



# UnboundID

## UnboundID® Metrics Engine

Administration Guide

Version: 5.2.0.1

UnboundID Corp  
13809 Research Blvd., Suite 500  
Austin, Texas 78750  
Tel: +1 512.600.7700  
Email: [support@unboundid.com](mailto:support@unboundid.com)

---

---

# Copyright

---

Copyright © 2016 UnboundID Corporation

All rights reserved

This document constitutes an unpublished, copyrighted work and contains valuable trade secrets and other confidential information belonging to UnboundID Corporation. None of the foregoing material may be copied, duplicated, or disclosed to third parties without the express written permission of UnboundID Corporation.

This distribution may include materials developed by third parties. Third-party URLs are also referenced in this document. UnboundID is not responsible for the availability of third-party web sites mentioned in this document. UnboundID does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. UnboundID will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources. UnboundID and the UnboundID Logo are trademarks or registered trademarks of UnboundID Corp. in the United States and foreign countries. All other marks referenced are those of their respective owners.

---

# Table of Contents

---

<b>Copyright</b> .....	<b>i</b>
<b>Preface</b> .....	<b>viii</b>
About UnboundID .....	viii
Audience .....	ix
Documentation Included with the Metrics Engine .....	ix
Metrics Reference Documentation .....	ix
Related Documentation .....	x
<b>Chapter 1: Introduction</b> .....	<b>1</b>
Metrics Engine Overview .....	2
Metrics Engine Components .....	2
Data Collection .....	2
Performance Data .....	3
System and Status Data .....	3
Charts and Dashboards .....	4
PostgreSQL DBMS Details .....	4
<b>Chapter 2: Installing the Metrics Engine</b> .....	<b>5</b>
Supported Platforms .....	6
Install the JDK .....	6
Configure a Non-Root User .....	6
Optimize the Solaris Operating System .....	6
Restrict ZFS Memory Consumption .....	7
Limit ZFS Transaction Group Writes .....	7
Configure ZFS Access to Underlying Disks .....	7
Configure ZFS Compression .....	7
Optimize the Linux Operating System .....	7
Set the File Descriptor Limit .....	8
Set the Filesystem Flushes .....	8
Install sysstat and pstack on Red Hat .....	9
Install the dstat Utility .....	9
Disable Filesystem Swapping .....	9
Managing System Entropy .....	9
Configure Identity Servers to be Monitored .....	9
Disk Space Requirements and Monitoring Intervals .....	10

## Table of Contents

---

Tracked Applications .....	10
Install the Server .....	11
Server Folders and Files .....	13
Add Monitored Servers to the Metrics Engine .....	14
Using the monitored-servers Tool .....	14
Using the dsconfig Tool .....	15
Start and Stop the Server .....	15
Start the Metrics Engine as a Background Process .....	16
Start the Metrics Engine as a Foreground Process .....	16
Start the Metrics Engine at Boot Time .....	16
Stop the Metrics Engine .....	16
Restart the Metrics Engine .....	17
Uninstall the Server .....	17
Install the Web Console .....	17
Configure the Web Console .....	19
Configure SSL or StartTLS for the Console .....	20
Configure a Truststore for the Console .....	20
Log Into the Console .....	20
Upgrade the Web Console .....	21
Uninstall the Console .....	21
<b>Chapter 3: Managing the Metrics Engine .....</b>	<b>23</b>
Metrics Engine Error Logging .....	24
Logging Retention Policies .....	24
Logging Rotation Policies .....	24
Create Log Publishers .....	24
Error Log Publisher .....	25
Backend Monitor Entries .....	26
Disk Space Usage Monitor .....	28
Notifications and Alerts .....	28
Configure Alert Handlers .....	29
The Alerts Backend .....	29
View Information in the Alerts Backend .....	30
Modify the Alert Retention Time .....	30

---

---

Configure Duplicate Alert Suppression .....	31
System Alarms, Alerts, and Gauges .....	31
Testing Alerts and Alarms .....	32
To Test Alarms and Alerts .....	32
Back Up the Metrics Engine Database .....	33
Historical Data Storage .....	34
Planning the Backup .....	34
Start the DBMS Backup .....	35
Restore a DBMS Backup .....	35
Management Tools .....	35
Available Command-Line Utilities .....	36
The tools.property File .....	37
Tool-Specific Properties .....	38
Specify Default Properties Files .....	38
Using the Configuration API .....	39
Authentication and Authorization .....	39
Relationship Between the Configuration API and the dsconfig Tool .....	39
API Paths .....	48
Sorting and Filtering Configuration Objects .....	49
Updating Properties .....	49
Administrative Actions .....	51
Updating Servers and Server Groups .....	52
Configuration API Responses .....	52
Domain Name Service (DNS) Caching .....	53
IP Address Reverse Name Lookups .....	54
Server SDK Extensions .....	54
<b>Chapter 4: Collecting Data and Metrics .....</b>	<b>56</b>
Metrics Overview .....	57
Count Metrics .....	57
Continuous Metrics .....	57
Discrete Metrics .....	57
Dimensions .....	58
Query Overview .....	59
Select Query Data .....	60
Aggregate Query Results .....	60

## Table of Contents

---

Format Query Results .....	61
The query-metric Tool .....	61
Performance Data Collection .....	63
System Monitoring Data Collection .....	64
Stats Collector Plugin .....	64
System Utilization Monitors .....	65
External Collector Daemon .....	65
Server Clock Skew .....	65
Tune Data Collection .....	66
Reducing the Data Collected .....	66
Reducing the Frequency of Data Collection .....	66
Reducing the Frequency of Sample Block Creation .....	66
Reducing Metrics Engine Impact on Performance .....	67
Data Processing .....	67
Importing Data .....	67
Aggregating Data .....	68
Monitoring for Service Level Agreements .....	68
SLA Thresholds .....	69
Threshold Time Line .....	71
Configure an SLA Object .....	71
<b>Chapter 5: Configuring Charts and Dashboards .....</b>	<b>74</b>
Available Dashboards .....	75
Customize the LDAP Dashboard .....	78
Debug Dashboard Customization .....	78
Preserve Customized Files .....	79
The Chart Builder Tool .....	79
Chart Presentation Details .....	80
Chart Builder Parameters .....	81
Chart Properties File .....	82
Available Charts for Identity Servers .....	82
Charts for All Servers .....	82
Data Store Charts .....	83
Proxy Server Charts .....	83

