and "DNSZoneName", and deploy the CF template in multiple regions. The primary access hostname will be generated based on the vdb name, and all stacks should have a unique name, to ensure there are no conflicts in global resources like IAM roles.

Single OR Multi-Region DNS Balanced Cluster

This setup is designed to leverage DNS to balance traffic to proxy nodes vs. NLB. NLB is however still used to direct DNS queries to the proxies themselves, but due to the low volume of DNS queries vs. database traffic, this should result in a very low bill for the NLB. For this configuration to be effective, the following should be noted:

1. There should be a large number of application servers, or a high amount of thrashing of clients accessing the database/proxy. This is to ensure that caching of DNS results doesn't result in unbalanced traffic load on the proxies;
2. If possible, the application servers should be configured to periodically close and open new connections, AND to re-query DNS periodically. This is very similar to guidance on using RDS to access a read-only URL to access multiple reader nodes.
3. A public NLB will be created on a public IP to allow access to DNS and HTTP queries. The HTTP queries will be coming from Route53 health check nodes, and will simply be querying for the URL /status, which is exposed by the proxies for health monitoring. This NLB needs to be public for these functions to work.

To use this configuration:

1. In the Heimdall VDB configuration, under the advanced options, the auto-scale mode checkbox should be checked
2. DNS port set to 54 (53 may work in some environments, but not all so should be avoided)
3. Health Check Port should be set to 80, along with a reasonably low health check interval (5 is suggested)
4. The Proxy Redirect Name parameter in Advanced Options should be configured to either Proxy Private IP or AWS Public IP. If using a multi-region AWS configuration, unless all private IP's are configured across VPCs to be routable, then use the AWS Public IP
5. If a global RDS deployment is in place, make sure to check the "Use Response Metrics" option in the data source LB section to preferentially route to the closest database nodes for reads
6. Deploy the template in all regions proxies are desired

In the template, make sure the following are set correctly:

LBType=dns HostedZoneID and DNSZoneName are configured

Optional Variations

If Route53 configurations are used, then the dbEndpointName can be configured to provide a "failover" name to be used if the proxy environment fails. This applies to both NLB and DNS configurations.

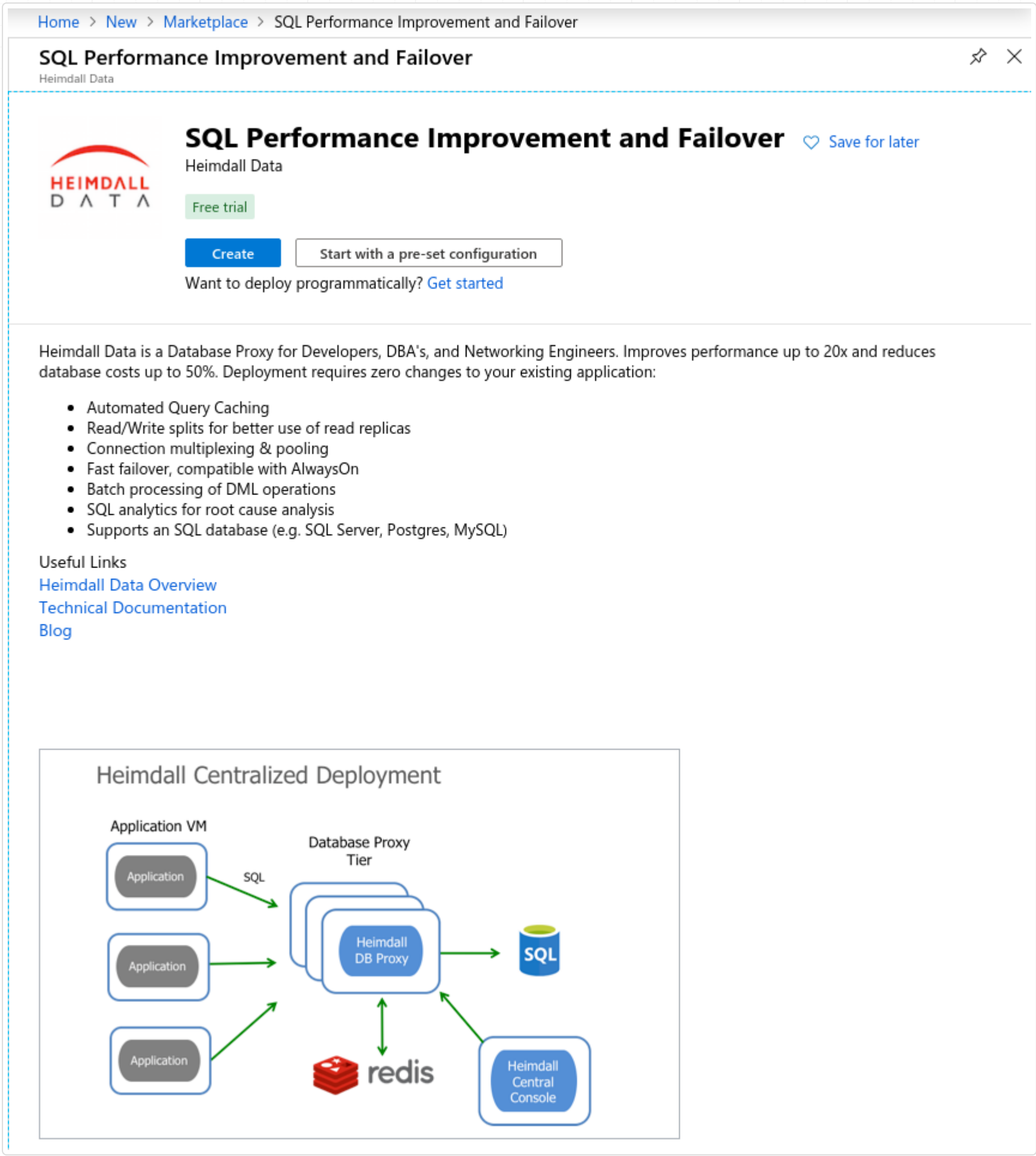
If a customer has two endpoints configured for their database, one for reads and one for writes, then you can use the SecondaryProxyPort to expose a second port on the proxy to provide the second endpoint.

If Heimdall proxy peer autodiscovery is desired (when using Hazelcast), then the autodiscoveryKey and autodiscoveryValue can be set. When using the cloudformation template, the aws tag key should be "Name" and the value should be set to the name of the proxies, generally the cloud formation name+"-proxy".

Microsoft Azure Specific Information

Azure Marketplace Install

When logged into your Azure console, navigate to the [Heimdall offering] (https://portal.azure.com/#blade/Microsoft_Azure_Marketplace/MarketplaceOffersBlade/selectedMenuItemId/home/searchQuery/heimdall) or here:



Configure the appropriate instance settings for your environment, and review/create:

Create a virtual machine

Basics | Disks | Networking | Management | Advanced | Tags | Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image.
Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.
Looking for classic VMs? Create VM from Azure Marketplace

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription: Pay-As-You-Go
* Resource group: azure-testing
Create new

Instance details

* Virtual machine name: heimdall-test
* Region: (US) East US
Availability options: No infrastructure redundancy required
* Image: Heimdall Data Premium Edition (Per Hour)
Browse all public and private images
* Size: **Standard A3**
4 vcpus, 7 GiB memory
Change size

Administrator account

Authentication type: (•) Password ( ) SSH public key
* Username: heimdall
* Password: ••••••••••••••••••
* Confirm password: ••••••••••••••••••

Review settings and create:

Create a virtual machine

Validation passed

Basics | Disks | Networking | Management | Advanced | Tags | Review + create

PRODUCT DETAILS

SQL Performance Improvement and Failover
by Heimdall Data
Terms of use | Privacy policy

Not covered by credits
1.2800 USD/hr

Standard A3
by Microsoft
Terms of use | Privacy policy

Subscription credits apply
0.2400 USD/hr
Pricing for other VM sizes

TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the Azure Marketplace Terms for additional details.

Name	undefined undefined
* Preferred e-mail address	erik@heimdalldata.com
* Preferred phone number	1-866-433-2824

Basics

Subscription	Pay-As-You-Go
Resource group	azure-testing
Virtual machine name	heimdall-test
Region	(US) East US
Availability options	No infrastructure redundancy required
Authentication type	Password
Username	heimdall

Disks

OS disk type	Standard SSD
Use managed disks	Yes
Use ephemeral OS disk	No

Networking

Virtual network	(new) azure-testing-vnet
Subnet	(new) default (10.0.2.0/24)
Public IP	(new) heimdall-test-ip
NIC network security group	(new) heimdall-test-nsg
Accelerated networking	Off
Place this virtual machine behind an existing load balancing solution?	No

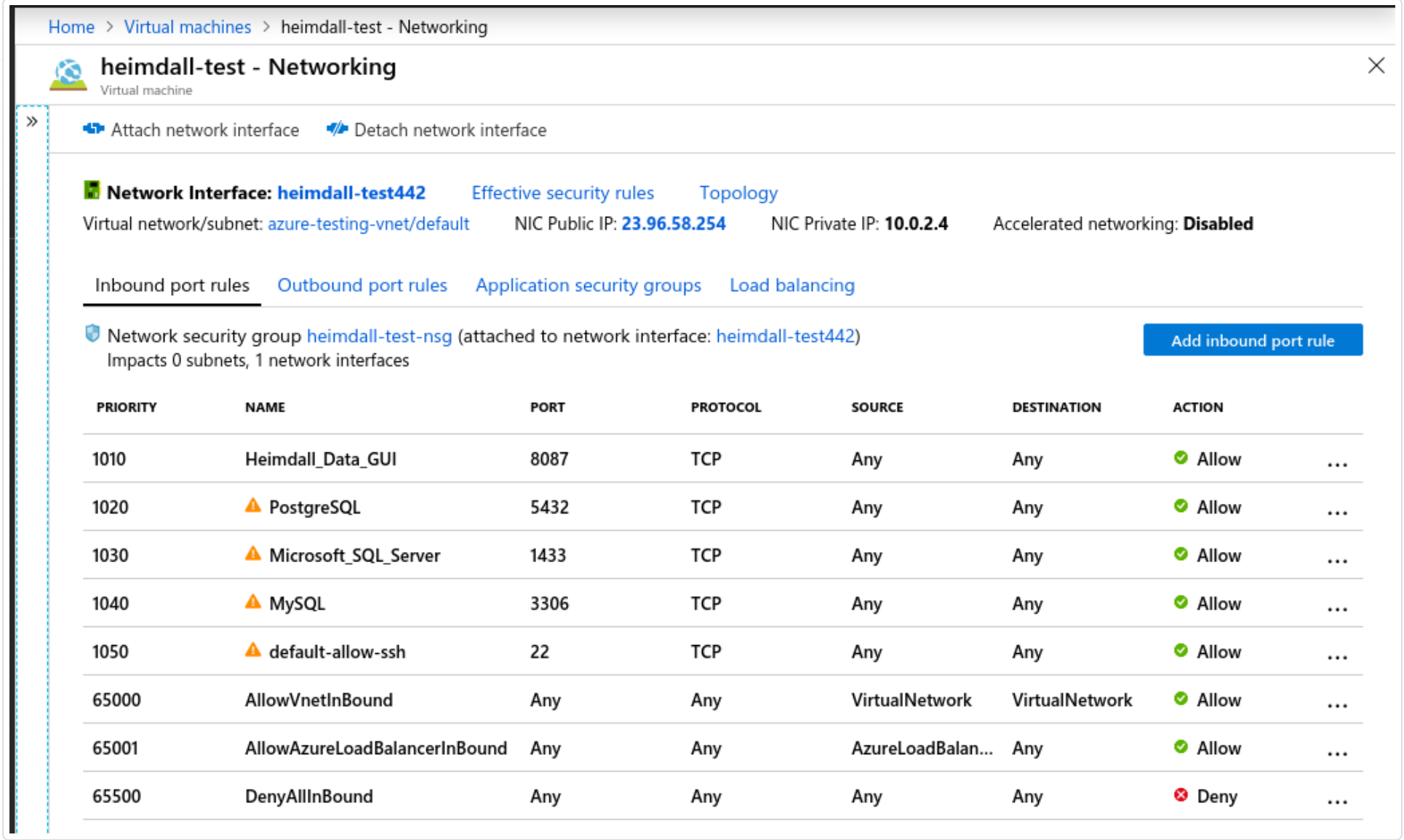
Management

Boot diagnostics	On
OS guest diagnostics	Off
Azure Security Center	Basic (free)
Diagnostics storage account	(new) azuretestingdiag739
System assigned managed identity	Off
Auto-shutdown	Off

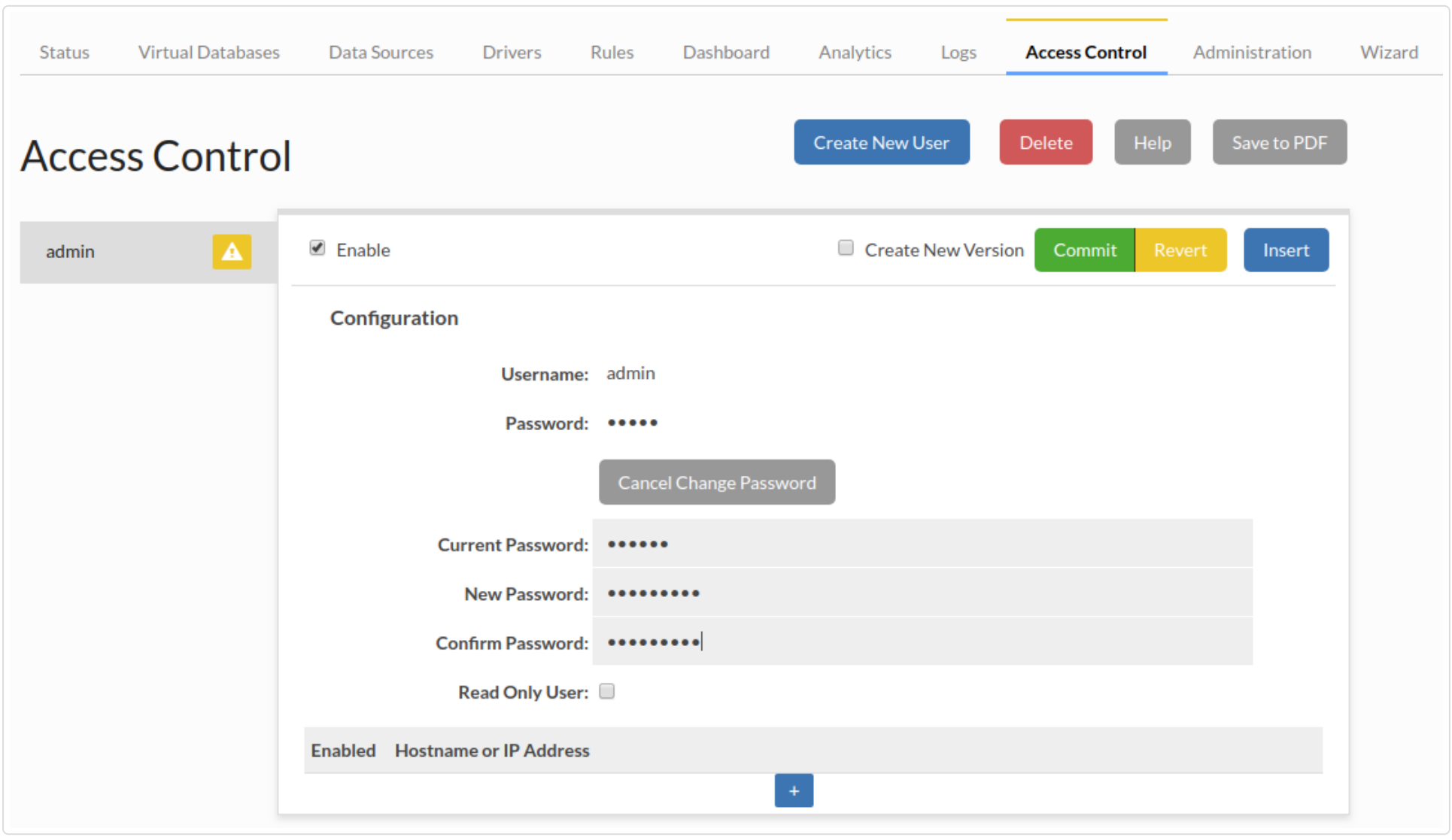
Advanced

Extensions	None
Cloud init	No

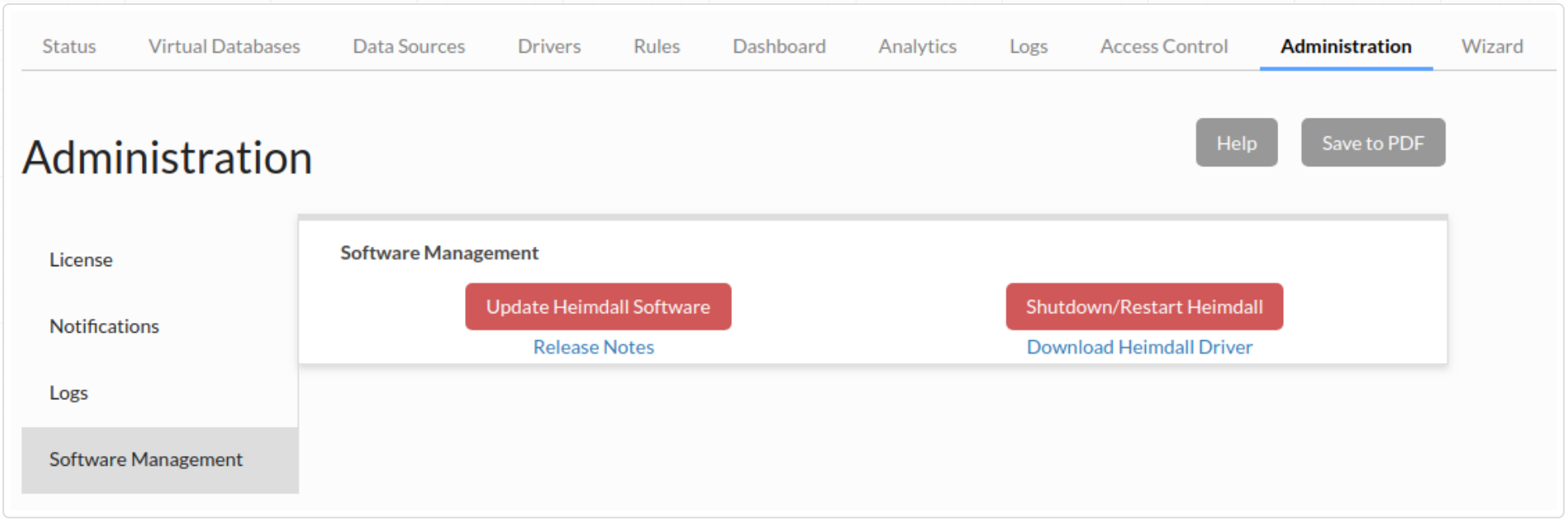
Once provisioned, review and adjust the instance networking settings to ensure that the appropriate ports are open, and unneeded ports are closed:



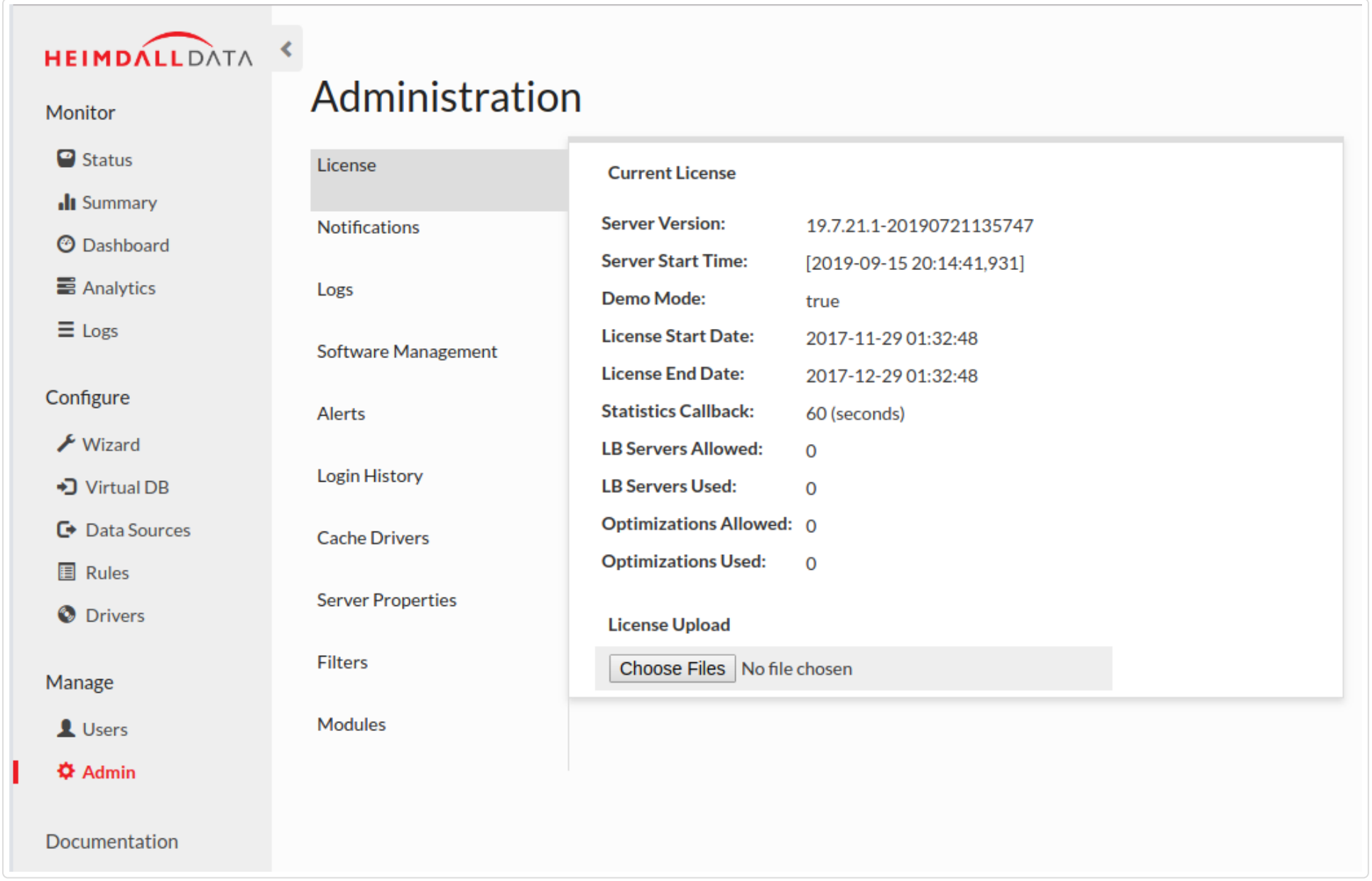
Once the instance is running, login via "admin" and "heimdall", then change the admin password:



Finally, update the software to the newest release version (The Azure image does not automatically download updates):



After a few minutes, refresh your browser (use ctrl-refresh if the UI appears broken), and check the server version, which should be at least 19.7.21.1.



If the "LB Servers Allowed" is set to zero, please contact Heimdall support for a no-cost Azure Marketplace license, to disable notices about unlicensed usage, and upload via the "License Upload" option (as shown above).

Azure Managed Databases

When using the Azure Postgres or Azure MySQL managed databases, there is a requirement that the username used is of the format "user@hostname". Heimdall provides an option in the data source options of "azureDbHost", which if set to true, will automatically map the username to this format based on which server is being connected to. Ensure that when using this option, the fully qualified domain name is set in the JDBC URL.

SSL Note

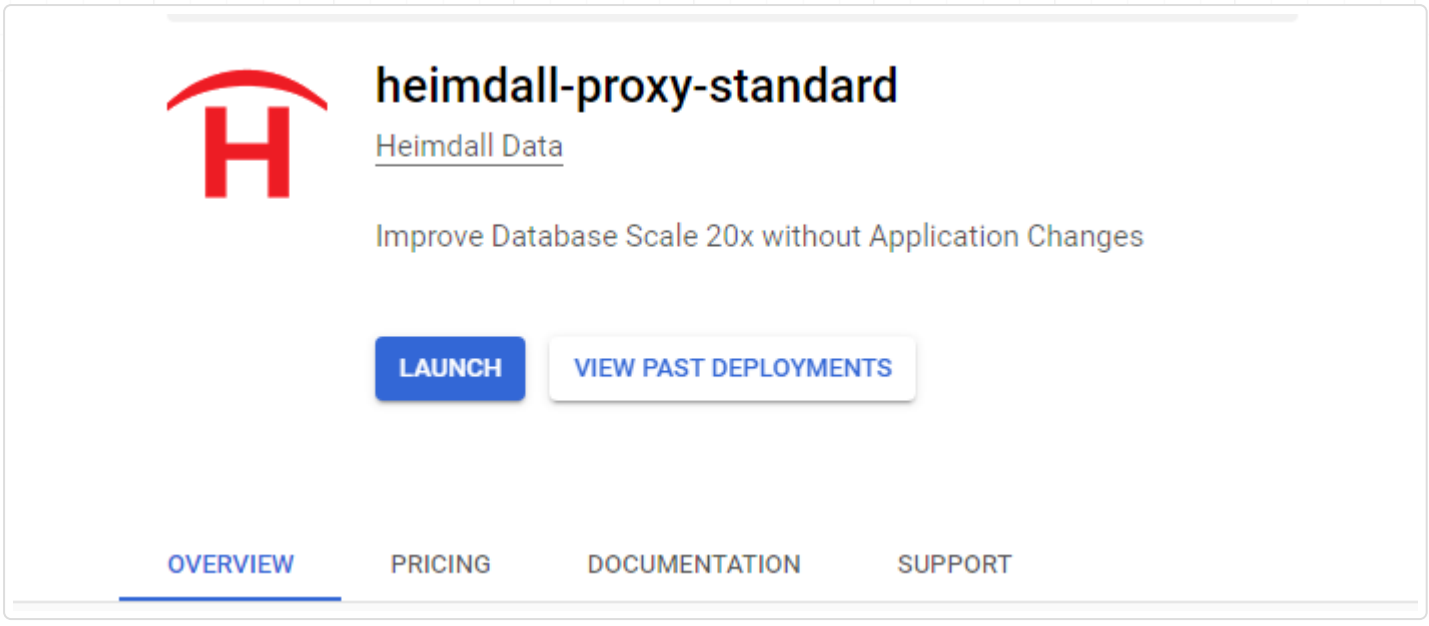
With Postgres Azure, SSL is required, so in the connection properties, please set properties of "ssl=true" and "sslmode=require". Other SSL settings may work as well, but this is sufficient to ensure that the connectivity can be established.

Google Cloud Specific Information

GCP Marketplace Install

When logged into your Google Console, navigate to the Heimdall offering and select the desired Heimdall offer (Standard or Enterprise) based on support needs.

Configure the appropriate instance settings for your environment, and review/create:



Review settings and change as needed and deploy, adjusting the ports that are open as needed. Port 22 and 8087 should be exposes, as well as the port desired for your database traffic:

Deployment name

heimdall-proxy-standard-draft-1

Zone

us-west1-c

Machine type

2 vCPUs | 4 GB memory | Customize

Boot Disk

Boot disk type

Standard Persistent Disk

Boot disk size in GB

20

Networking

Network interfaces

default default (10.138.0.0/20)

+ Add network interface

You have reached the maximum number of one network interface

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

Creating certain firewall rules may expose your instance to the Internet. Please check if the rules you are creating are aligned with your security preferences. Learn more

Allow TCP port 22 traffic from the Internet

Source IP ranges for TCP port 22 traffic

0.0.0.0/0, 192.169.0.2/24

Allow TCP port 8087 traffic from the Internet

Source IP ranges for TCP port 8087 traffic

0.0.0.0/0, 192.169.0.2/24

Allow TCP port 3306 traffic from the Internet

Source IP ranges for TCP port 3306 traffic

0.0.0.0/0, 192.169.0.2/24

Allow TCP port 5432 traffic from the Internet

Source IP ranges for TCP port 5432 traffic

0.0.0.0/0, 192.169.0.2/24

Allow TCP port 1433 traffic from the Internet

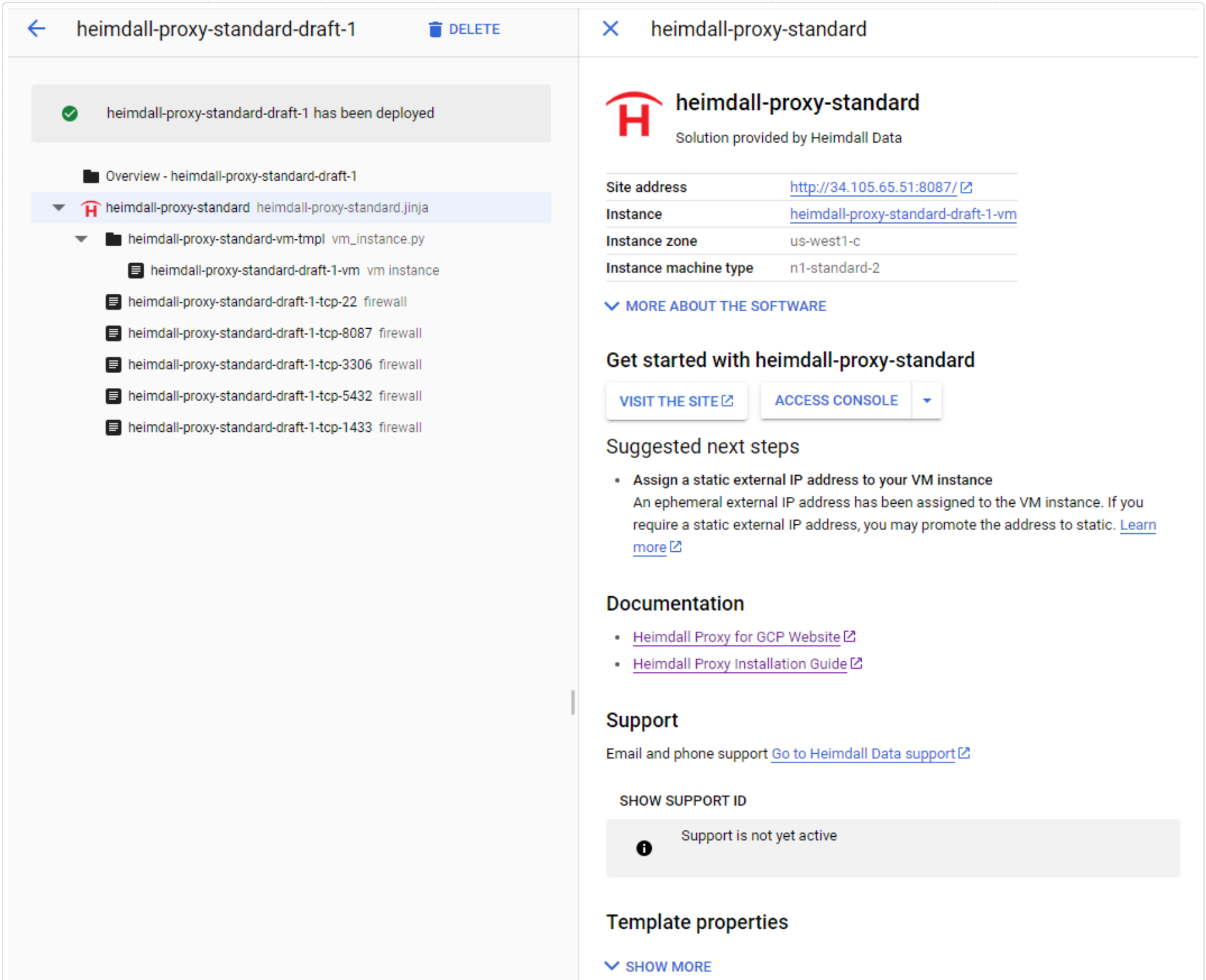
Source IP ranges for TCP port 1433 traffic

0.0.0.0/0, 192.169.0.2/24

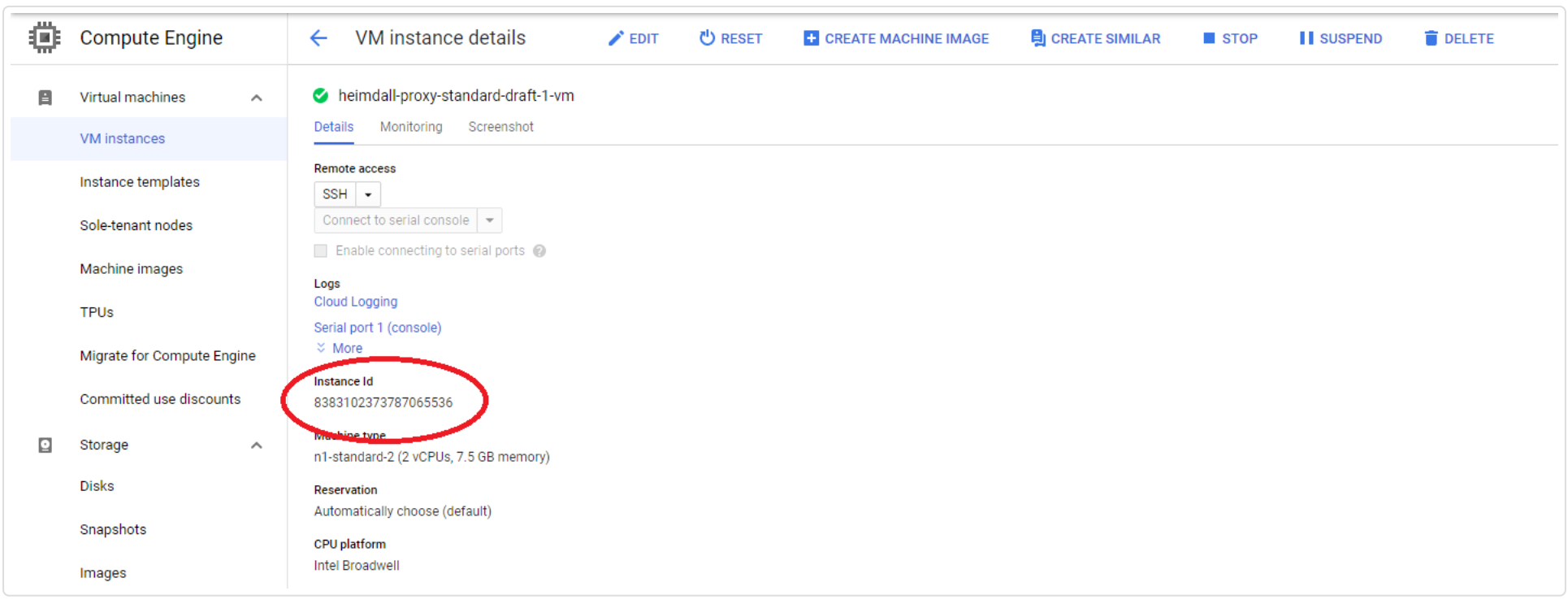
More

Deploy

Once deployed, the following (or similar) should be provided, including a link to the Admin interface via "visit the site" or the SSH console with "Access console":



Once the instance is running, login via "admin" and the instance ID the deployment is under:



From here, the instance can be used as per the rest of the install documentation.

Databases Support

Heimdall supports all the CloudSQL database engines as well as AlloyDB.

Cloud SQL Support

Heimdall has been tested with Cloud SQL:

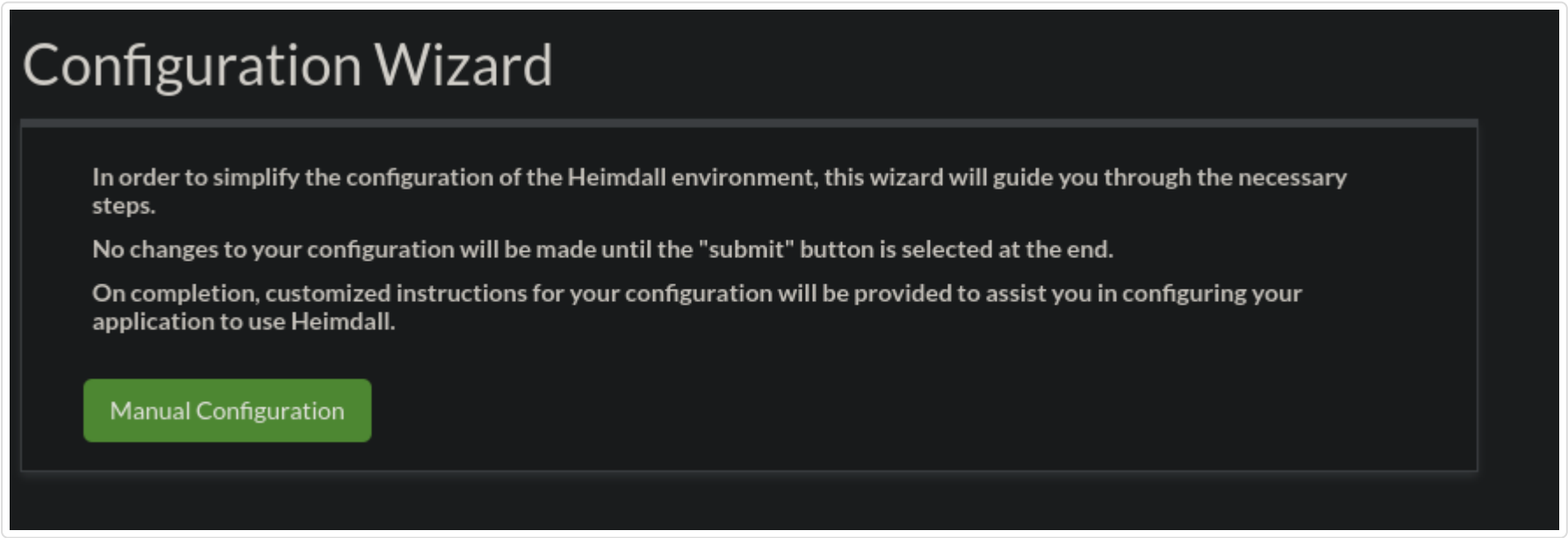
- MySQL 8.0, 5.7, and 5.6
- PostgreSQL 15, 14, 13, 12, 11, 10, and 9.6
- SQL Server 2022, 2019, and 2017

The following configurations have also been tested:

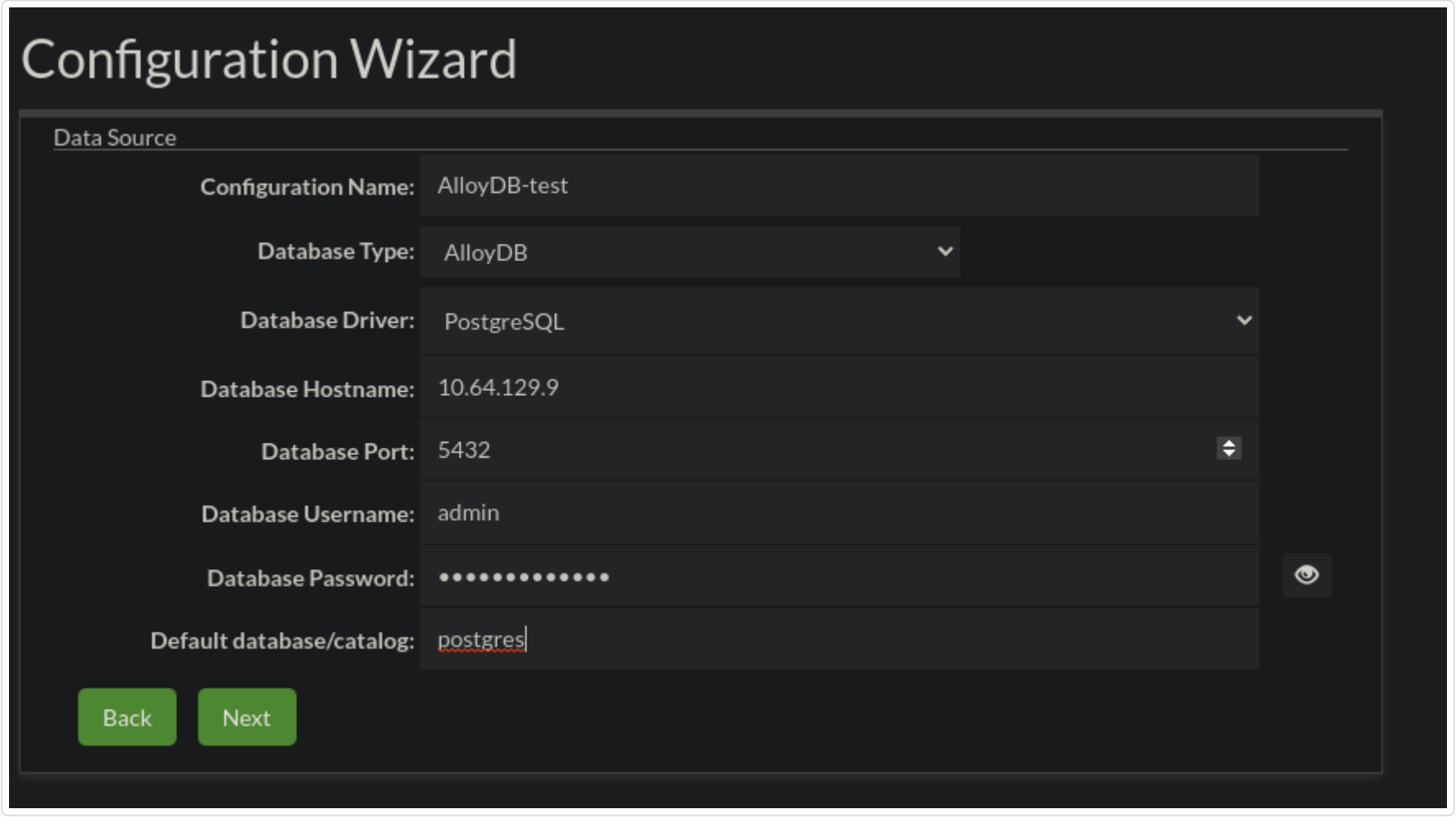
- Single Zone
- Multiple Zones

To configure a CloudSQL VDB, please use the configuration wizard.