---

---

Data Sync Server Charts .....	83
Metrics Engine Server Charts .....	83
Data Broker Charts .....	84
Velocity Templates .....	84
Supporting Multiple Content Types .....	86
Velocity Context Providers .....	87
Velocity Tools Context Provider .....	88
<b>Chapter 6: Troubleshooting .....</b>	<b>89</b>
Using the collect-support-data Tool .....	90
Delays in Sample Data Availability .....	90
Slow Queries Based on Sample Cache Size .....	91
Performance Troubleshooting Example .....	92
Insufficient Memory Errors .....	95
Unexpected Query Results .....	96
Installation and Maintenance Issues .....	96
The setup Program will not Run .....	96
The Server will not Start .....	97
The Server has Shutdown .....	100
The Server will not Accept Client Connections .....	100
The Server is Unresponsive .....	101
Problems with the Web Console .....	101
<b>Chapter 7: Metrics Engine API Reference .....</b>	<b>103</b>
Connection and Security .....	104
Secure Error Messages .....	105
Response Codes .....	105
List Monitored Instances .....	105
EXAMPLES .....	106
Retrieve Monitored Instance .....	107
EXAMPLE: .....	107
List Available Metrics .....	108
EXAMPLES .....	109
Retrieve a Metric Definition .....	110
EXAMPLE .....	110
Perform a Metric Query .....	111
Data Set Structure .....	113

---

## Table of Contents

---

Chart Image .....	115
Google Chart Tools Datasource Protocol .....	116
Access Alerts .....	118
Retrieving Event Types .....	118
Retrieving Events .....	118
LDAP SLA .....	119
Retrieving the SLA Object .....	119
EXAMPLE .....	120
Pagination .....	122
<b>Index .....</b>	<b>123</b>



---

# Preface

---

The Metrics Engine Administration Guide provides procedures to install and manage the Metrics Engine in a multi-client environment.

## About UnboundID

UnboundID Corp Corp is a leading identity infrastructure domain solutions provider with proven experience in large-scale environments. The UnboundID Platform provides a unified view of customer data across all applications, channels, partners, and lines of business.

The UnboundID Platform provides the following:

- **Secure End-to-End Customer Data Privacy Solution** – A comprehensive identity data platform with authorization and access controls to enforce privacy policies, control user consent, and manage resource flows.
- **Purpose-Built Platform** – Solutions to consolidate, secure, and deliver customer consent-given identity data. The system provides security measures to protect sensitive identity data and maintain its visibility. The broad range of platform services include, policy management, cloud provisioning, federated authentication, data aggregation, and directory services.
- **Performance across Scale and Breadth** – Support for the three pillars of performance-at-scale: users, response time, and throughput. The system manages real-time data at large-scale consumer facing service providers.
- **Support for External APIs** – Standards-based solutions that can interface with various external APIs to access a broad range of services. APIs include XACML 3.0, SCIM, LDAP, OAuth2, and OpenID Connect.

## Audience

This guide is intended for administrators who are responsible for installing and managing servers in an identity enterprise environment. Knowledge of the following is recommended:

- Identity platforms and LDAP concepts.
- General system administration and networking practices.
- Java VM optimization.
- Application performance monitoring.
- Statistics and metrics basics.

## Documentation Included with the Metrics Engine

The following documents are installed with the Metrics Engine:

- *UnboundID Metrics Engine Administration Guide (PDF)* – provides installation, administration, and management tasks for the Metrics Engine.
- *UnboundID Security Guide* – provides security-specific information for UnboundID servers.
- *UnboundID Metrics Reference (HTML)* – provides information about the metrics collected by the Metrics Engine.
- *UnboundID Metrics Configuration Reference (HTML)* – provides information about the configuration options available for the Metrics Engine server.
- *UnboundID Metrics Command-Line Tool Reference (HTML)* – provides information about each of the command-line tools available with the Metrics Engine and their options and use.
- *UnboundID Metrics Engine Release Notes (HTML)* – provides features and fixes included in this release.

## Metrics Reference Documentation

The Metrics Engine package contains online reference documentation that can be used to implement custom charts. Access the documentation at the following URL:

```
https://<metrics-engine-host>:<port>/docs/index.html
```

The Metrics Engine Documentation page provides links to a reference file that details every metric available per product, a Metrics Engine REST API documentation that explains the endpoints, and the Metrics Engine Chart Builder tool to customize any chart.

The page displays the following columns:

- **Name** – Provides a link to a given metric. Click a name to launch the Chart Builder tool and display a preview chart for that specific metric.
- **Produced By** – Indicates the UnboundID product source that is generating the metric.
- **Description** – Provides a brief description of the metric.
- **Dimensions** – Displays the type of data on the chart.
- **Statistics** – Displays the type of measurement taken for the metric.

The Metrics documentation page also provides a **Dimensions** tab, showing the type of dimensions available for a customized chart. This information is useful when configuring charts and dashboards. See [Configuring Charts for Identity Servers](#) for more information.

## Related Documentation

The following documents represent the rest of the UnboundID product set and may be referenced in this guide:

- *UnboundID Data Store Reference (HTML)*
- *UnboundID Data Store Administration Guide (PDF)*
- *UnboundID Security Guide (PDF)*
- *UnboundID Data Sync Server Reference Guide (HTML)*
- *UnboundID Data Sync Server Administration Guide*
- *UnboundID Proxy Server Reference (HTML)*
- *UnboundID Proxy Server Administration Guide (PDF)*
- *UnboundID Data Broker Reference (HTML)*
- *UnboundID Data Broker Administration Guide (PDF)*
- *UnboundID Data Broker Installation Guide (PDF)*
- *UnboundID Data Broker Application Developer Guide (PDF)*
- *UnboundID Security Guide (PDF)*
- *UnboundID LDAP SDK (HTML)*
- *UnboundID Server SDK (HTML)*

---

# Chapter 1: Introduction

---

The Metrics Engine collects performance data from the UnboundID Platform.