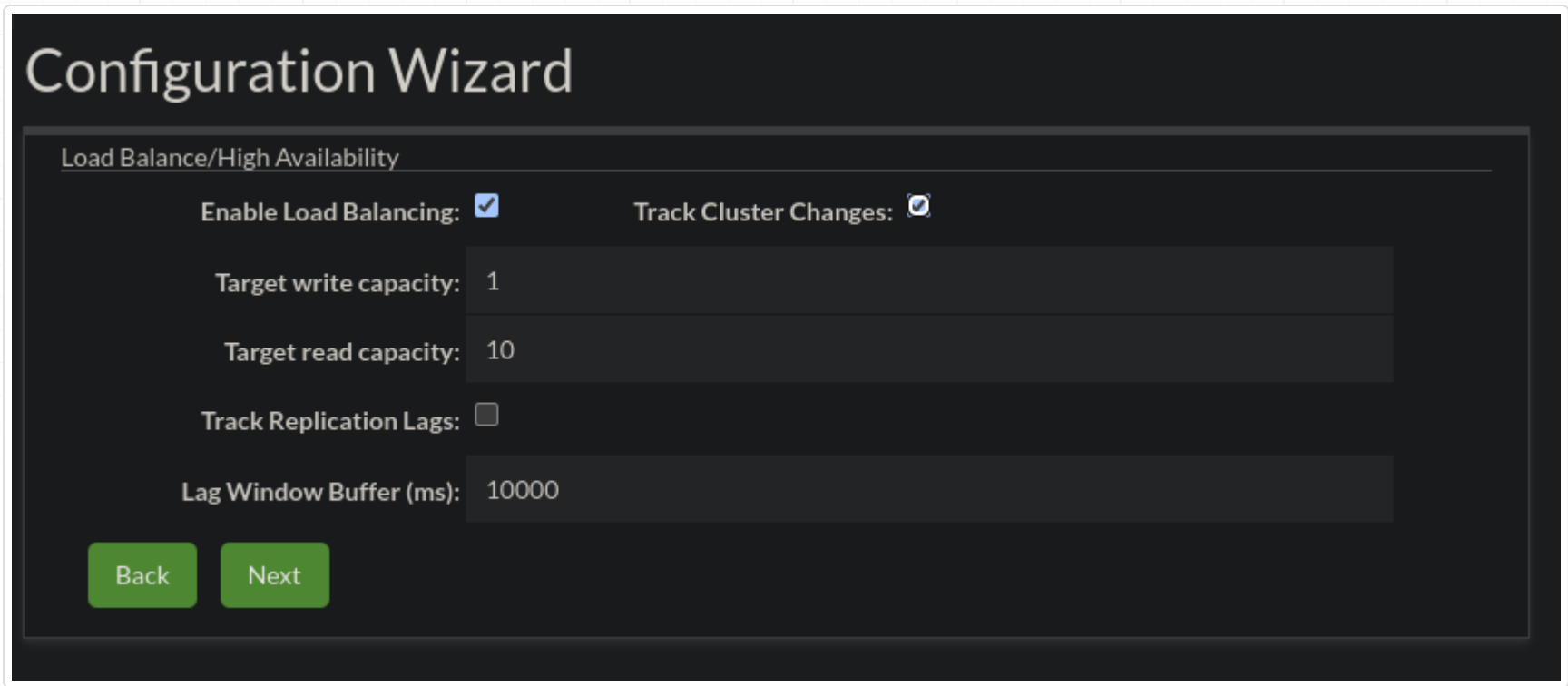
First, select manual configuration to configure the data source:



Next, fill in the appropriate parameters, selecting the appropriate type as the DB type, and the matching database driver for the driver, and the IP address of the writer node:



Continue through the wizard. Once completed, you can navigate to the data source tab for the CloudSQL source:



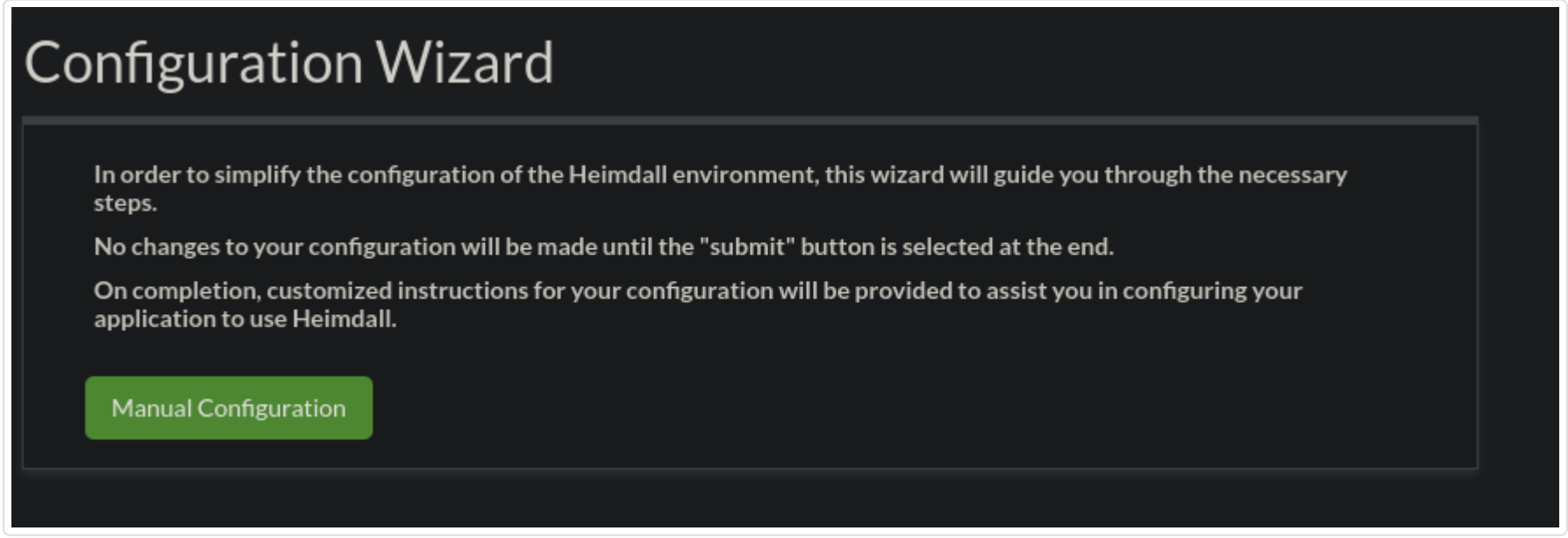
AlloyDB Support

Heimdall has been tested with AlloyDB with Postgres 14 compatibility, in the following modes:

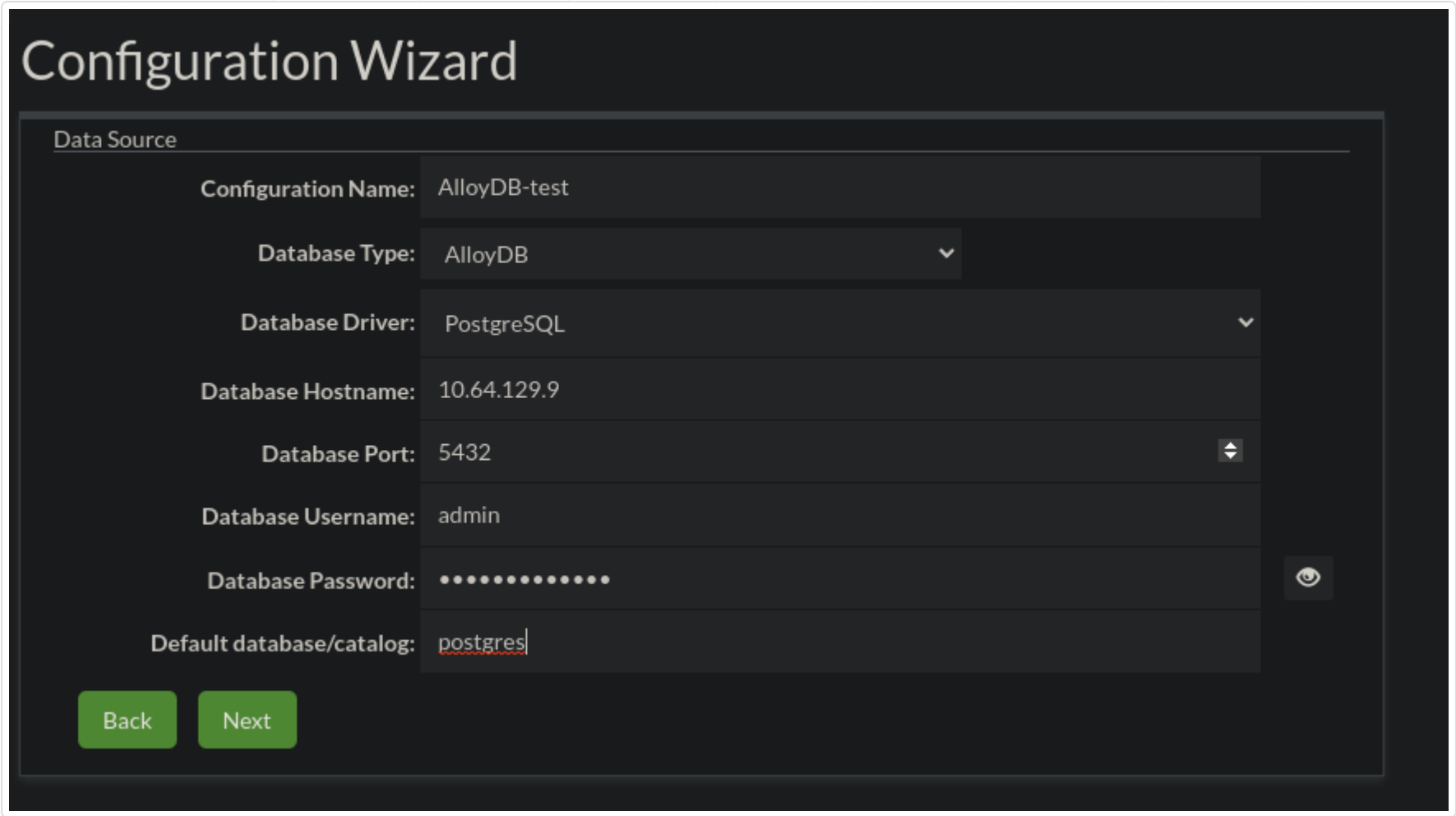
- Highly available
- Highly available w/ read pool(s)
- Basic
- Basic w/ read pools

To configure an AlloyDB VDB, please use the configuration wizard.

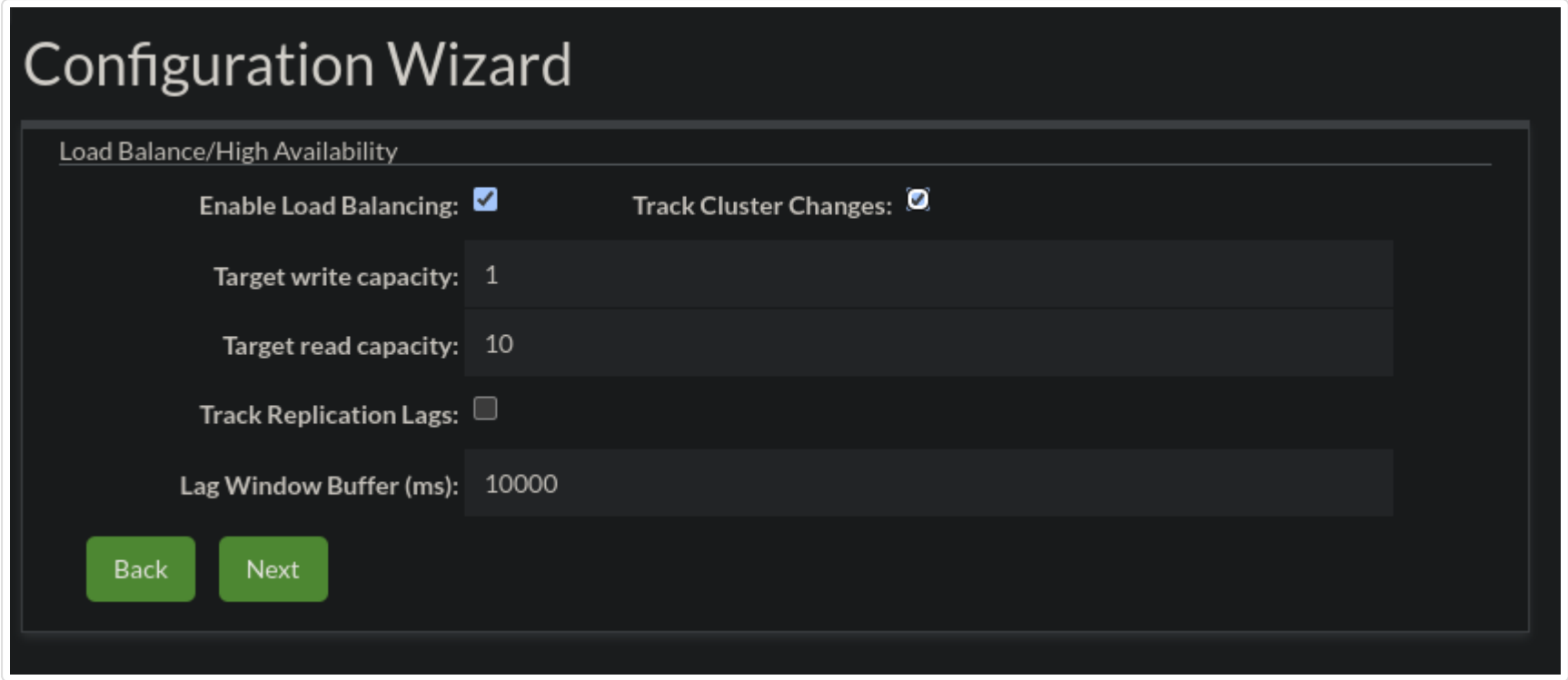
First, select manual configuration to configure the data source:



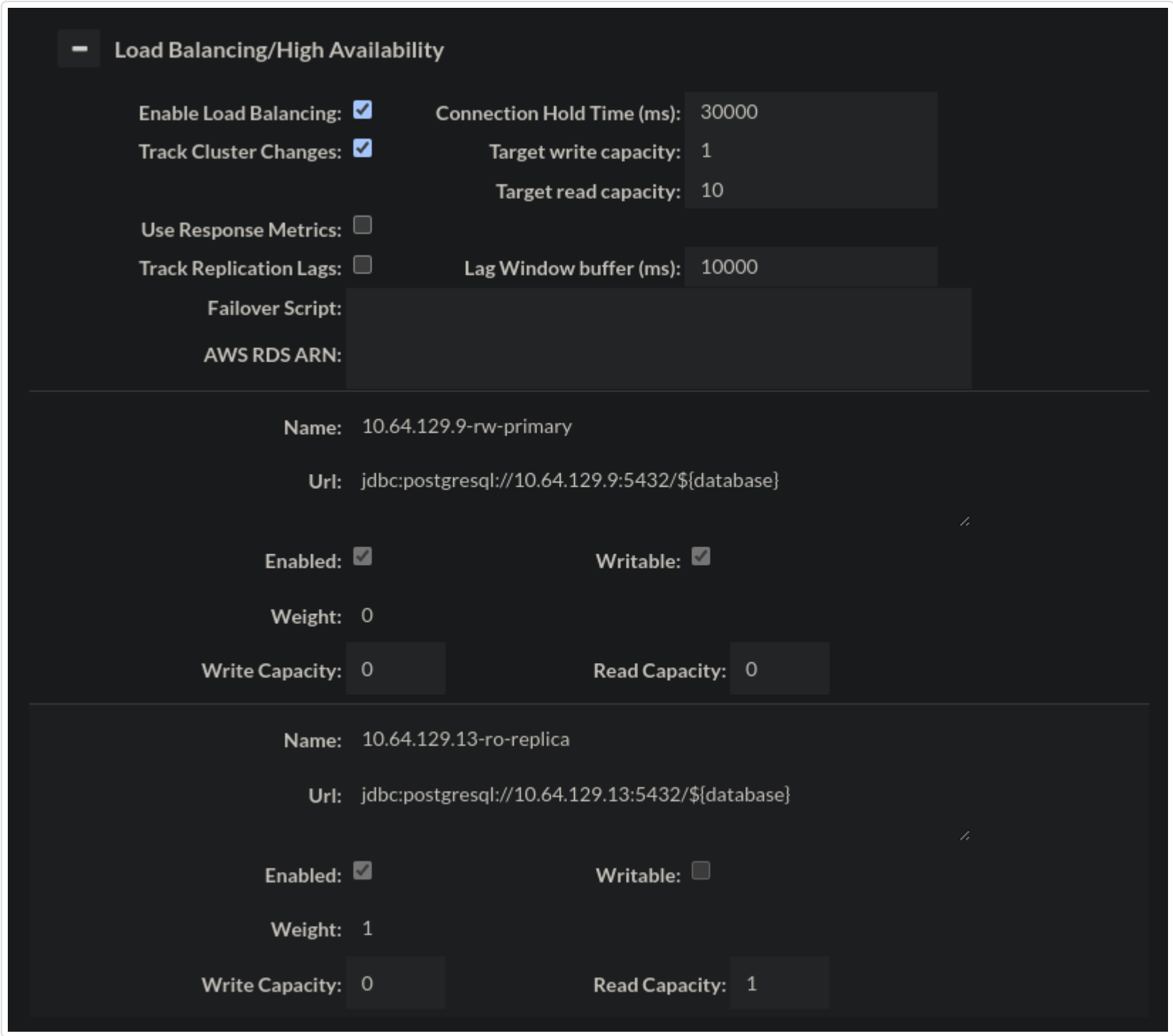
Next, fill in the appropriate parameters, selecting AlloyDB as the DB type, and the postgres driver for the driver, and the IP address of the writer node:



Next, enable load balancing (if desired) and enable track cluster changes, so as to auto-detect reader nodes and cluster changes:



Continue through the wizard. Once completed, you can navigate to the data source tab for the AlloyDB source, and in the LB section (if enabled), you should have all the nodes listed in your cluster:


Load Balancing/High Availability
Enable Load Balancing:
Track Cluster Changes:
Use Response Metrics:
Track Replication Lags:
Failover Script:
AWS RDS ARN:
Connection Hold Time (ms): 30000
Target write capacity: 1
Target read capacity: 10
Lag Window buffer (ms): 10000
Name: 10.64.129.9-rw-primary
Url: jdbc:postgresql://10.64.129.9:5432/${database}
Enabled:
Writable:
Weight: 0
Write Capacity: 0
Read Capacity: 0
Name: 10.64.129.13-ro-replica
Url: jdbc:postgresql://10.64.129.13:5432/${database}
Enabled:
Writable:
Weight: 1
Write Capacity: 0
Read Capacity: 1

MemSQL/SingleStore Specific Information

Common Issues

As MemSQL/SingleStore doesn't have a mysql database, set the "defaultCatalog" to be "memsql" instead, otherwise, the proxy won't be able to properly connect for initialization.

PHP PDO server's certificate verification

Known are problems about verifying server certificate by PHP PDO during using SSL for connection to a proxy. To help with resolving a problem, below suggestions can be helpful.

Applications suggested to use during work on a problem:

- KeyStore Explorer

Reminder before work

- All keys used by Heimdall are saved in a keystore named `keystore.p12`.
- Password to the keystore is `heimdall`.
- All key pair have to have empty password.
- Default global key pair used by each proxy is named `global_use_certificate`.
- Specific key pair for a proxy can be specified by adding key pair with alias matching lowercase name of a proxy, i.e. proxy named `MySQLVirtualDatabase` will use key pair aliased `mysqlvirtualdatabase`. That key pair have higher priority than global key pair.

1. Check certificate

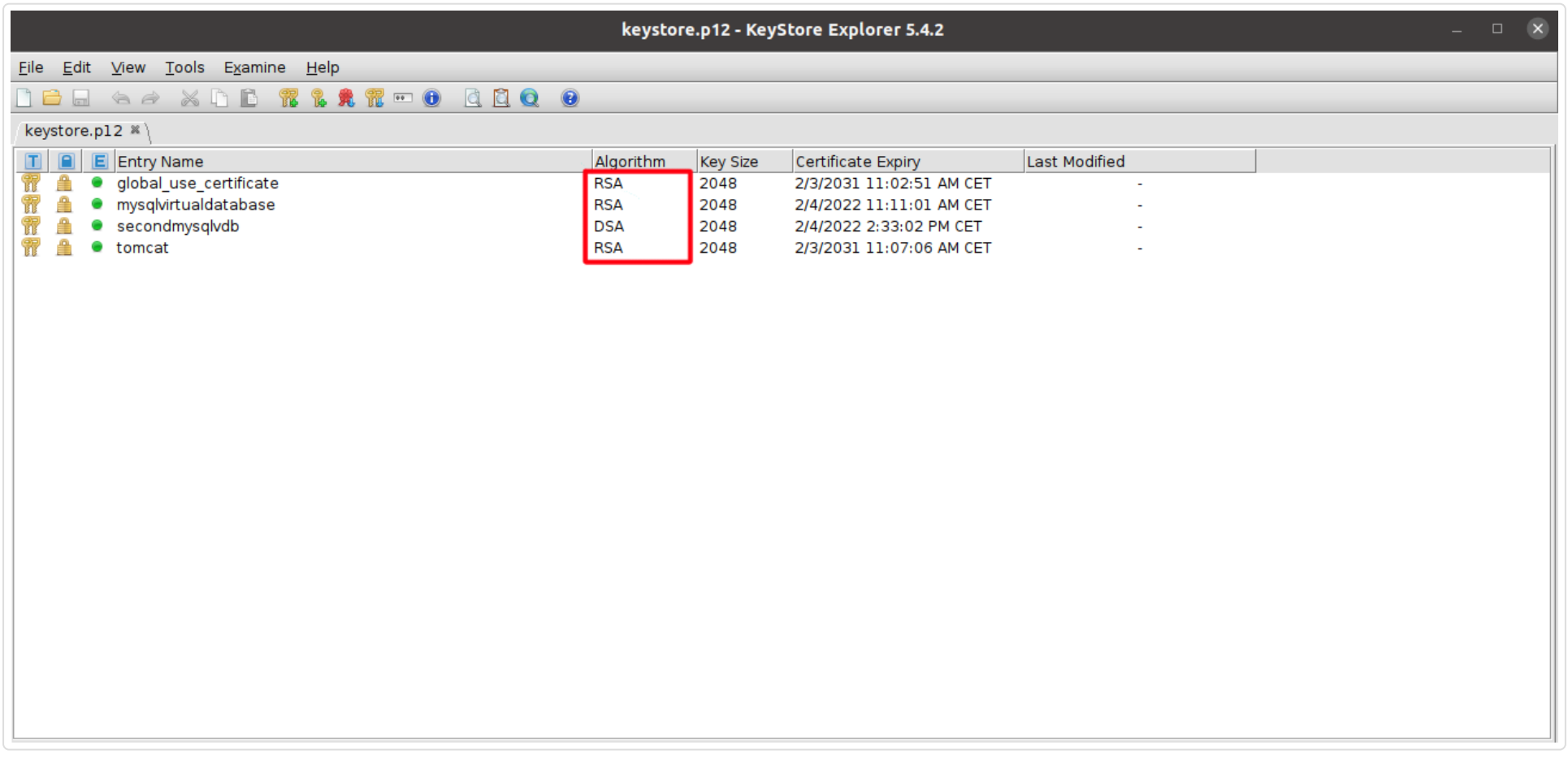
There are known two issues with certificates, which can lead to problems during using PHP PDO. Issue can be:

- an incorrect algorithm used during generating certificate,
- an incorrect value of CN, which can be translated to an IP address.

1.1. Check used algorithm

First, algorithm used to generate key pair should be checked. There are known issues, when certificate from a key pair generated via DSA algorithm was causing issues. By default, self-signed certificate generated by Heimdall is generated by using RSA algorithm, but if key pair was generated or imported by an user, then that can't be assured.

Below can be seen marked section were is written what algorithm was used to generate key pair. Window below is visible after opening and unlocking keystore file.



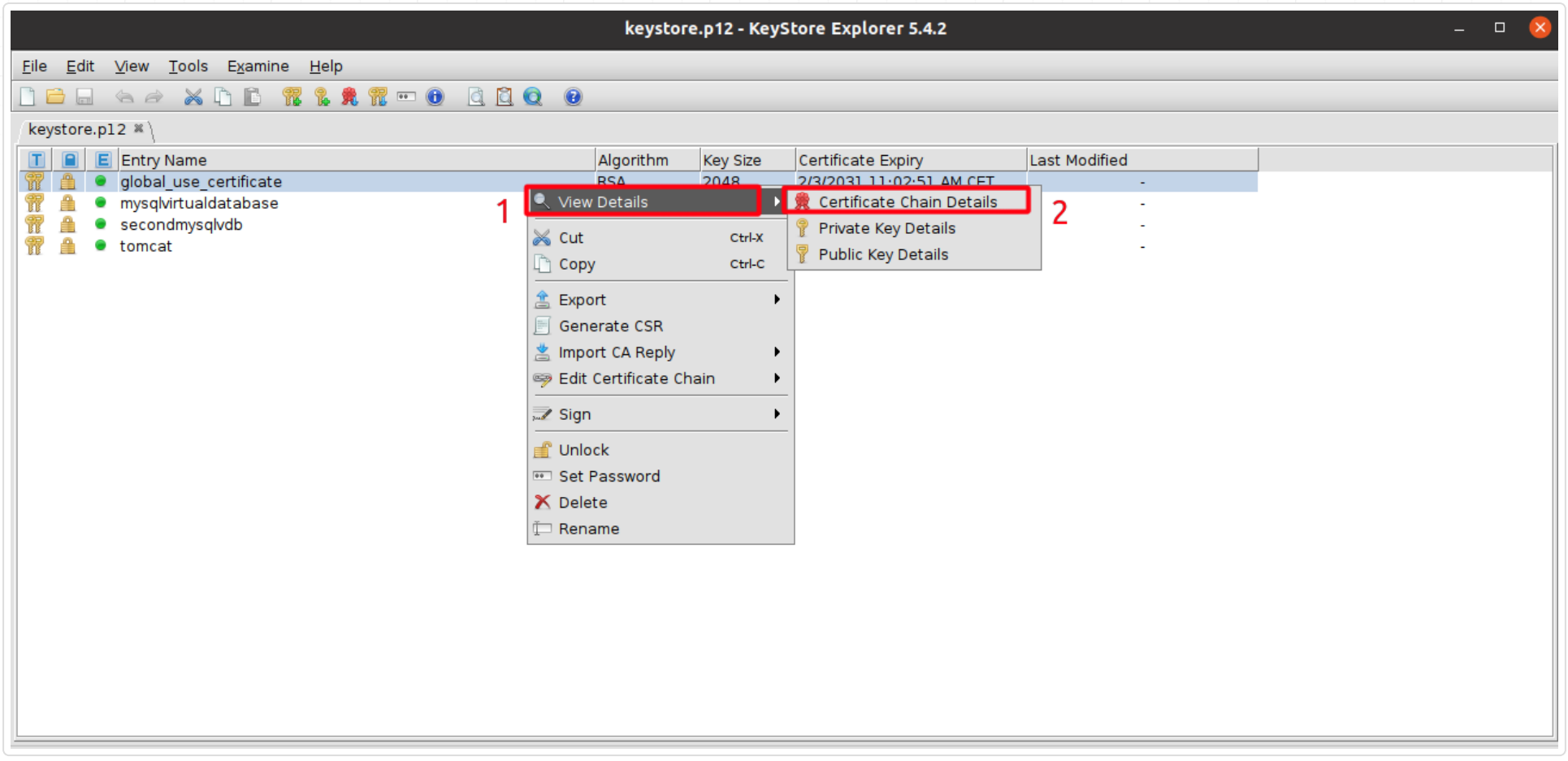
If key pair was generated with using DSA, then suggested is generating or importing key pair with used RSA algorithm to generate them.

1.2. Check CN value

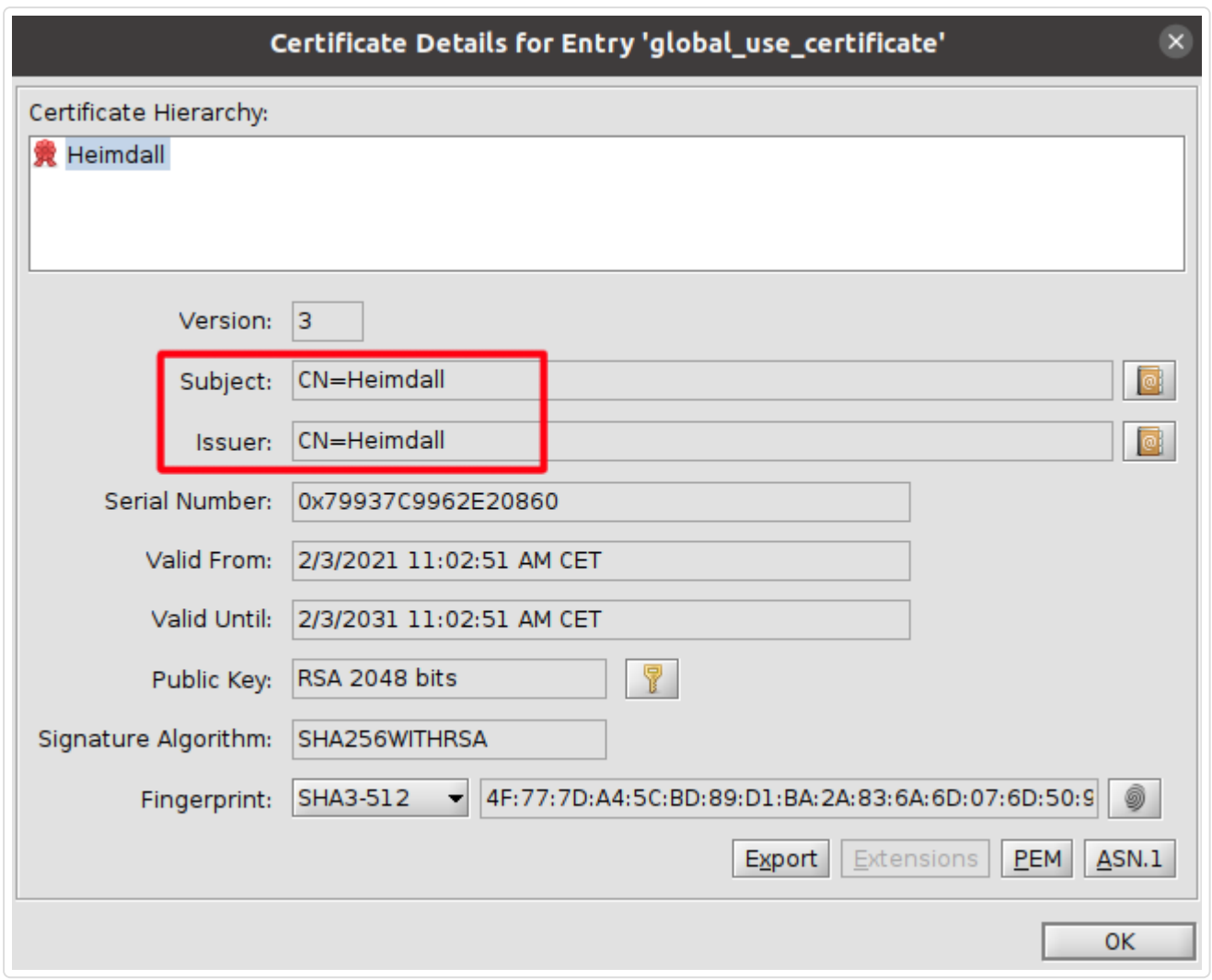
After checking algorithm, CN value used to sign certificate should be checked. There is a known requirement on PHP PDO using SSL that CN value should match or be possible to translate to an IP address.

First step to check CN value is checking Certificate Chain Details. To get access to that window, below steps have to be done:

1. Right click chosen key pair.
2. Choose option `View Details` (area 1 marked on the image below).
3. Click option `Certificate Chain Details` (area 2 marked on the image below).



After completing above steps, the window below should pop up.In the marked area is given information about certificate details of a key pair. By default, certificate generated by Heimdall is self-generated and CN value is matching value `Heimdall`. The CN value should be a proper IP address or be possible to translate to an IP address.

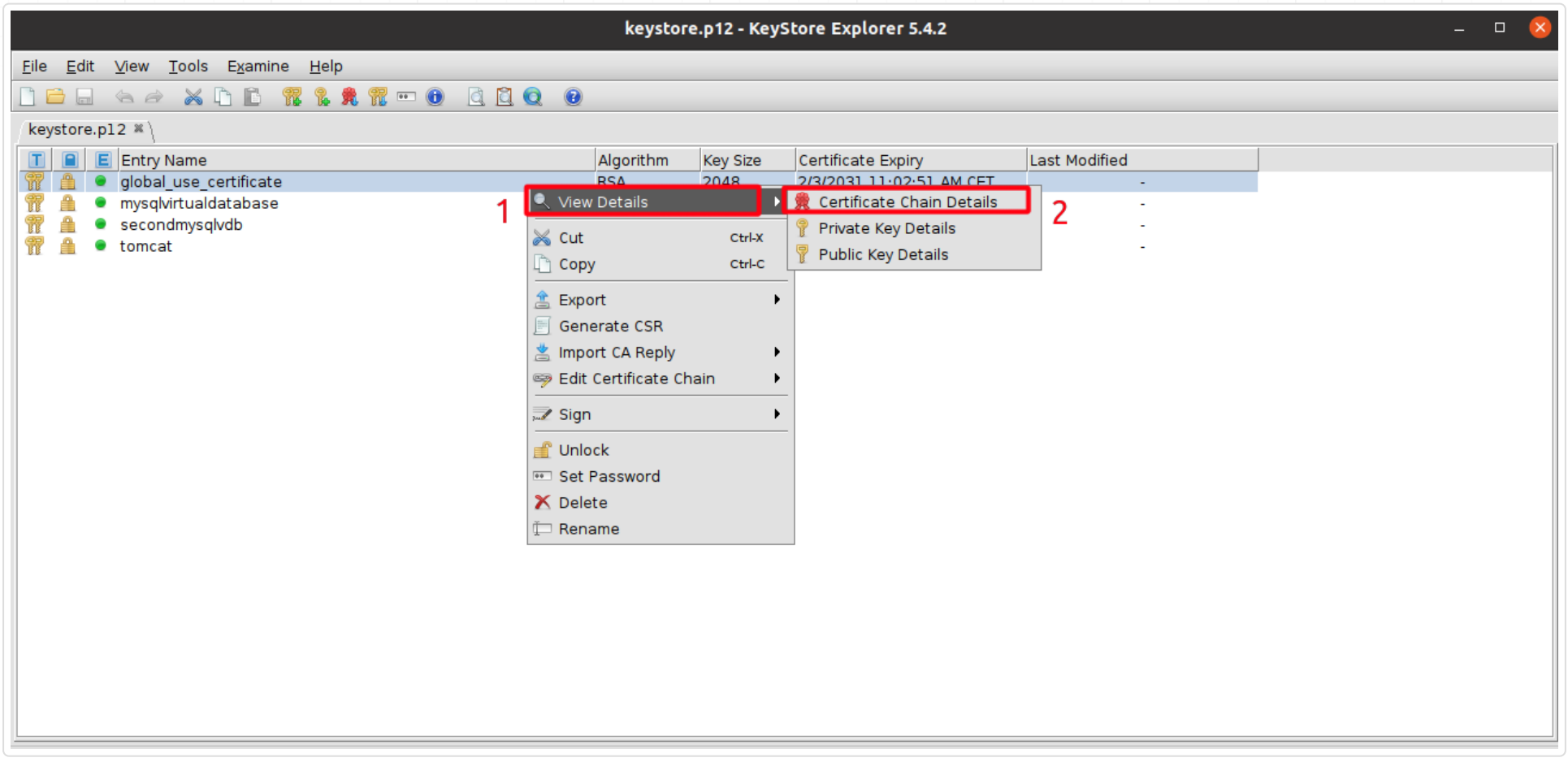


If key pair's CN value isn't a proper IP address and can't be translated to an IP address, then suggested is generating or importing key pair with CN value which meets these requirements.