Topics include:

[Metrics Engine Overview](#)

[Metrics Engine Components](#)

[Data Collection](#)

[Charts and Dashboards](#)

[PostgreSQL DBMS Details](#)

## Metrics Engine Overview

The Metrics Engine provides insight into the transactions and performance of the UnboundID Platform. The Metrics Engine collects data from configured instances and replicas of the Data Store, the Proxy Server, the Data Sync Server, and the Data Broker Server. Data collected from the Metrics Engine enables:

- Measuring the performance of the identity infrastructure as a whole service, not a collection of individual servers.
- Identifying client applications that require the greatest amount of resources.
- Determining which servers have the most available resources to handle requests.
- Predicting the capacity and needs of the identity infrastructure to plan for increased traffic.
- Analyzing all aspects of the identity infrastructure for troubleshooting performance issues.

## Metrics Engine Components

The Metrics Engine consists of the following components:

**Metrics Engine** – A stand-alone server that relies on the PostgreSQL database for collected metrics. The Metrics Engine gathers data for itself and configured UnboundID servers.

**Metrics API** – A REST API that provides access to collected metrics data. The API is accessible over HTTPS and supports multiple management parameters including filtering, averaging, and setting ranges for multiple data sets.

**query-metric tool** – The primary command-line tool for metric data access. This tool can also be used for scripted automation of extracting data from the Metrics Engine. An explore option enables custom queries and additions to charts and dashboards.

**SNMP access** – System-level metrics can be accessed over SNMP.

**Data Set** – A proprietary data structure that is designed for interoperability with charting libraries such as Highcharts, FusionCharts, or JFreeChart.

**Charts, Chart Builder, and Dashboard Templates** – Tools for customizable, web-based metrics charts and dashboards.

## Data Collection

The Metrics Engine collects data from all monitored servers through LDAP queries to the server's backend. Each monitored server collects and stores a limited history of data locally. Data includes system status and performance information. To collect data, the Metrics Engine regularly polls all monitored servers for data that is stored in time-contiguous blocks, gathers the recent data, and stores data in a PostgreSQL database. Polling has minimal impact on the monitored servers.

## Performance Data

The majority of information collected represents the performance of the monitored server. Each monitored server should be configured to enable the Metrics Engine to adequately keep up with the flow. Performance data represents multiple dimensions of a metric. For example, a response time metric can represent the request type, time to respond, the application that made the request, and the action that was taken.

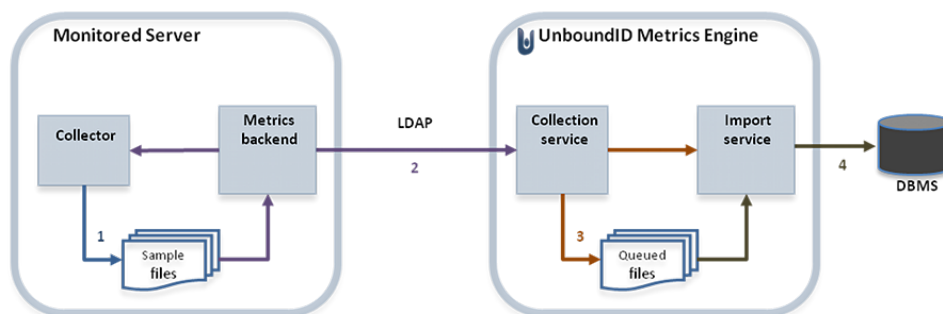
## System and Status Data

All servers configured to be monitored by the Metrics Engine store server and host system data. Server and machine metrics are retrieved from the `cn=monitor` backend of the monitored server.

The Stats Collector plugin is responsible for collecting performance data from the `cn=monitor` backend. Data includes server responses, replication activity, local database activity, and host system metrics. Stats Collector configuration defines:

- Data sample and collection intervals.
- The granularity of data collected (basic, extended, or verbose).
- The types of host system data collected such as CPU, disk, and network.
- The type of data aggregation that occurs for LDAP application statistics.

See [Tuning Data Collection](#) for more information. The following illustrates the data collection process:



### Data Collection

Data collection occurs in the following steps:

1. Data samples are taken and stored in time-contiguous blocks on the disk of the monitored server.
2. The Metrics Engine collection service polls for new sample blocks.
3. The new sample blocks are queued to disk on the Metrics Engine.
4. The Metrics Engine import service loads new blocks into the database.

## Charts and Dashboards

The Metrics Engine provides a number of charts and dashboards to display metrics information. A Chart Builder tool enables configuring charts on an HTML page and saving the properties for use in a dashboard. Several charts are provided for general system information and specific UnboundID Corp server functions. All dashboards are viewed from the Metrics Engine. The Data Broker dashboard can be surfaced in the Data Broker Console interface.

## PostgreSQL DBMS Details

The Metrics Engine uses a PostgreSQL DBMS to store data, which is included in the installation. This is a traditional table-based DBMS best suited for tabular data. The Metrics Engine interacts with the DBMS in four ways:

- **Data import** – Import places steady write load on the DBMS and accounts for 80% of the writes. This single-threaded interaction puts a lock on the target table. A Metrics Engine that monitors 20 servers keeps a single 10K RPM disk 70% busy with this single interaction.
- **Data aggregation** – Data aggregation places a less frequent read/write load on the DBMS. This interaction is responsible for the aggregation of the data samples from one time resolution to the next, reading from one set of tables and writing to another set. Sample aggregation uses no table-level locks and the ratio of records between read:write is between 60:1 and 24:1.
- **Data sample age-out** – Sample age-out occurs at regular intervals and results in a table being dropped and/or added. Age-out occurs every 30 minutes, though some intervals may drop and/or add more than one table.
- **Data query** – Sample queries occur when clients request metric samples from the public API. The API can aggregate multiple dimensions and multiple servers in a single request. A single request may fetch several million rows from the DBMS, though it only returns a few hundred data points to the client. Samples from previous queries are cached by the Metrics Engine, but initial queries for a given metric may take several seconds and result in a large amount of disk read activity.

Over time, the storage of samples in the data tables is optimized to match the access patterns of the queries. However, the public API supports queries where the results are the aggregate of thousands of different dimension sets, and each dimension set may have thousands of samples within the time range of the query. For example, a query about the throughput of all Data Store and Proxy Servers for all applications and all LDAP operations over the last 72 hours might result in four to six million DBMS records being read into memory, aggregated, and finally reduced to 100 data values. The results from each query are cached so that a subsequent request for the same data results in less DBMS activity. Both disk seek time and rotational delay impact the performance of a first-time query, so disks with faster RPM speeds provide a measurable improvement for first-time queries.