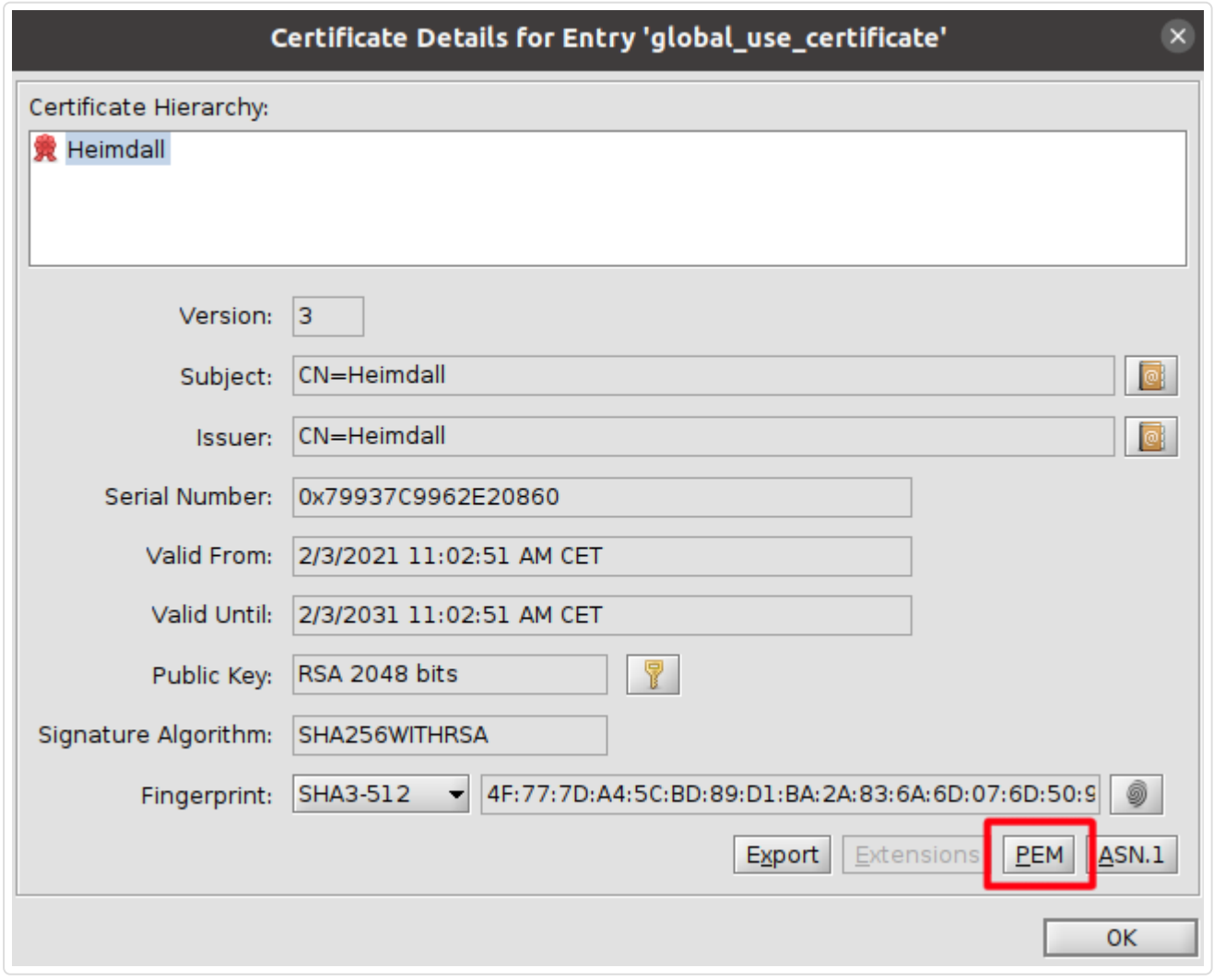
2. Export certificate to PEM format

After ensuring that certificate meets the requirements of PHP PDO's SSL certificate verification, the certificate can be finally exported to .pem file. To do that, first should be opened Certificate Chain Details window. To get access to that window, below steps have to be done:

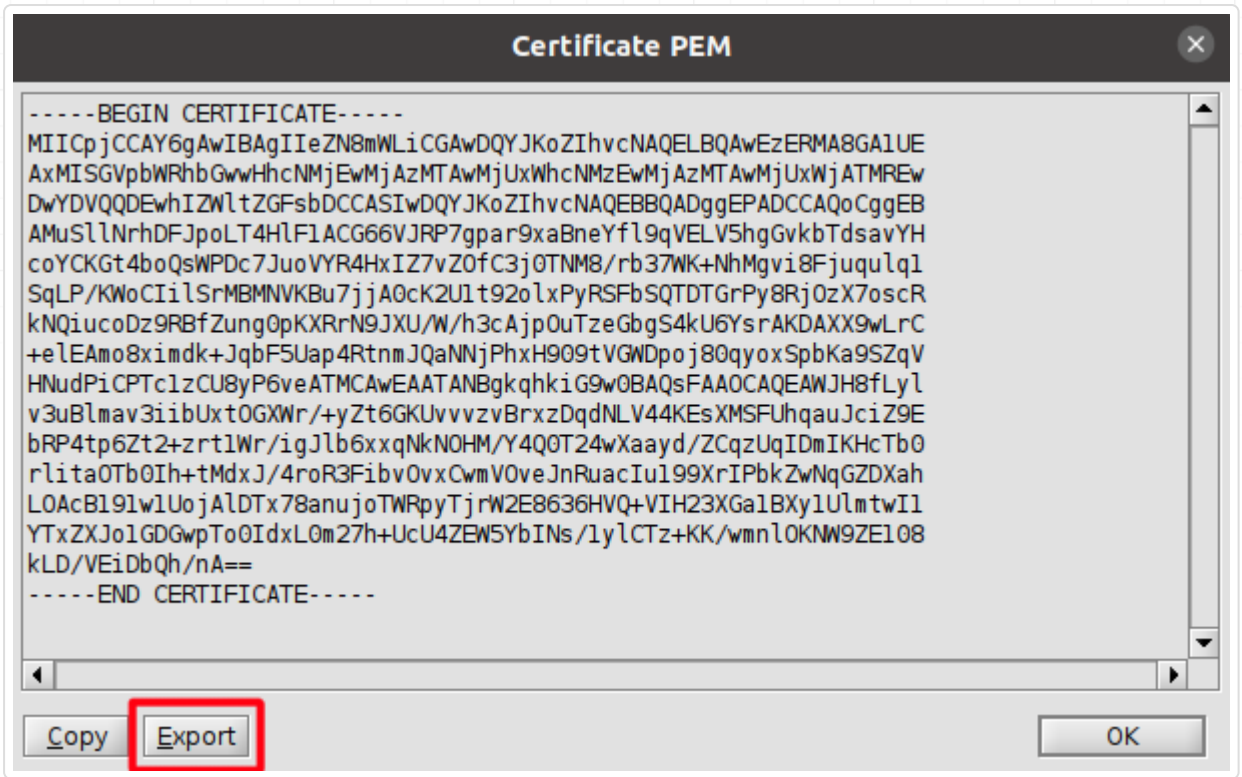
1. Right click chosen key pair.
2. Choose option `View Details` (area 1 marked on the image below).
3. Click option `Certificate Chain Details` (area 2 marked on the image below).



After completing above steps, below window should pop up. Next step is opening certificate view in PEM format. To do that, click marked on the below image button `PEM`.

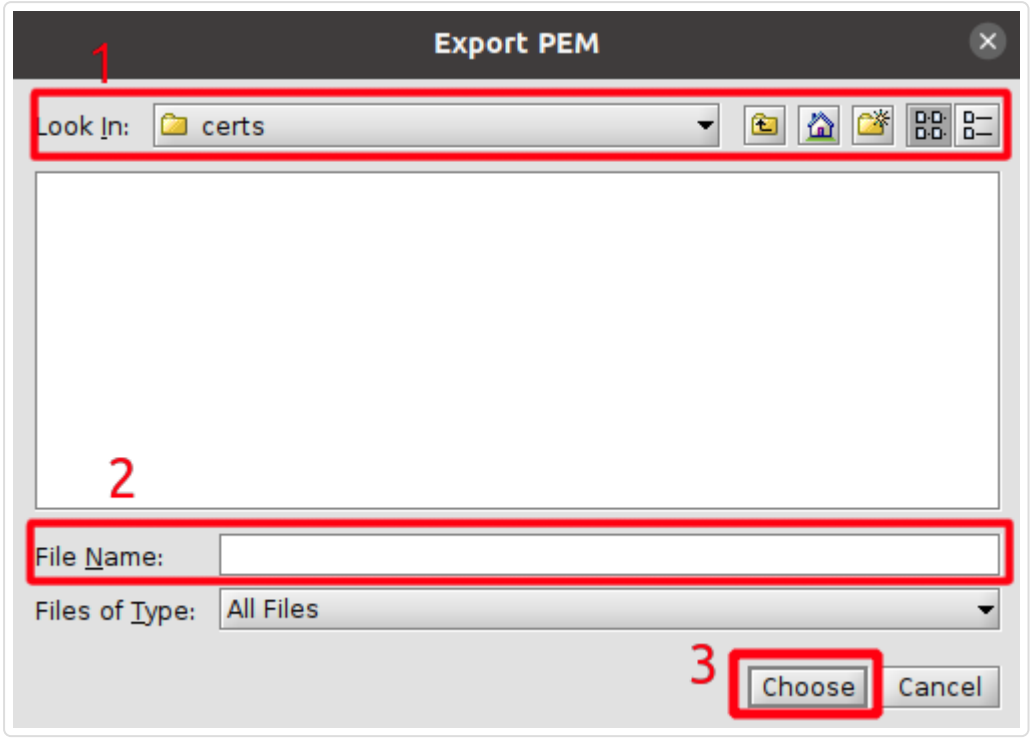


After clicking `PEM` button, below window should pop up. To begin exporting certificate in PEM format, button `Export` should be clicked (as marked on the image below).



After clicking `Export` button, below window should pop up. The last step is to save exported PEM certificate to the file. To do that:

1. Choose in what directory should the certificate be saved (by using controls in marked are 1).
2. Name a file with exported PEM file by writing name in area 2 marked on the image below. Remember that suffix `.pem` won't be added automatically, so can be added, if needed, during naming file.
3. Click button `Choose` in the marked area 3 on the image below.



After following above steps, the certificate should be exported in the `PEM` format to the chosen file.

Example configuration in PHP PDO application

After exporting certificate to a PEM format, it is ready to use in PGP PDO application. Below can be seen an example configuration in PHP code to set connection via PDO to a MySQL database, with defined SSL parameters.

```
$myPDO = new PDO('mysql:host=example.database.com;dbname=mysql', 'root', 'password', array(
    PDO::MYSQL_ATTR_SSL_KEY    => '/example/keys/client-key.pem',
    PDO::MYSQL_ATTR_SSL_CERT   => 'certificates/client-cert.pem',
    PDO::MYSQL_ATTR_SSL_CA     => 'certificates/server-cert.pem'
));
```

SSL parameters used above are used to declare:

- `PDO::MYSQL_ATTR_SSL_KEY` - declares a file with a key, to be used during SSL connection
- `PDO::MYSQL_ATTR_SSL_CERT` - declares a file with a certificate, to be used during SSL connection
- `PDO::MYSQL_ATTR_SSL_CA` - declares a file with a certificate, to be verified if matches server's certificate. A certificate exported in previous steps should be set here.

Above parameters declare file path. The file path can be written as absolute file path (as can be seen for parameter `PDO::MYSQL_ATTR_SSL_KEY`) or as relative file path (as can be seen for parameters `PDO::MYSQL_ATTR_SSL_CERT` and `PDO::MYSQL_ATTR_SSL_CA`).

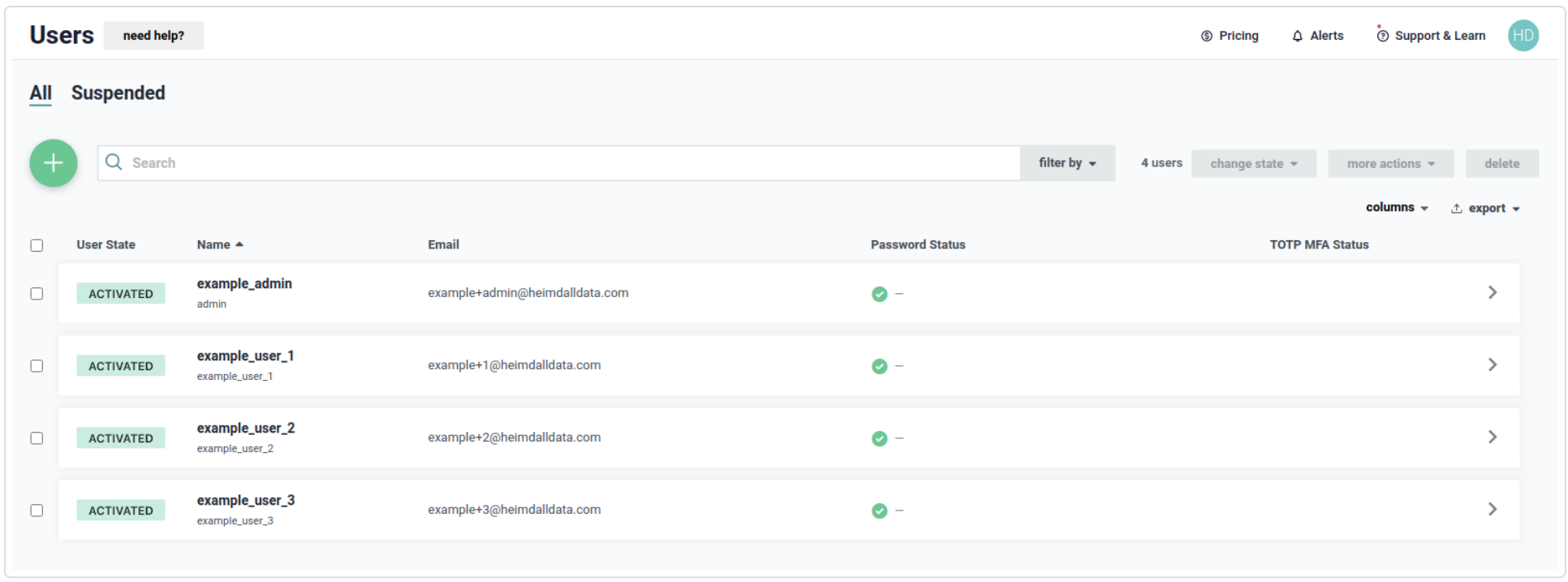
JumpCloud's LDAP authentication

This section describes how to configure Heimdall Proxy's LDAP authentication to work with JumpCloud's LDAP implementation.

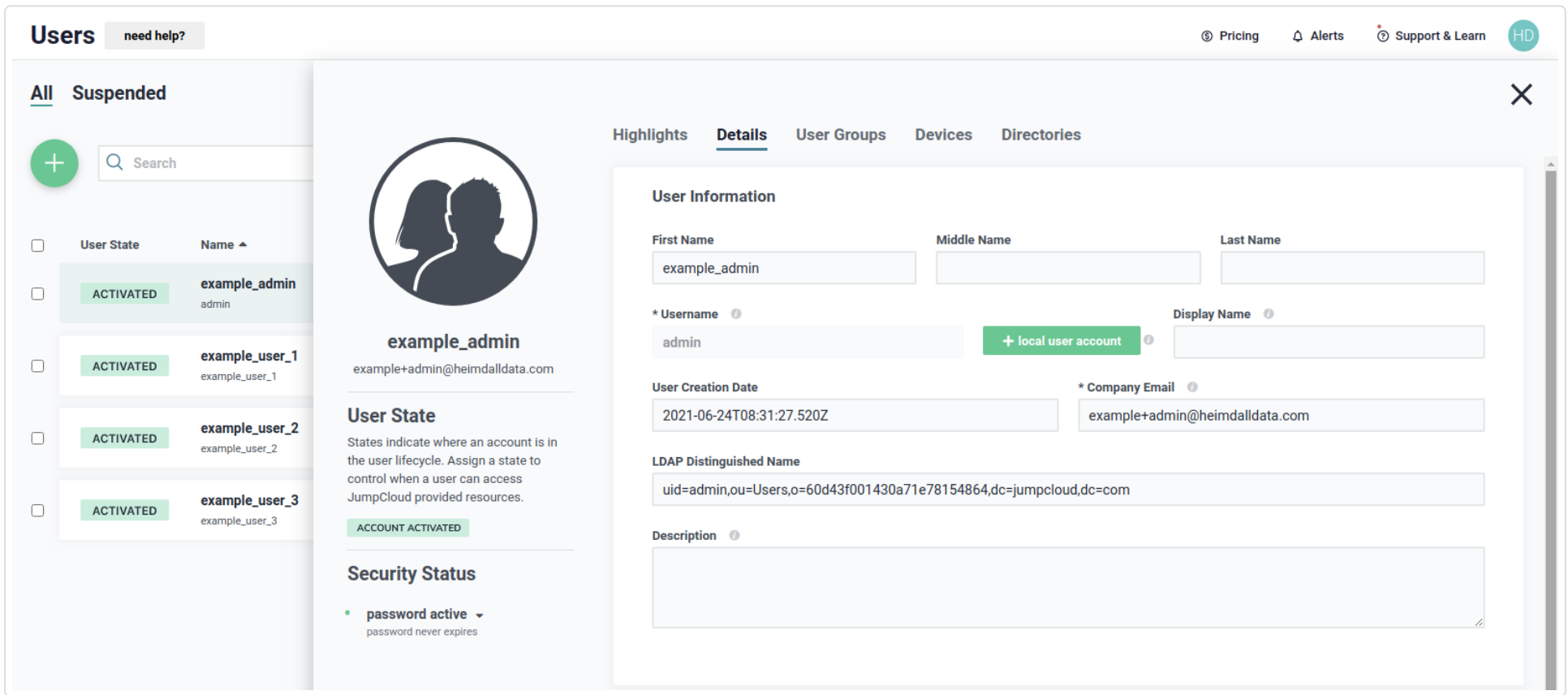
Remember that for easy checking if configuration works in your environment, you can use "Authentication Test" in "Proxy Configuration" in Virtual DB configuration tab. By providing only a username and password, you can check if your actual configuration works and users can be authenticated as you expect. Other details, like IP address, database or using SSL, aren't needed to be filled to do a test for LDAP authentication mode.

Prerequisites

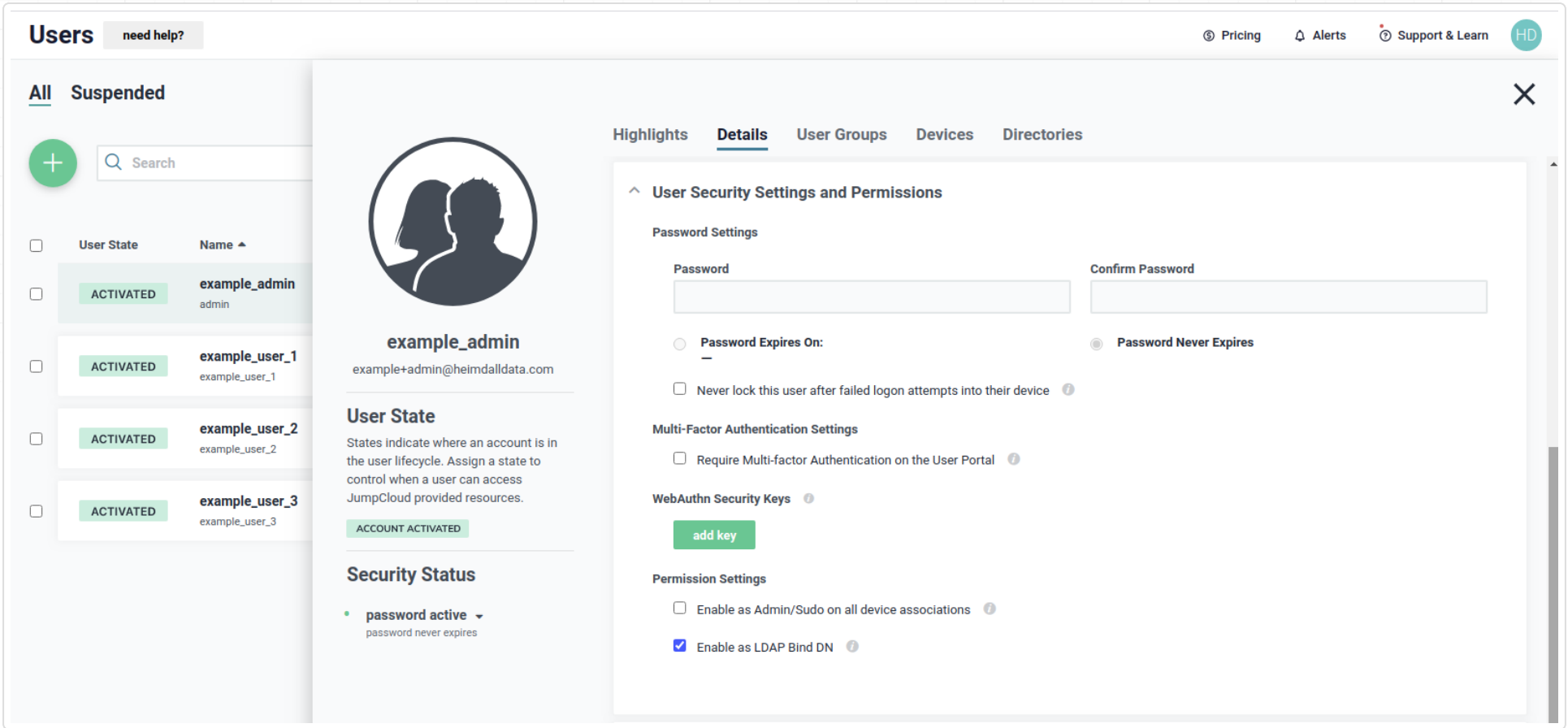
At the beginning, we will describe the JumpCloud LDAP configuration which will be used in this section. First part is setting users. As can be seen on the below screenshot, we have four users - `example_admin`, `example_user_1`, `example_user_2` and `example_user_3`.