---

## Chapter 2: Installing the Metrics Engine

---

This section describes how to install and run the Metrics Engine. It includes pre-installation requirements and considerations.

Topics include:

[Supported Platforms](#)

[Installing Java](#)

[Creating a Non-Root user](#)

Configuring the Operating System for [Linux](#) and [Solaris](#)

[Configuring the Identity Servers to Gather Metrics](#)

[Installing the Metrics Engine](#)

[Server Folders and Files](#)

[Adding Monitored Servers to the Metrics Engine](#)

[Installing the Web Console](#)



## Supported Platforms

The Metrics Engine is a pure Java application. It is intended to run within the Java Virtual Machine on any Java Standard Edition (SE) or Enterprise Edition (EE) certified platform. For the list of supported platforms and Java versions, access the UnboundID Corp Customer Support Center portal or contact an UnboundID Corp authorized support provider.

### **Note**

It is highly recommended that a Network Time Protocol (NTP) system be in place so that multi-server environments are synchronized and timestamps are accurate.

## Install the JDK

The Java 64-bit JDK is required on the server. Even if Java is already installed, create a separate Java installation for use by the server to ensure that updates to the system-wide Java installation do not inadvertently impact the installation.

Solaris systems require both the 32-bit (installed first) and 64-bit versions. The 64-bit version of Java on Solaris relies on a number of files provided by the 32-bit installation.

## Configure a Non-Root User

The Metrics Engine installer cannot be run as the root user, and generally the Metrics Engine (and PostgreSQL) should not be run as root. As a non-root user, network port numbers below 1024 cannot be used.

In general, this account will need the ability to do the following:

- Listen on privileged network ports.
- Bypass restrictions on resource limits.

For security, the account should be restricted from the following:

- The ability to see processes owned by other users on the system.
- The ability to create hard links to files owned by other users on the system.

## Optimize the Solaris Operating System

UnboundID recommends the use of ZFS™, which is provided with Solaris systems. All of the server's components should be located on a single storage pool (zpool), rather than having separate pools configured for different server components. Multiple filesystems can be created inside the pool. ZFS's copy-on-write transactional model does not require isolating I/O-intensive components. Therefore, all available disks should be placed in the same zpool.

The following configurations should be made to optimize ZFS. Most configuration changes require a reboot of the machine.

## Restrict ZFS Memory Consumption

Database caching rather than filesystem caching is needed for performance. Configure the ZFS memory in the `etc/system` file to use no more than 2GB for caching, such as:

```
set zfs:zfs_arc_max= 0x80000000
```

This property sets the maximum size of the ARC cache to 2GB (0x80000000 or 2147483648 bytes).

## Limit ZFS Transaction Group Writes

To improve write throughput and latency, set the `zfs_write_limit_override` property in the `etc/system` file to the size of the available disk cache on the system. For example, for a system that has a 32MB cache per disk, set the following parameter:

```
set zfs:zfs_write_limit_override=0x2000000
```

## Configure ZFS Access to Underlying Disks

ZFS should be given direct access to the underlying disks that will be used to back the storage. In this configuration, the zpool used for the server should have a RAID 1+0 configuration (a stripe across one or more 2-disk mirrors). Although this setup can reduce the amount of available space when compared with other configurations, RAID 1+0 provides better performance and reliability.

## Configure ZFS Compression

ZFS should have compression enabled to improve performance. In most cases, the reduced costs of the disk I/O outweighs the CPU cost of compressing and decompressing the data. Turn on ZFS compression by running the `zfs` command:

```
# zfs set compression=on <zfs-filesystem-name>
```

The changes take effect without a machine reboot.

# Optimize the Linux Operating System

Configure the Linux filesystem by making the following changes.

### **Note**

The server explicitly overrides environment variables like `PATH`, `LD_LIBRARY_PATH`, and `LD_PRELOAD` to ensure that settings used to start the server do not inadvertently impact its behavior. If these variables must be edited, set values by editing the `set_environment_vars` function of the `lib/_script-util.sh` script. Stop and restart the server for the change to take effect.

## Set the File Descriptor Limit

The server allows for an unlimited number of connections by default, but is restricted by the file descriptor limit on the operating system. If needed, increase the file descriptor limit on the operating system with the following procedure.

### Note

If the operating system relies on `systemd`, refer to the Linux operating system documentation for instructions on setting the file descriptor limit.

1. Display the current hard limit of the system. The hard limit is the maximum server limit that can be set without tuning the kernel parameters in the `proc` filesystem.

```
ulimit -aH
```

2. Edit the `/etc/sysctl.conf` file. If the `fs.file-max` property is defined in the file, make sure its value is set to at least 65535. If the line does not exist, add the following to the end of the file:

```
fs.file-max = 65535
```

3. Edit the `/etc/security/limits.conf` file. If the file has lines that set the soft and hard limits for the number of file descriptors, make sure the values are set to 65535. If the lines are not present, add the following lines to the end of the file (before `#End of file`). Insert a tab between the columns.

```
* soft nofile 65535
* hard nofile 65535
```

4. Reboot the system, and then use the `ulimit` command to verify that the file descriptor limit is set to 65535 with the following command:

```
ulimit -n
```

Once the operating system limit is set, the number of file descriptors that the server will use can be configured by either using a `NUM_FILE_DESCRIPTOR` environment variable, or by creating a `config/num-file-descriptors` file with a single line such as, `NUM_FILE_DESCRIPTOR=12345`. If these are not set, the default of 65535 is used. This is strictly optional if wanting to ensure that the server shuts down safely prior to reaching the file descriptor limit.

## Set the Filesystem Flushes

Linux systems running the ext3 filesystem only flush data to disk every five seconds. If the server is on a Linux system, edit the mount options to include the following:

```
commit=1
```

This variable changes the flush frequency from five seconds to one. Also, set the flush frequency in the `/etc/fstab` file to make sure the configuration remains after reboot.

## Install sysstat and pstack on Red Hat

The server troubleshooting tool `collect-support-data` relies on the `iostat`, `mpstat`, and `pstack` utilities to collect monitoring, performance statistics, and stack trace information on the server's processes. For Red Hat systems, make sure that these packages are installed, for example:

```
$ sudo yum install sysstat gdb dstat -y
```

## Install the dstat Utility

The `dstat` utility is used by the `collect-support-data` tool.

## Disable Filesystem Swapping

Disable disk swapping on the filesystem to protect the server JVM process from an overly aggressive filesystem cache. Run the following command:

```
# sysctl -w vm.swappiness=0
```

## Managing System Entropy

Entropy is used to calculate random data that is used by the system in cryptographic operations. Some environments with low entropy may have intermittent performance issues with SSL-based communication. This is more typical on virtual machines, but can occur in physical instances as well. Monitor the `kernel.random.entropy_avail` in `sysctl` value for best results.

If necessary, update `$JAVA_HOME/jre/lib/security/java.security` to use `file:/dev/./urandom` for the `securerandom.source` property.

## Configure Identity Servers to be Monitored

Before installing the Metrics Engine, configure the servers to be monitored:

- Data Store
- Proxy Server
- Data Sync Server
- Data Broker

The monitored servers require sufficient disk space to store the monitoring data, and can be configured with Tracked Applications if there are specific application bind DNs that should be monitored.

## Disk Space Requirements and Monitoring Intervals

The metrics backend on the monitored servers is responsible for the temporary storage of metric data, and is configured to keep a maximum amount of metric history based on log retention policies, which are configured with the `dsconfig` tool.

The default retention policies define a cap on disk space usage, which in turn determines the amount of metric history retained. If the Metrics Engine is stopped for a period of time, the monitored servers should be configured to retain enough metrics history to prevent gaps in data when the Metrics Engine restarts. The amount of disk space required for metrics history can also depend on the monitored server's Stats Collector Plugin settings. In general, 500MB is enough to retain an eight-hour span of metrics history.

The value of the `sample-flush-interval` property of the monitored server's metrics backend determines the maximum delay between when a metric is captured and when it can be picked up by the Metrics Engine. The flush interval can be set between 15 and 60 seconds, with longer values resulting in less processing load on the Metrics Engine. However, this flush interval increases the latency between when the metric was captured and when it becomes visible in a chart or dashboard. Changing the `sample-flush-interval` attribute to 60 seconds, has the Metrics Engine keep 2000 minutes of history.

The number of metrics produced per unit of time varies based on the configuration. No formula can be provided to compute exact storage required for each hour of history. In general, 60MB per hour is a standard estimate.

## Tracked Applications

If the Metrics Engine will monitor client applications associated with the monitored servers, the Tracked Applications feature should be configured for monitored servers as well. Activity performed by a particular LDAP Bind DN can be associated with a Metrics Engine application-name, which in turn can be included in Metrics Engine SLA definitions.

The Processing Time Histogram plugin is configured on each Data Store and Proxy server as a set of histogram ranges. These ranges should be defined identically across all monitored servers. For each monitored server, set the `separate-monitor-entry-per-tracked-application` property of the processing time histogram plugin to `true`. Per-application monitoring information will appear under `cn=monitor`. The `per-application-ldap-stats` property must also be set to `per-application-only` in the Stats Collector Plugin. See the *UnboundID Data Store Administration Guide* for Tracked Application configuration details.

The following sets the required property of the Processing Time Histogram plugin:

```
$ bin/dsconfig set-plugin-prop \  
  --plugin-name "Processing Time Histogram" \  
  --set separate-monitor-entry-per-tracked-application:true
```

The following example sets the required property of the Stats Collector plugin:

```
$ bin/dsconfig set-plugin-prop \  
  --plugin-name "Stats Collector" \  
  --set per-application-ldap-stats:per-application-only
```

# Install the Server

Use the `setup` tool to install the server. The server needs to be started and stopped by the user who installed it.

**Note**

A Windows installation requires that the Visual Studio 2010 runtime patch be installed prior to running the `setup` command.

1. Log in as a user, other than root.
2. Obtain the latest zip release bundle from UnboundID Corp and unpack it in a directory owned by this user.

```
$ unzip UnboundID-<Server>-<version>.zip
```

3. Change to the server root directory.

```
$ cd UnboundID-<Server>
```

4. Run the `setup` command.

```
$ ./setup
```

5. Type **yes** to accept the End-User License Agreement and press **Enter** to continue.
6. Read the installation process and prerequisites. Press **Enter** to continue.
7. Type the port number of for the PostgreSQL database instance to use to store monitoring data, or press **Enter** to accept the default port.
8. Enter the directory to be used for PostgreSQL data files, or press **Enter** to accept the default (`pgsql_data`). If the name entered is a relative path name, it will be created in the current working directory.
9. Enter a name for the database administrative account, or press **Enter** to accept the default. The `setup` tool will create a user (role) and database to be used by the Metrics Engine. These credentials are strictly for use by this tool during this session and are not retained.
10. Enter and confirm a password for this account.
11. Specify the name of the PostgreSQL account to be associated with the Metrics Engine historical monitoring data, or press **Enter** to accept the default (`metricsengine`). The tool will create this user account using the administrative account specified in step 9.
12. The password generated for this account is `metricsengine`, press **Enter** to accept the default, or type **yes** and provide and confirm a new password.
13. Enter the fully-qualified host name for the server, or press **Enter** to accept the default.
14. Create the initial root user DN for the server, or press **Enter** to accept (`cn=Directory Manager`)

## Chapter 2: Installing the Metrics Engine

15. Enter and confirm a password for this account.
16. Enter the port for HTTPS connection to the Platform (SCIM and the Configuration) APIs, or press **Enter** to accept the default.
17. Enter the port on which the Metrics Engine will accept LDAP client connections, or press **Enter** to accept the default.
18. To enable LDAPS, type **yes**, or press **Enter** to accept the default **no**.
19. If LDAPS is enabled, enter the port on which the server will accept LDAPS client connections, or press **Enter** to accept the default (2636).
20. To enable StartTLS, type **yes**, or press **Enter** to accept the default **no**.
21. Choose a certificate option for the server.


Certificate server options:

- 1) Generate self-signed certificate (recommended for testing purposes only)
- 2) Use an existing certificate located on a Java KeyStore (JKS)
- 3) Use an existing certificate located on a PKCS12 KeyStore
- 4) Use an existing certificate on a PKCS11 token

Depending on the option chosen, other information may be needed. If the Java or the PKCS#12 KeyStore is chosen, the KeyStore path and PIN is needed. If the PKCS#11 token is chosen, the key PIN is needed.