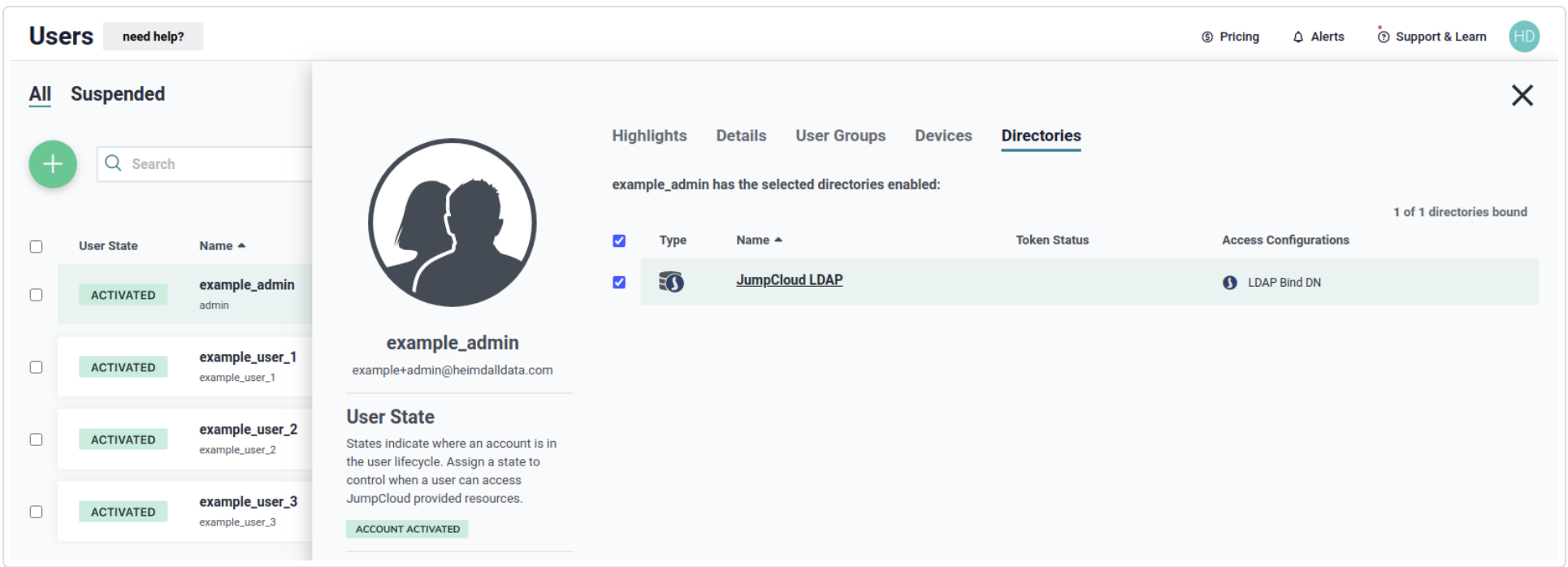
First user - `example_admin` has a details as shown below. Please notice that username is different from first name. Username of the user is `admin` and that name determines LDAP Distinguished Name.



User `example_admin` will be used for performing bind and search operations in JumpCloud LDAP, so we have to give that user a permission to do that, as shown below.

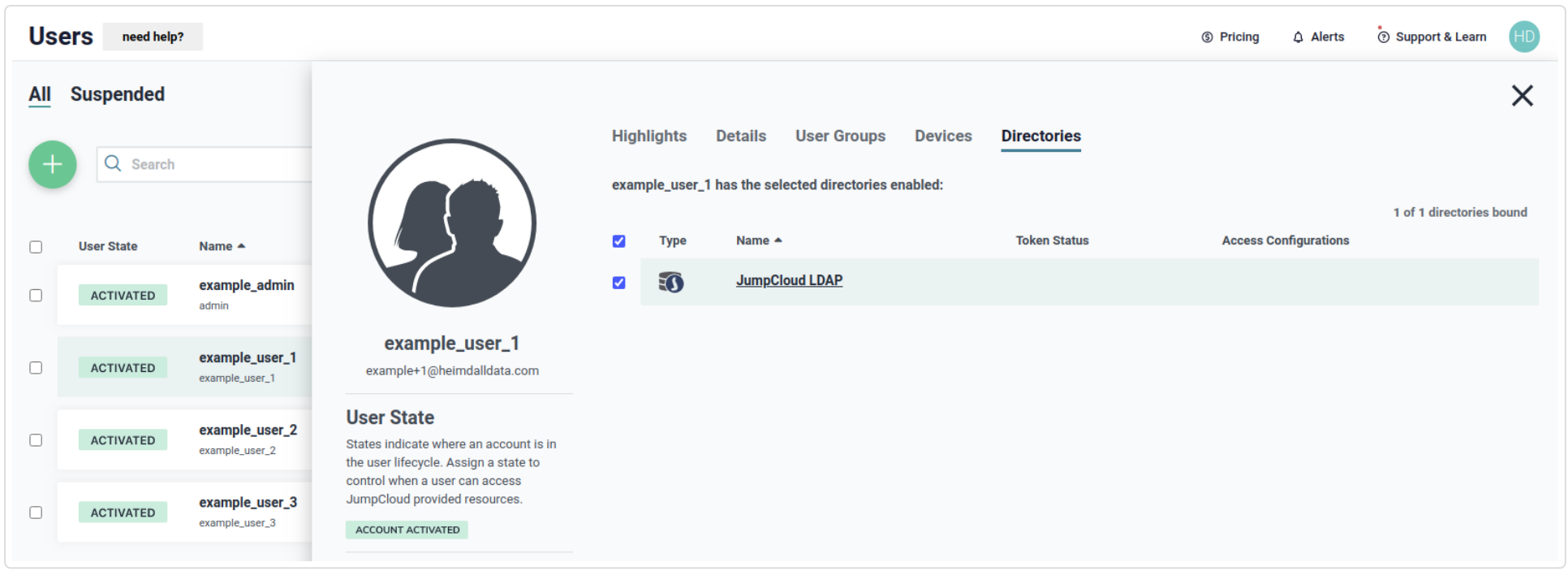


We need to be sure that the user belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.

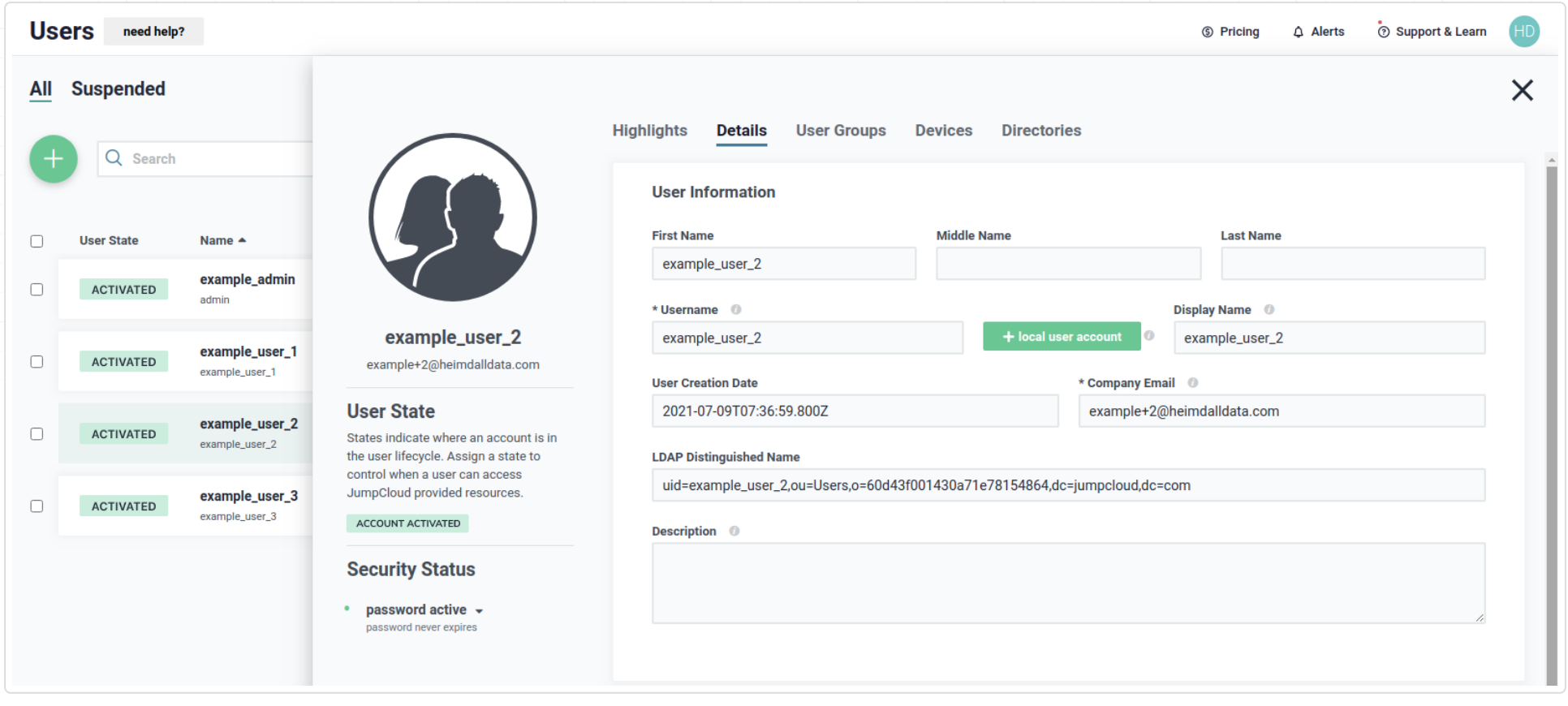


The next user, `example_user_1`, has a details as shown below.

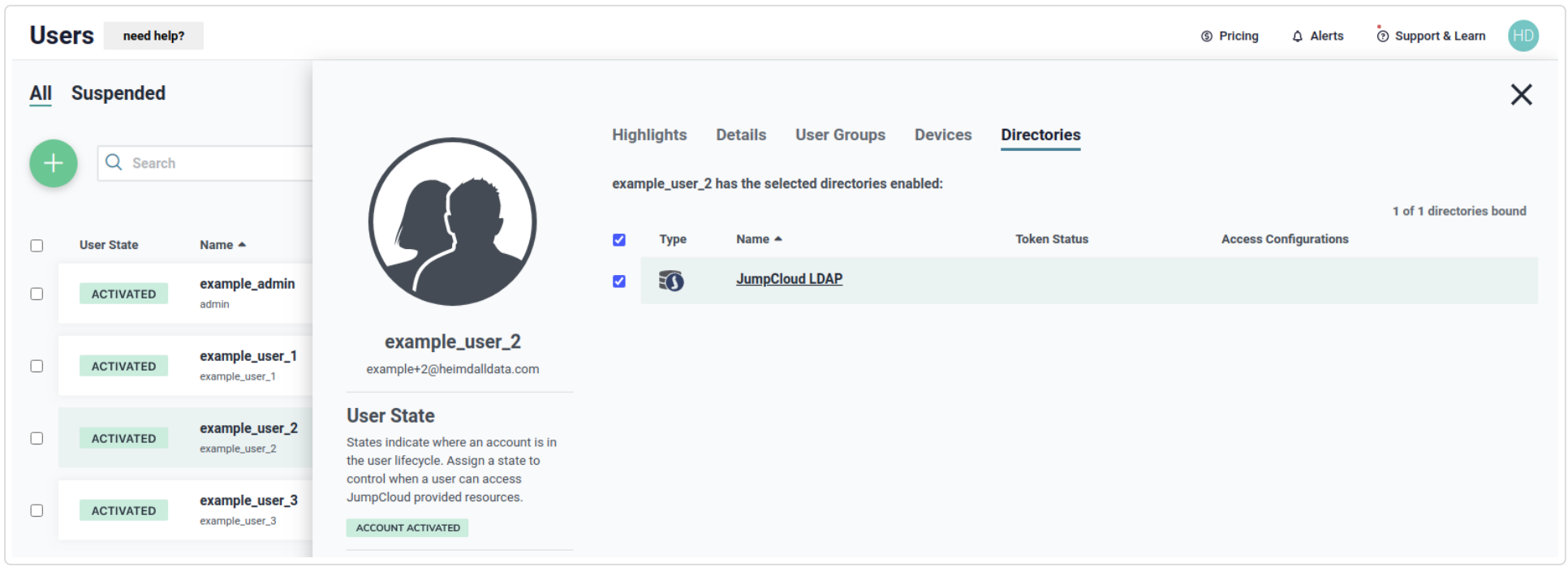
We need to be sure that the user belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.



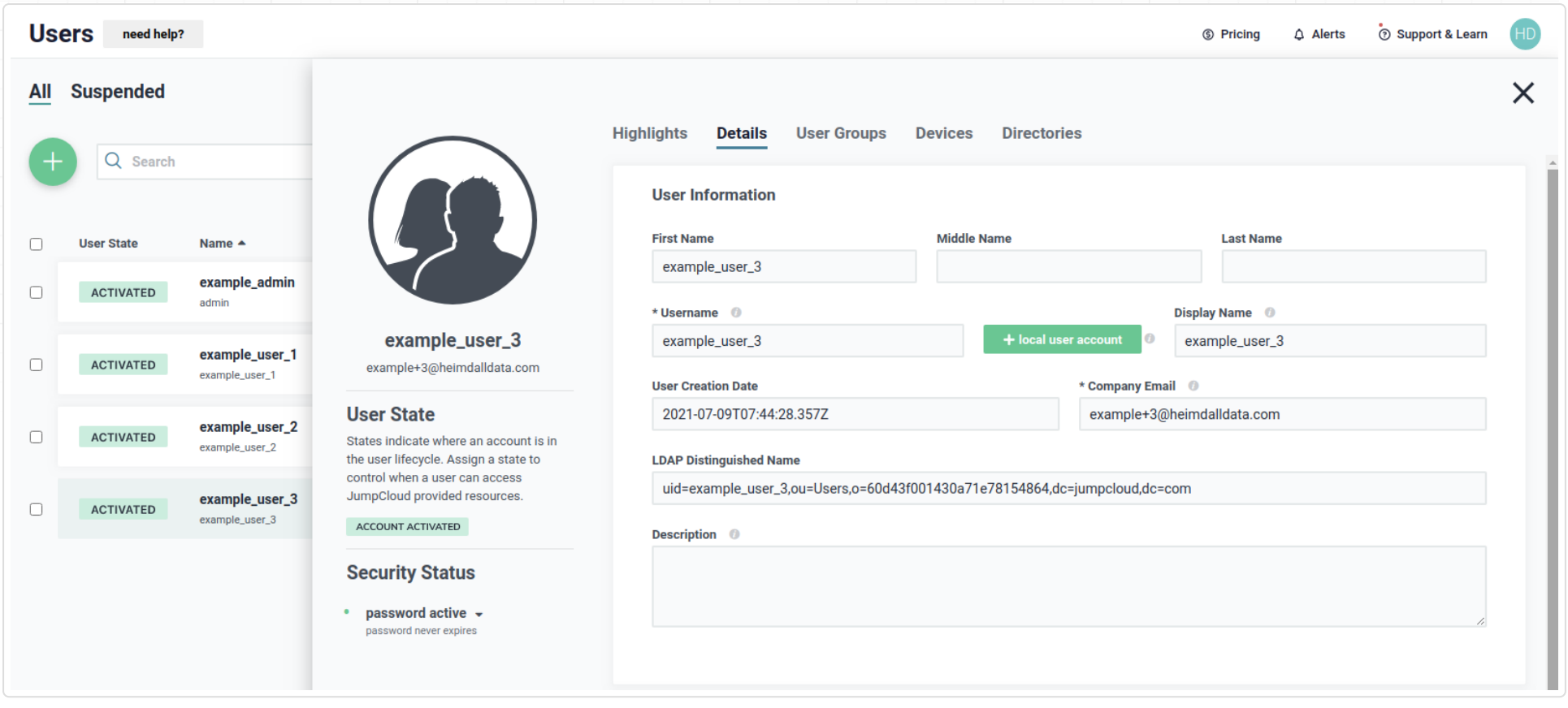
The next user, `example_user_2`, has a details as shown below.



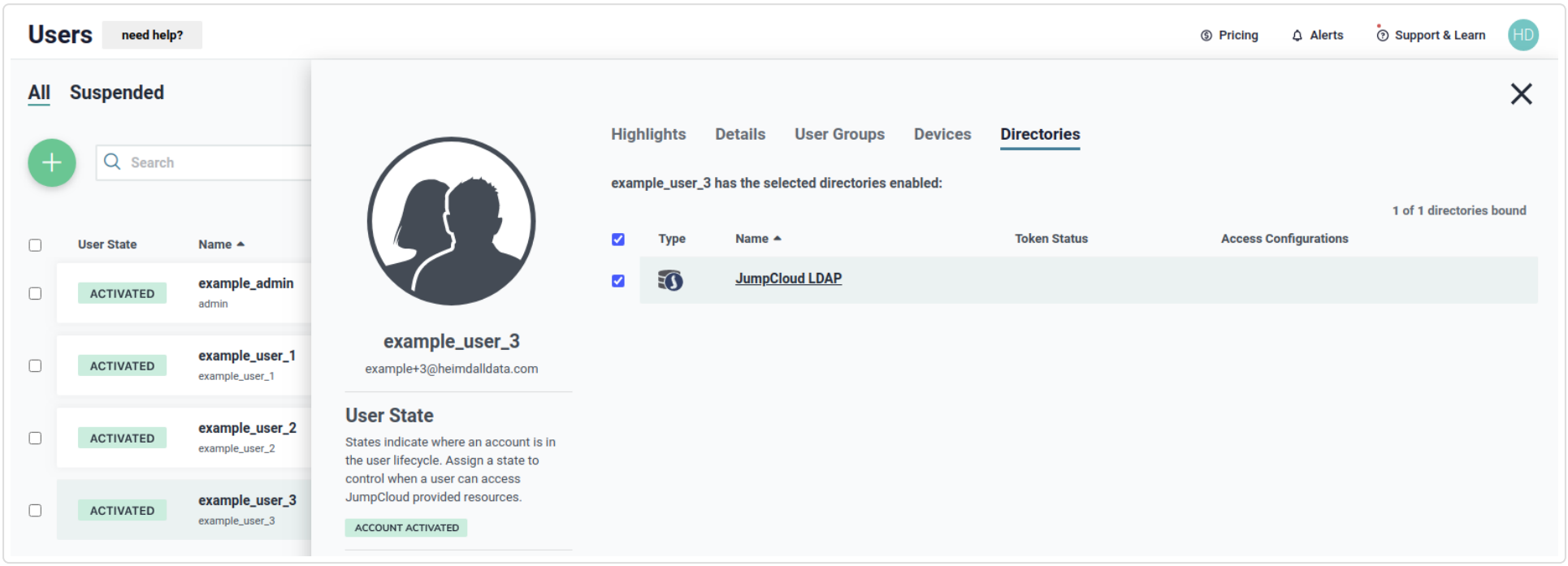
We need to be sure that the user belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.



The next user, `example_user_3`, has a details as shown below.

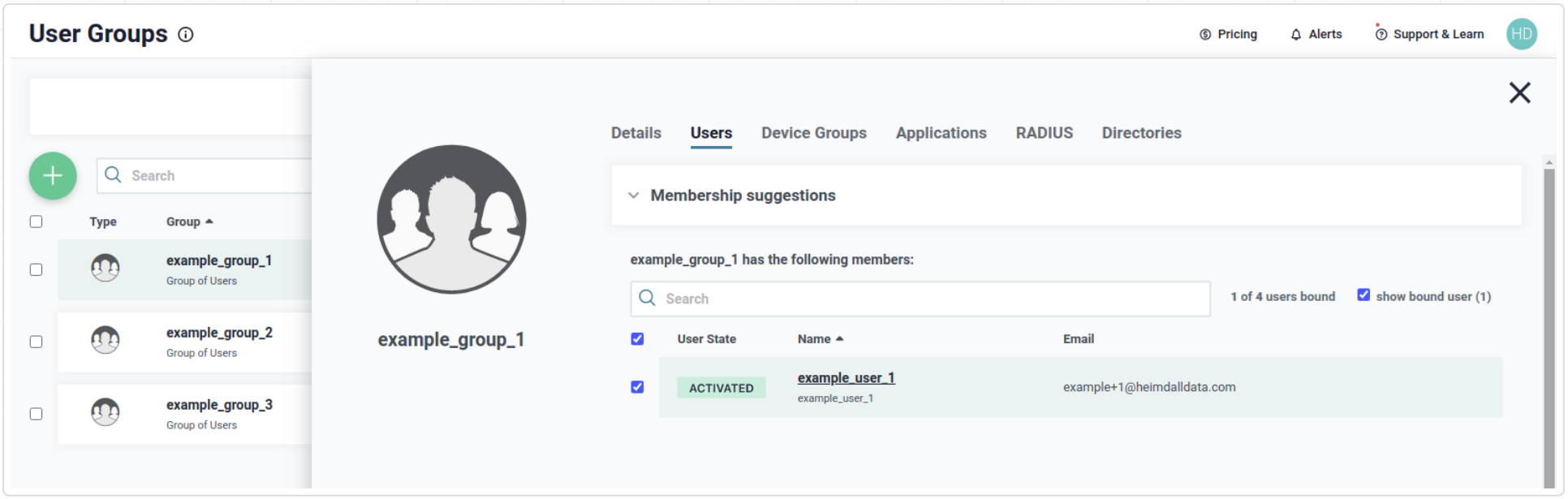


We need to be sure that the user belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.