22. Choose an option to assign the amount of memory that the server should allocate to the Metrics Engine and press **Enter**.
23. Press **Enter** (yes) to start the server when configuration is complete.
24. Press **Enter** to install the Metrics Engine with the defined parameters.

After the Metrics Engine server is installed, access the Metrics landing page (<https://<host>:<HTTPS-port>/view/index>) for access to the default dashboards, chart builder tool, and online documentation.


Metrics Engine

The UnboundID Metrics Engine includes these customizable dashboards for monitoring server operations and for analysis of the activity, performance, and capacity of an UnboundID Platform deployment.

[Documentation](#)
[List of Available Metrics](#)
[REST API Documentation](#)
[Try the Chart Builder](#)

**Server Information**  
velocity/templates/server-info.vm  
A simple template that displays basic server data.

**SLA Viewer**  
velocity/templates/sla-viewer.vm  
Displays the current operation response time and throughput, with drill-down to historical values. Use this template for monitoring adherence to the Service Level Agreements that are defined for critical applications.

**Topology Dashboard**  
velocity/templates/dap-dash-board.vm  
Use this template for monitoring the entire UnboundID deployment. Includes operational metrics at the individual server, data center, and aggregate level by client application and service.

**Broker Dashboard**  
velocity/templates/broker-dash-board.vm

**Consumer Dashboard**  
velocity/templates/consumer.vm

**Enforcement Dashboard**  
velocity/templates/enforcement.vm

Three variations of an example dashboard—showing trends and metrics collected from the Data Broker. These provide details about data use and policy enforcement, such as trends in consumer consent, applications accessing the most data, and the policies used most often—and to allow or deny data requests.

## Server Folders and Files

After the distribution file is unzipped, the following folders and command-line utilities are available:

Directories/Files/Tools	Description
ldif	Stores any LDIF files that you may have created or imported.
import-tmp	Stores temporary imported items.
classes	Stores any external classes for server extensions.
db	For the Data Store, this is where its Berkeley DB files reside.
bak	Stores the physical backup files used with the backup command-line tool.
velocity	Stores Velocity templates that define the server's application pages.
update.bat, and update	The update tool for UNIX/Linux systems and Windows systems.
uninstall.bat, and uninstall	The uninstall tool for UNIX/Linux systems and Windows systems.
unboundid_logo.png	The image file for the UnboundID Corp logo.
setup.bat, and setup	The setup tool for UNIX/Linux systems and Windows systems.
revert-update.bat, and revert-update	The revert-update tool for UNIX/Linux systems and Windows systems.
README	README file that describes the steps to set up and start the server.
License.txt	Licensing agreement for the product.
legal-notice	Stores any legal notices for dependent software used with the product.
docs	Provides the release notes, Configuration Reference (HTML), API Reference, and all other product documentation.



Directories/Files/Tools	Description
metrics	Stores the metrics that can be gathered for this server and surfaced in the Metrics Engine.
bin	Stores UNIX/Linux-based command-line tools.
bat	Stores Windows-based command-line tools.
lib	Stores any scripts, jar files, and library files needed for the server and its extensions.
collector	Used by the server to make monitored statistics available to the Metrics Engine.
locks	Stores any lock files in the backends.
postgres	Stores PostgreSQL files.
tmp	Stores temporary files.
resource	Stores the MIB files for SNMP and can include Idif files, make-Idif templates, schema files, dsconfig batch files, and other items for configuring or managing the server.
config	Stores the configuration files for the backends (admin, config) as well as the directories for messages, schema, tools, and updates.
logs	Stores log files.

## Add Monitored Servers to the Metrics Engine

Configure the Metrics Engine to monitor servers using the `monitored-servers` tool or configure them individually using the `dsconfig` tool.

### Using the monitored-servers Tool

The `monitored-servers` command-line tool configures communication between the servers and the Metrics Engine, then adds external server definitions to the Metrics Engine based on the server's administrative data. Before a server is added to the Metrics Engine configuration, the system determines whether communication needs to be configured. If so, the `cn=Monitoring User` root user account is created on the external server.

Running the tool with the `add-servers` subcommand creates an external server based on the information discovered about the remote server. It also uses the information located in the `cn=admin` data entry to discover other servers in the topology, which are also added to the configuration.

The following examples use the `monitored-servers` tool:

- Run the `monitored-servers` tool with the `add-servers` subcommand. Specify connection information for the Metrics Engine, as well as connection information for any remote servers in use.

```
$ bin/monitored-servers add-servers \
  --bindDN uid=admin,dc=example,dc=com \
  --bindPassword password \
  --monitoringUserBindPassword password \
```

```
--remoteServerHostname localhost \
--remoteServerPort 1389 \
--remoteServerBindPassword password
```

- Use the `--dry-run` option to generate output detailing the work that would be done in a live session without actually making changes to the server configuration.

```
$ bin/monitored-servers add-servers \
--bindDN uid=admin,dc=example,dc=com \
--bindPassword password \
--monitoringUserBindPassword password \
--remoteServerHostname localhost \
--remoteServerPort 1389 \
--remoteServerBindPassword password \
--dry-run
```

## Using the dsconfig Tool

Only servers specified in the `monitored-server` property are actively monitored. Use the `dsconfig` tool to configure individual servers to be monitored by the Metrics Engine with the following steps:

1. Run the `dsconfig` tool.

```
$ bin/dsconfig
```

2. Enter the host, connection type, connection port, and user bind DN account. The Configuration Main Menu displays.
3. Select the **Monitoring Configuration** option.
4. Select the **View and edit the Monitoring Configuration** option.
5. Edit the `monitored-server` property. By default, the Metrics Engine server is the only server listed after installation.
6. Select the **Add one or more values** option to add a new server.
7. Follow the prompts to add the new server.

## Start and Stop the Server

When the Metrics Engine starts for the very first time, it downloads new samples from the monitored servers and adds data to the database. Until it has finished this first data collection, the Metrics Engine will not be able to answer metric queries to the database. The Metrics Engine processes samples from the oldest to the newest, so queries on more recent data may require more start-up time. If the monitored servers have been collecting samples for several days, there may be a significant backlog of data to collect.

To determine if the server is ready to respond to metric queries, run the `status` tool. If the `Sample Import Backlog` property is zero (0), the server is ready.