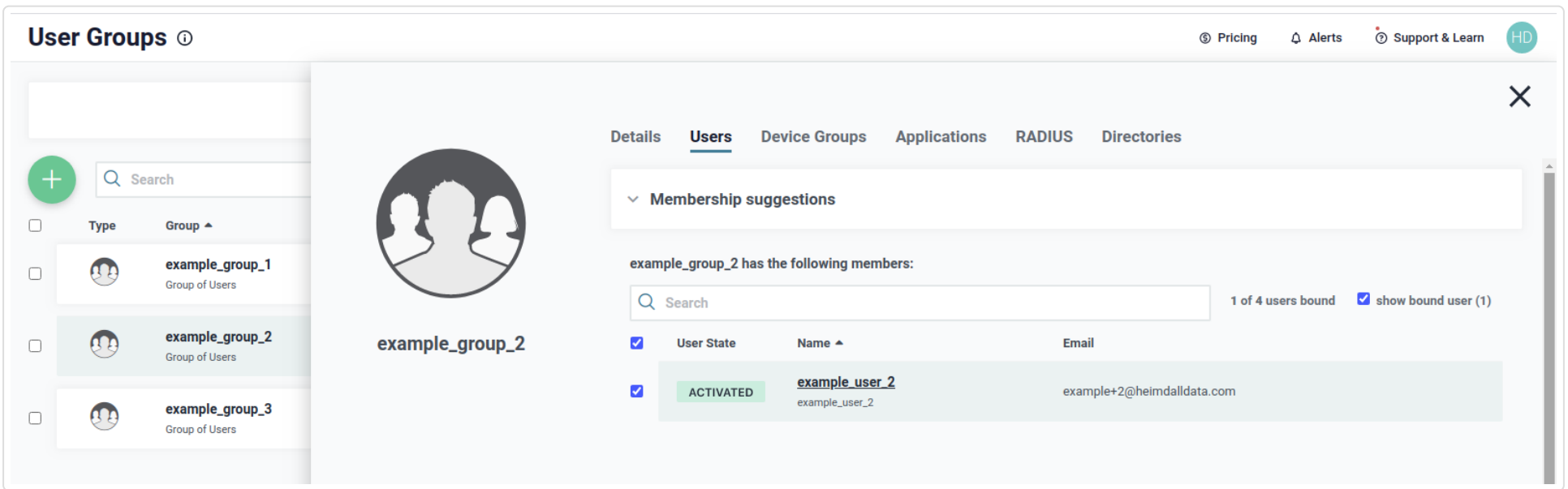
Next step of configuration is defining groups. For cases described here, three groups will be needed - `example_group_1`, `example_group_2` and `example_group_3`, as shown below.

First group is `example_group_1`. The group has only one member, which is `example_user_1`, as shown below.

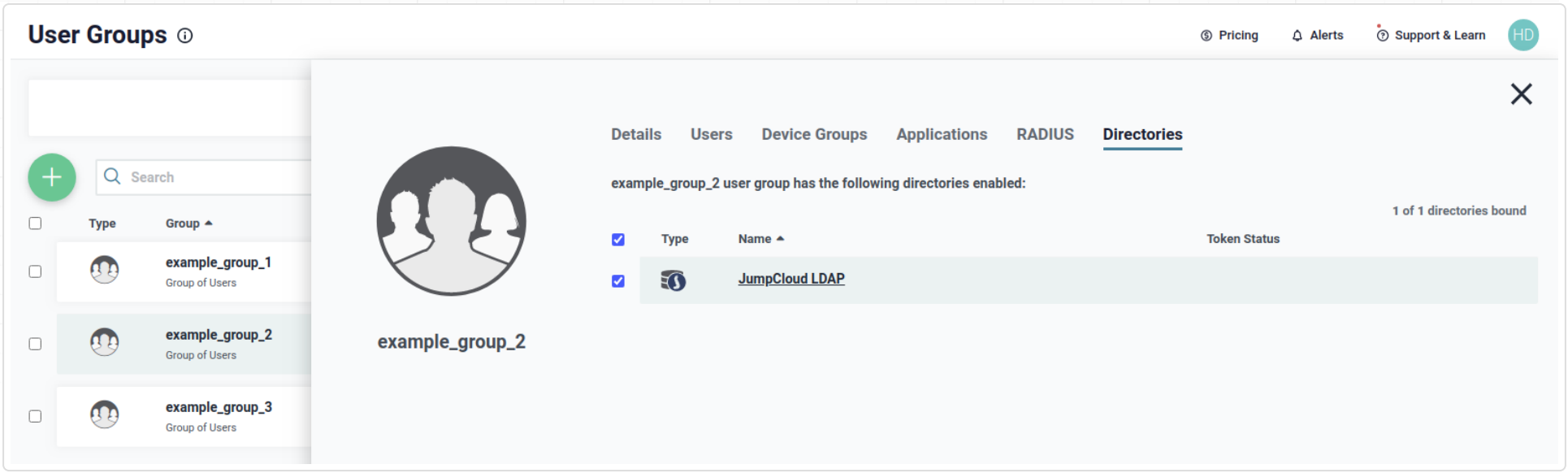


We need to be sure that the group belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.

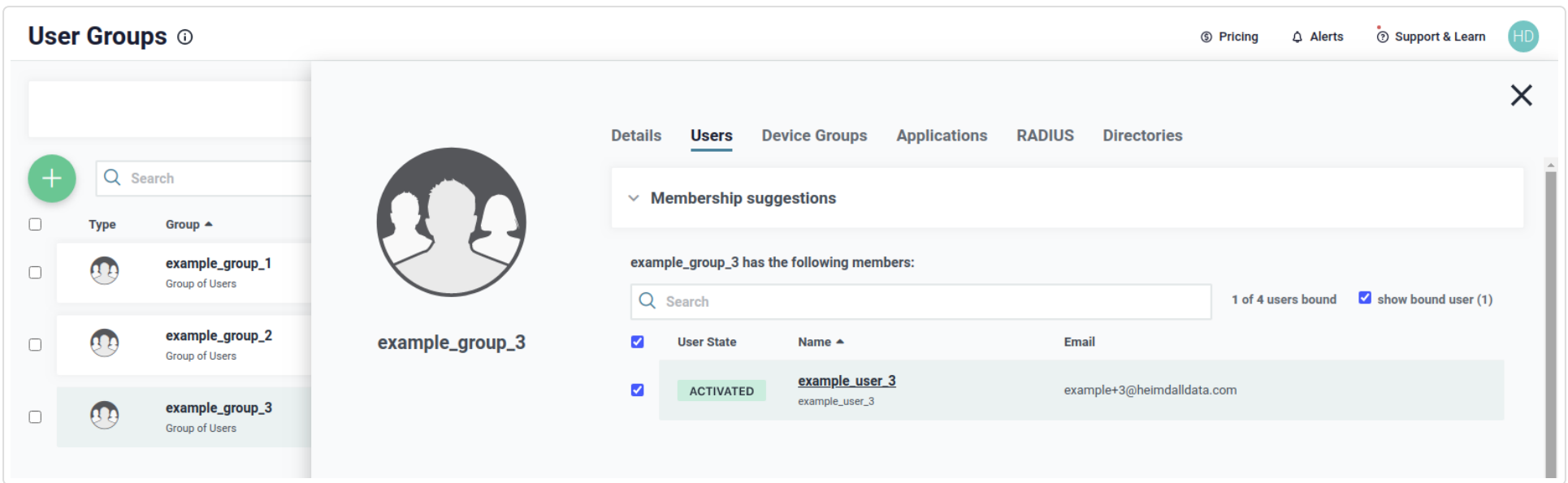
Next group is `example_group_2`. The group has only one member, which is `example_user_2`, as shown below.



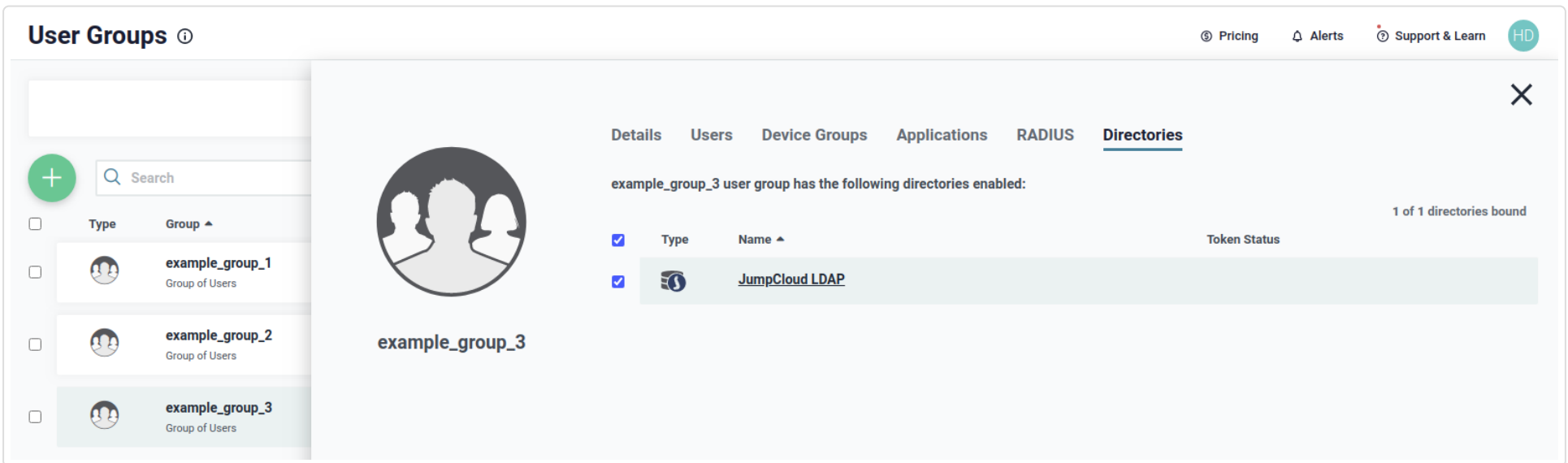
We need to be sure that the group belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.



Next group is `example_group_3`. The group has only one member, which is `example_user_3`, as shown below.



We need to be sure that the group belong to the JumpCloud LDAP directory, as shown below, to use it during authentication.



More information about our objects can be retrieved by using tool named `ldapsearch`. By using the below line, the search will be done with:

- simple authentication instead of SASL (`-x`)
- using `uid=admin,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com` user as Distinguished Name to bind to LDAP directory (`-D "uid=admin,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com"`)
- with a prompt for simple authentication, instead specifying the password on the command line (`-W`)
- with URI specified to `ldap://ldap.jumpcloud.com:389` (`-H ldap://ldap.jumpcloud.com:389`)
- with searchbase `ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com` (`-b ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com`)
- with scope set to subtree (`-s sub`)

```
ldapsearch -x -D "uid=admin,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com" -W -H ldap://ldap.jumpcloud.com:389 -b "ou=User
```

Information about objects will give us details about attributes of each object. That information will be helpful during configuration of authentication.

Details about user `example_user_1` are as shown below.

```
# example_user_1, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: uid=example_user_1,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: shadowAccount
objectClass: posixAccount
objectClass: jumpcloudUser
uid: example_user_1
uidNumber: 5003
homeDirectory: /home/example_user_1
mail: example+1@heimdalldata.com
sn: example_user_1
gidNumber: 5003
loginShell: /bin/bash
givenName: example_user_1
cn: example_user_1
displayName: example_user_1
memberOf: cn=example_group_1,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Details about user `example_user_2` are as shown below.

```
# example_user_2, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: uid=example_user_2,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
uidNumber: 5004
homeDirectory: /home/example_user_2
mail: example+2@heimdalldata.com
givenName: example_user_2
sn: example_user_2
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: shadowAccount
objectClass: posixAccount
objectClass: jumpcloudUser
uid: example_user_2
gidNumber: 5004
loginShell: /bin/bash
cn: example_user_2
displayName: example_user_2
memberOf: cn=example_group_2,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Details about user `example_user_3` are as shown below.

```
# example_user_3, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: uid=example_user_3,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
displayName: example_user_3
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: shadowAccount
objectClass: posixAccount
objectClass: jumpcloudUser
uidNumber: 5005
gidNumber: 5005
mail: example+3@heimdalldata.com
givenName: example_user_3
uid: example_user_3
loginShell: /bin/bash
homeDirectory: /home/example_user_3
sn: example_user_3
cn: example_user_3
memberOf: cn=example_group_3,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Details about group `example_group_1` are as shown below.

```
# example_group_1, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: cn=example_group_1,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
ou: example_group_1
objectClass: top
objectClass: groupOfNames
description: tagGroup
cn: example_group_1
member: uid=example_user_1,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Details about group `example_group_2` are as shown below.

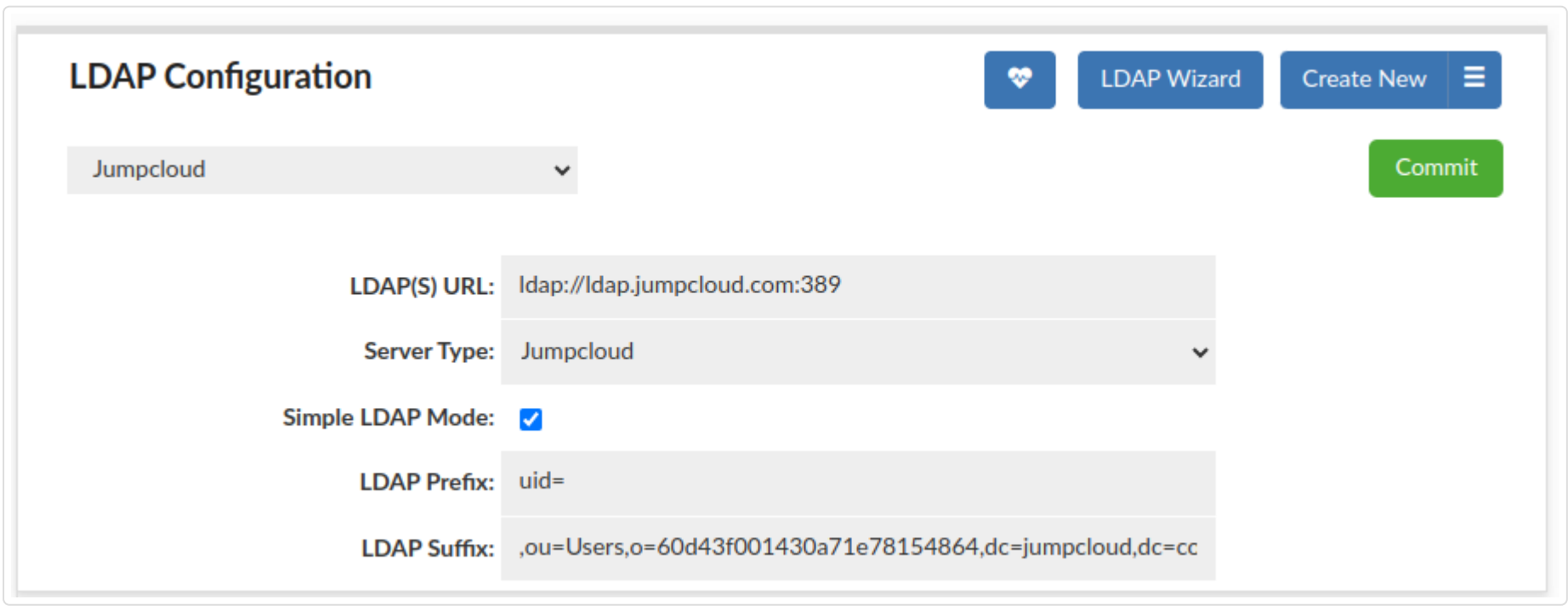
```
# example_group_2, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: cn=example_group_2,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
cn: example_group_2
ou: example_group_2
objectClass: top
objectClass: groupOfNames
description: tagGroup
member: uid=example_user_2,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Details about group `example_group_3` are as shown below.