## Chapter 2: Installing the Metrics Engine

### **Note**

The Metrics Engine needs to be started and stopped by the user who installed it. If the metrics engine is started or stopped by a different user, the following error is listed in the postgres.log file when Postgres starts:

```
FATAL: role "<username>" does not exist
```

## Start the Metrics Engine as a Background Process

Navigate to the server root directory, and run the following command:

```
$ bin/start-metrics-engine
```

For Windows systems:

```
$ bat/start-metrics-engine
```

## Start the Metrics Engine as a Foreground Process

Navigate to the server root directory, and run the following command:

```
$ bin/start-metrics-engine --nodetach
```

## Start the Metrics Engine at Boot Time

By default, the Metrics Engine does not start automatically when the system is booted. To configure the monitoring server to start automatically when the system boots, use the `create-rc-script` tool to create a run control script as follows:

1. Create the startup script as the non-root Metrics Engine user. In this example `ds` is the user.

```
$ bin/create-rc-script --outputFile UnboundID-ME.sh \  
--userName ds
```

2. Log in as root, move the generated `UnboundID-ME.sh` script into the `/etc/init.d` directory, and create symlinks to it from the `/etc/rc3.d` (starting with an "S" to start the server) and `/etc/rc0.d` directory (starting with a "K" to stop the server).

```
# mv UnboundID-ME.sh /etc/init.d/  
# ln -s /etc/init.d/UnboundID-ME.sh /etc/rc3.d/S50-UnboundID-ME.sh  
# ln -s /etc/init.d/UnboundID-ME.sh /etc/rc0.d/K50-UnboundID-ME.sh
```

## Stop the Metrics Engine

Navigate to the server root directory, and run the following command:

```
$ bin/stop-metrics-engine
```

## Restart the Metrics Engine

Restart the Metrics Engine using the `--restart` or `-R` option. Running this command is equivalent to shutting down the server, exiting the JVM session, and then starting up again, which requires a re-priming of the JVM cache.

Navigate to the server root directory, and run the following command:

```
$ bin/stop-metrics-engine --restart
```

## Uninstall the Server

Use the `uninstall` command-line utility to uninstall the server using either interactive or non-interactive modes. Interactive mode provides options, progress, and a list of the files and directories that must be manually deleted if necessary.

Non-interactive mode, invoked with the `--no-prompt` option, suppresses progress information, except for fatal errors. All options for the `uninstall` command are listed with the `--help` option.

The `uninstall` command must be run as either the root user or the user account that installed the server.

Perform the following steps to uninstall in interactive mode:

1. Navigate to the server root directory.

```
$ cd UnboundID-<server>
```

2. Start the uninstall command:

```
$ ./uninstall
```

3. Select the components to be removed, or press **Enter** to remove all components.
4. If the server is running, press **Enter** to shutdown the server before continuing.
5. Manually remove any remaining files or directories, if required.

## Install the Web Console

The Web Console provides configuration and schema management functionality in addition to monitoring and server information. Like the `dsconfig` configuration tool, all changes made using the Web Console are recorded in `logs/config-audit.log`.

The Web Console must be deployed in a servlet container that supports the servlet API 2.5 or later.

**Note** The Web Console supports JBoss 7.1.1 or later.

The following steps use Apache Tomcat as the container.

## Chapter 2: Installing the Metrics Engine

1. Download and install the servlet container.
2. Using Tomcat, set the appropriate environment variables. The `setclasspath.sh` and `catalina.sh` files are in the Tomcat `bin` directory.

```
$ echo "BASEDIR=/path/to/tomcat" >> setclasspath.sh
$ echo "CATALINA_HOME=/path/to/tomcat" >> catalina.sh
```

3. Download the console ZIP file, `<server>-web-console-<version>-GAimage.zip` and unzip the file. The following files are listed:

```
3RD-PARTY-LICENSE.TXT
LICENSE.TXT
README
<server>console.war
```

4. Create a `<server>console` directory in the `apache-tomcat-<version>/webapps` directory and copy the `<server>console.war` file to that directory. If the servlet is running and auto-deploy is enabled, the `.war` file will install in the directory.

```
$ mkdir apache-tomcat-<version>/webapps/<server>console
$ cp <server>console.war apache-tomcat-<version>/webapps/<server>console
```

5. Navigate to the `apache-tomcat-<version>/webapps/<server>console` directory to extract the contents of the console. The `jar` command is included with the JDK.

```
$ cd apache-tomcat-<version>/webapps/<server>console
$ jar xvf <server>console.war
```

6. The `WEB-INF/web.xml` file can be edited to point to the correct server instance. Uncomment the necessary parameters and change the host and port to match the server. For example:

```
<context-param>
  <param-name>ldap-servers</param-name>
  <param-value>localhost:389</param-value>
</context-param>
```

If the `ldap-servers` parameter is not defined, a user logging into the Web Console must enter the server host and port.

7. The session timeout value can also be changed by editing `apache-tomcat-<version>/conf/web.xml` and adding a new value in minutes:

```
<session-config>
  <session-timeout>120</session-timeout>
</session-config>
```

8. Optional. To remove sensitive information from console error messages, such as LDAP and server information, edit `apache-tomcat-<version>/conf/web.xml`, and change the value of this configuration parameter to `false`:

```
<context-param>
  <param-name>detailedErrorMessage</param-name>
```

```
<param-value>>false</param-value>
</context-param>
```

When set to `false`, the console will display a generic error page with server information removed. Server logs will still contain detailed error information.

9. Start the server if it is not already running, and then start the console using the `apache-tomcat-<version>/bin/startup.sh` script. (On Microsoft Windows, use `startup.bat`.)
10. Set the `JAVA_HOME` environment variable to specify the location of the Java installation to run the server.

```
$ env JAVA_HOME=/ds/java bin/startup.sh
Using CATALINA_BASE: /apache-tomcat-<version>
Using CATALINA_HOME: /apache-tomcat-<version>
Using CATALINA_TMPDIR: /apache-tomcat-<version>/temp
Using JRE_HOME: /ds/java
```

11. Open a browser to `http://hostname:8080/<server>console`. By default, Tomcat listens on port 8080 for HTTP requests.

## Configure the Web Console

The Web Console uses a `web.xml` descriptor file for its configuration and deployment settings. Configuration in this file includes defining one or more primary servers and configuring security and truststore settings. If servers are defined in this file, the console will automatically attempt to connect to the server(s) in the order in which they are specified until one of the servers can authenticate the username and password given on the login page. The console also uses this server to "discover" other servers in the topology, making them available for monitoring and management in the console.