```
# example_group_3, Users, 60d43f001430a71e78154864, jumpcloud.com
dn: cn=example_group_3,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
cn: example_group_3
ou: example_group_3
objectClass: top
objectClass: groupOfNames
description: tagGroup
member: uid=example_user_3,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com
```

Simple user authentication

To begin, let's assume that we want to give access to all users, which we defined in JumpCloud's LDAP. Additionally, we don't care if a user belong to a group or not. In this case, on the Heimdall Proxy's side, to authenticate a user, without looking at groups to which the user belongs, the best is LDAP Simple Mode authentication. An example of configuration to set proper authentication can be seen below.

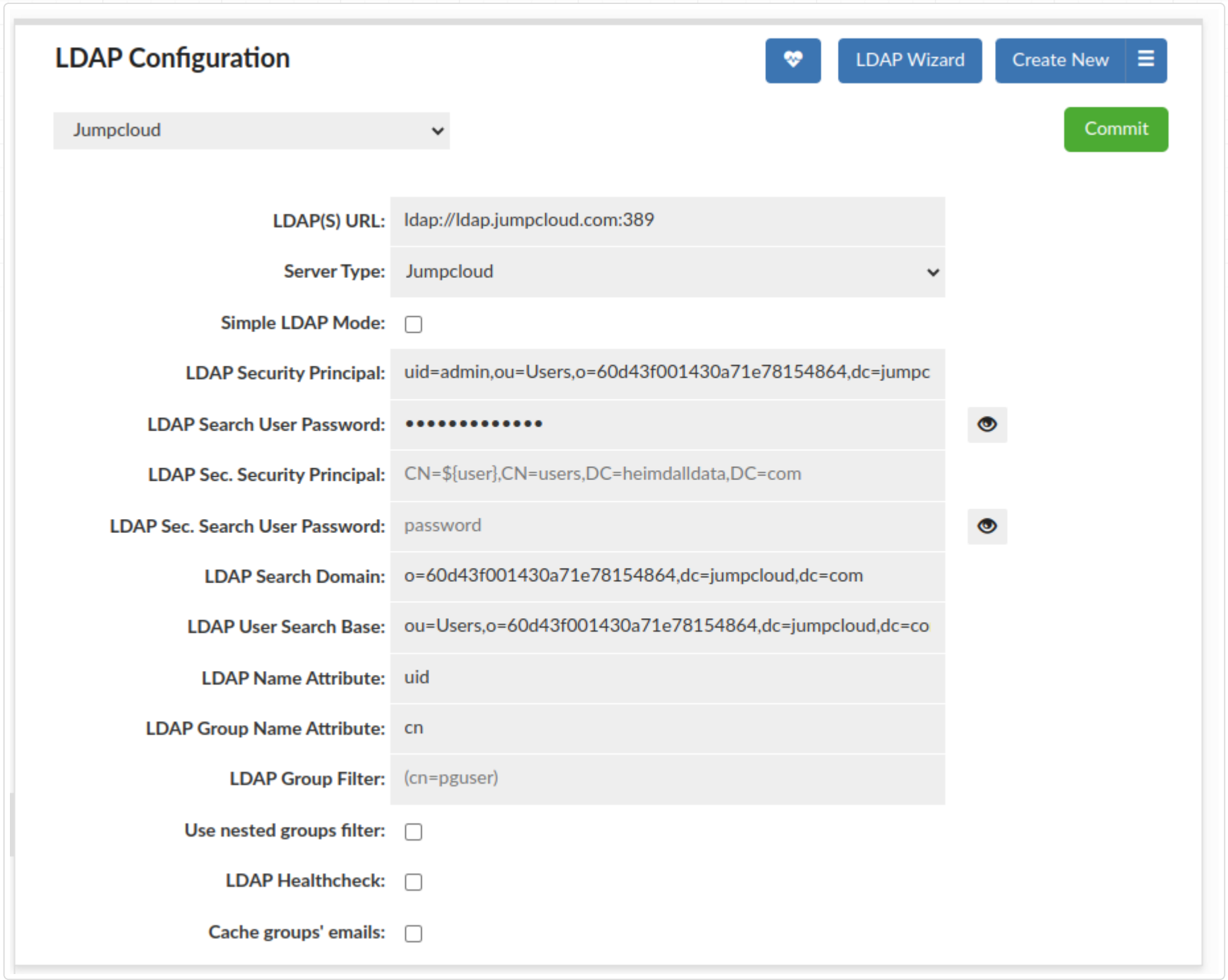


As can be seen, the configuration specifies only two values: LDAP Prefix and LDAP Suffix. Both of them, with username of a user connecting to a proxy, will build a string, which is going to be used to connect to a JumpCloud LDAP. That single step is enough to perform an authentication in Simple Mode.

With above configuration, all users, so `example_user_1`, `example_user_2` and `example_user_3`, will be authenticated successfully (assuming that proper credentials, so username and password, were given).

Authenticating user from a group

A more complex case than previous one, would be a situation, when we care if a user belong to a group, but we don't care what group it is. For that situation the best will be LDAP Bind+Search Mode authentication without a group filter. An example of configuration to set proper authentication can be seen below.



The configuration for Bind+Search Mode is more complex than for Simple Mode, so we are going to explain each property.

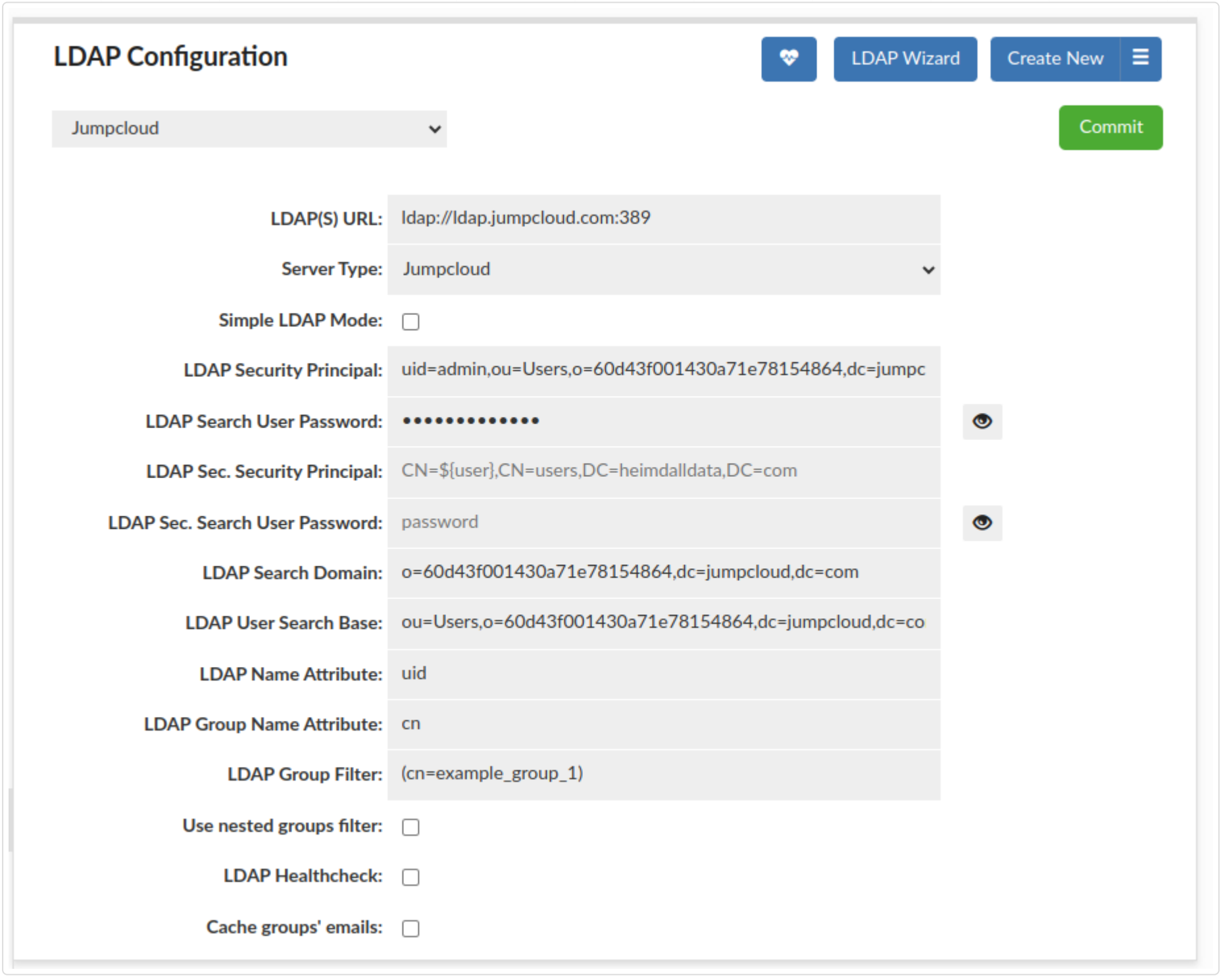
Ldap property	Value	Explanation
LDAP(S) URL	ldap://ldap.jumpcloud.com:389	Specifies URL of JumpCloud LDAP. In the example specified to a port without using SSL, but can be configured to an address using it.
LDAP Security Principal	uid=admin,ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com	Specifies user with enabled bind and search in JumpCloud LDAP service. For our case, it is a LDAP Distinguished Name of `example_admin` user.
LDAP Search User Password	(hidden)	Specifies password to authenticate as user specified in LDAP Security Principal. For our case, it is a password for `example_admin` user.

Ldap property	Value	Explanation
LDAP Search Domain	o=60d43f001430a71e78154864,dc=jumpcloud,dc=com	Specifies domain in which a user's group will be searched for. For our case, the value is as shown, without declaring any `ou`, because search will be done with subtree scope.
LDAP User Search Base	ou=Users,o=60d43f001430a71e78154864,dc=jumpcloud,dc=com	Specifies base for searching a user to be authenticated. For our case, the value is as shown, with declaring `ou` as `Users`, because we will be looking for users.
LDAP Name Attribute	uid	Specifies attribute, which should be used to identify a user, so if value of the attribute is equal to a username connecting to a proxy. The value for our case is based on reads for users objects from `ldapsearch` results.
LDAP Group Name Attribute	cn	Specifies attribute, which should be used to extract information about a group, so the name of a group. For our case, we read from `ldapsearch` results that objects of group don't have `uuid` attribute, so we need to specify a different one. From `ldapsearch` result can be read, that `cn` value is unique for our groups, so we are going to use it to authenticate them.
LDAP Group Filter	(empty)	Specifies group filter, which will be added during a search for a user's groups. For this example we don't need to specify it. When this attribute is specified, filter is added during searching for a user's groups and the filter can specify what restrictions have to be met by a group, e.g. value of an attribute of a group object.
Use nested groups filter	false	At the time of writing this section (8th July 2021), JumpCloud LDAP doesn't handle nested groups, so that option have to be turned off, because turned on can break authentication and return no results.

With above configuration, all users, so `example_user_1`, `example_user_2` and `example_user_3`, will be authenticated successfully (assuming that proper credentials, so username and password, were given). That will happen, because all of these users belongs to groups (user `example_user_1` is in group `example_group_1`, user `example_user_2` is in group `example_group_2`, user `example_user_3` is in group `example_group_3`). If there was a user without a group, that user would be rejected by authentication.

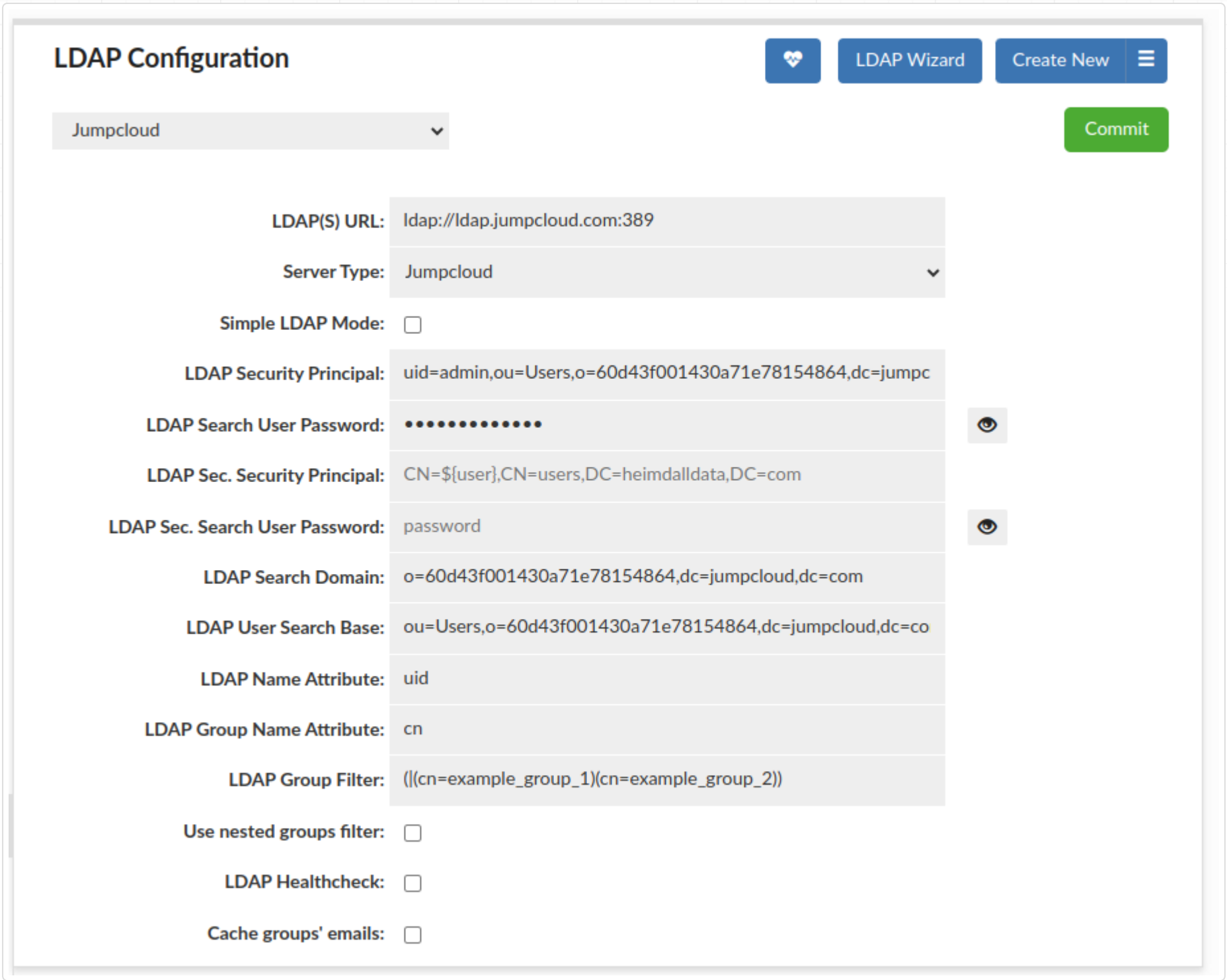
Authenticating user from a specified group

The most complex is when we want only to authenticate users from particular groups. First, let's assume that we only want to authenticate users from `example_group_1` group. For that situation the best will be to use LDAP Bind+Search Mode authentication with a group filter. An example of the configuration to accept users only from `example_group_1` group, can be seen below.



The configuration differs from a previous one by set value of LDAP Group Filter which specifies what value of `cn` attribute a group must have. Based on details read by using `ldapsearch` before, we know that `cn` value is unique between groups, so can be used for authenticating them. For `example_group_1` group, that value is `example_group_1`, so that's why group filter is `(cn=example_group_1)`. Using that configuration, only user `example_user_1` will be authenticated (assuming that proper credentials, so username and password, were given). For user `example_user_2` and `example_user_3` authentication will end up with a failure, during checking their groups.

Next situation is when we want to authenticate users from `example_group_1` or `example_group_2`. With that case we want to be sure that a user belong to any of these groups two groups. An example of configuration to accept users only from `example_group_1` or `example_group_2` group, can be seen below.



Again, the only change is in value of LDAP Group Filter. Set filter specifies that `cn` value of a group has to match `example_group_1` or `example_group_2`, so only groups with these values will be used, and that mean that only users from these groups will be authenticated. For our case, only `example_user_1` and `example_user_2` will be authenticated (assuming that proper credentials, so username and password, were given). For user `example_user_3` authentication will end up with a failure, during checking their groups.

The example of the LDAP Group Filter can be extended to set restrictions on more arguments as needed. The main rule is that filter has to met requirements of LDAP filters used to search. The second rule is that group filter has to be in brackets. If group filter exists, is joined, by using AND operator, to filter checking if a user is a member of a group.

Odoo Specific Information

Odoo Transaction Behavior

Odoo operates by putting every request in a transaction, and further, specifies "repeatable read" as part of this. It is this transaction behavior that actually causes much of the overhead of Odoo on the database, and as a result, Heimdall is not able to do caching or read/write split by default, so removing the majority of benefits that Odoo users desire.

In order to resolve this, Heimdall provides the option for "delayed transactions", in the VDB advanced tab. The way this works is to trap when a transaction is desired, and actually NOT start the transaction then. If a DML (insert/update/delete/etc) is performed, then the transaction is started, and used until the transaction either commits or is rolled back. Technically, this behavior is in violation of the transaction "repeatable read" specification, in that even reads before a DML should read consistent data, but with this option, only after a DML is this correct. This has not been observed to cause issues with customers. If this is a concern, please note that NO solution can provide caching and/or read/write split without violating this in some way. Other customers are using this successfully without it causing errors, but if you have custom Odoo modules that depend on strict repeatable read behavior, please contact Heimdall support. We do have ways to change behaviors via rules so that if an issue is identified, we can modify the behavior for the queries in question, while providing benefit for others.

Configuration Items for Odoo

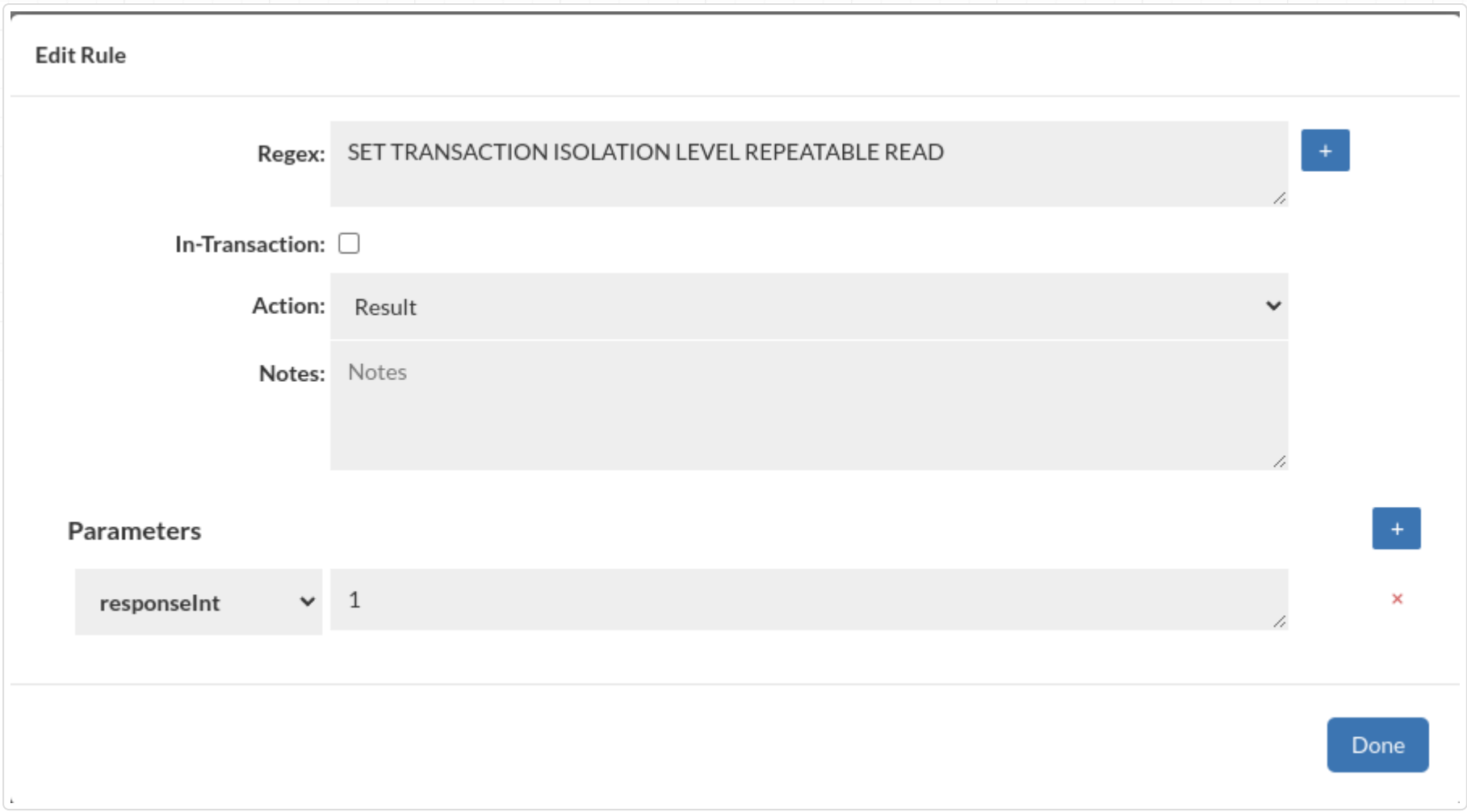
You can use a ready-made template, which should be selected in the wizard, or you can do it manually. If so, follow the instructions below.

First, configure the data source using the Wizard and verify connectivity. Then following these additional steps:

- In the postgresql.conf, set "default_transaction_isolation = 'repeatable read'" to set the default isolation level to repeatable read by default, avoiding the need for this to be set on connections. If you don't have access to the postgresql.conf or can't set it globally, you can alternatively set it at the database level with:

```
ALTER DATABASE SET DEFAULT_TRANSACTION_ISOLATION TO 'repeatable read';
```

- In the VDB, for Odoo, specify in the Advanced tab:
    - Multiplex with a timeout of 0
    - Delayed Transactions
- In the Data source tab:
    - Make sure the catalog field in the JDBC url is filled in with ${catalog}
    - In connection properties the defaultCatalog parameter is specified, generally with "odoo" as the default
    - Connection pooling is enabled
- In the rules tab, besides the normal caching and read/write split rules as needed, a rule with regex of "SET TRANSACTION ISOLATION LEVEL REPEATABLE READ", intrans NOT set, Action of Result, and responseInt of "1". This rule will prevent the commands to set the transaction isolation level from being sent to the server, which with delayed transactions will result in the server generating warnings in it's logs.



Internally, Heimdall will capture certain types of queries that appear to be operating against sequences (which Odoo does) and will automatically flag these as potential DML operations. This will be noted in the Analytics by some queries not getting cached that may otherwise appear cachable. Please see the help section to debug caching for information on how to debug why a query is not cached.

Cache Validation

if there is concern about if stale data is being returned from the cache, Heimdall includes a feature to validate cache data, and report if it was stale. To do this, on the cache rule, specify the property of "revalidateOnly" and set to "true". This will bypass serving from the cache, but will still cache objects and compare the cached object with what was previously there. In the event an object is not byte-for-byte identical, then an alert will be generated on the GUI with the query that generated the issue. This can be used to generate cache bypass rules if there are issues.