Perform the following steps to add servers:

1. Open the `<server>console/WEB-INF/web.xml` file in a text editor to specify the server(s) that the console uses to authenticate.
2. Remove the comment tags (`<!-- -->`) in the `ldap-servers` section.
3. Specify the servers as `host:port` ( `server1.example.com:389`) or using the LDAPS protocol to specify security information (`ldaps://server1.example.com:389`). Separate multiple servers with a space. For example, a server using standard LDAP communication and another other using SSL would be listed as the following:

```
<context-param>
  <param-name>ldap-servers</param-name>
  <param-value>localhost:389 ldaps://svr1.example.com:389</param-value>
</context-param>
```

3. Save the file.

## Configure SSL or StartTLS for the Console

Configure the console to communicate with all of its servers over SSL or StartTLS.

1. Open the `<server>console/WEB-INF/web.xml` file in a text editor.
2. Remove the comment tags (`<!--` and `-->`) in the security section.
3. Specify `none`, `ssl`, or `starttls` for the type of security used to communicate with the server.

```
<context-param>
  <param-name>security</param-name>
  <param-value>ssl</param-value>
</context-param>
```

4. Save the file.

## Configure a Truststore for the Console

For SSL and StartTLS communication, specify the truststore and its password (or password file) in the `web.xml` file. If no truststore is specified, all server certificates are trusted.

1. Open the `<server>console/WEB-INF/web.xml` file in a text editor.
2. Remove the comment tags (`<!--` and `-->`) in the truststore section.
3. Specify the path to the truststore.

```
<context-param>
  <param-name>trustStore</param-name>
  <param-value>/path/to/truststore</param-value>
</context-param>
```

4. Specify the password or the path to the password pin file.

```
<context-param>
  <param-name>trustStorePassword</param-name>
  <param-value>password</param-value>
</context-param>

<context-param>
  <param-name>trustStorePasswordFile</param-name>
  <param-value>/path/to/truststore/pin/file</param-value>
</context-param>
```

4. Save the file.

## Log Into the Console

To log into the Console, use a DN (for example, `cn=Directory Manager`) or provide the name of an administrator stored under `cn=admin` data. The `dsframework` command can be used to create a global administrator, for example:

```
$ dsframework create-admin-user \
--hostname server1.example.com \
--port 1389 --bindDN "cn=Directory Manager" \
--bindPassword secret \
--userID someAdmin \
--set password:secret
```

Perform the following steps to log into the Web Console:

1. Navigate to the server root directory.

```
$ cd UnboundID-<Server>
```

2. Start the server.

```
$ start-<server>
```

3. Start the Apache Tomcat application server.

```
$ /apache-tomcat-<version>/bin/startup.sh
```

4. Open a browser to `http://hostname:8080/<server>console/`.
5. Type the root user DN (or any authorized administrator user name) and password, and click **Login**.
6. On the Web Console, click **Configuration**.
7. View the **Configuration** menu. By default, the Basic object type properties are displayed. The complexity level of the object types is changed using the **Object Types** drop-down list.

## Upgrade the Web Console

Perform the following steps to upgrade the Web Console:

1. Shut down the console and servlet container.
2. In the current deployment of the console, move the `webapps/<server>console/WEB-INF/web.xml` file to another location.
3. Download and deploy the latest version of the console. See [Install the Web Console](#).
4. Apply any configuration changes from the previous version of the `web.xml` file to the new file.
5. Start the servlet container.

## Uninstall the Console

Perform the following steps to uninstall the Web Console:



## Chapter 2: Installing the Metrics Engine

1. Close the console.
2. Shut down the servlet container with the following command (On Microsoft Windows, use `shutdown.bat`):

```
$ apache-tomcat-<version>/bin/shutdown.sh
```

3. Remove the `webapps/<server>console` directory with the following command:

```
$ rm -rf webapps/<server>console
```

4. Restart the servlet container instance if running other applications.

---

## Chapter 3: Managing the Metrics Engine

---

There are several ways to manage the Metrics Engine status and performance.

Topics include:

[Metrics Engine Error Logging](#)

[Backend Monitor Entries](#)

[Configure Alert Handlers](#)

[The Alerts Backend](#)

[System Alarms and Gauges](#)

[Backup the Metrics Engine Database](#)

[Management Tools](#)

[Using the Configuration API](#)

[Domain Name Service \(DNS\) Caching](#)

[IP Address Reverse Name Lookup](#)

[Server SDK Extensions](#)

## Metrics Engine Error Logging

The Metrics Engine provides logging for warnings, errors, or significant events that occur within the server. Log publishers rely on log rotation and retention policies. Customization options for log publishers are available with the `dsconfig` command, (`bin/dsconfig` or `bat/dsconfig` on Windows).

Each log publisher must have at least one log rotation policy and log retention policy configured. Configure the log rotation policy for each log publisher. When a rotation limit is reached, the server rotates the current log and starts a new log.

### Logging Retention Policies

Select retention configuration from the following:

**Time Limit Rotation Policy** – Rotates the log based on the length of time since the last rotation. Default implementations are provided for rotation every 24 hours and every seven days.

**Fixed Time Rotation Policy** – Rotates the logs every day at a specified time (based on 24-hour time). The default time is 2359.

**Size Limit Rotation Policy** – Rotates the logs when the file reaches the maximum size for each log. The default size limit is 100MB.

**Never Rotate Policy** – Used in a rare event that does not require log rotation.

### Logging Rotation Policies

Select rotation configuration from the following:

**File Count Retention Policy** – Sets the number of log files for the Metrics Engine to retain. The default file count is 10 logs. If the file count is set to 1, then the log will continue to grow indefinitely without being rotated.

**Free Disk Space Retention Policy** – Sets the minimum amount of free disk space. The default free disk space is 500MB.

**Size Limit Retention Policy** – Sets the maximum size of the combined archived logs. The default size limit is 500MB.

**Custom Retention Policy** – Create a new retention policy. This requires developing custom code to implement the log retention policy.

**Never Delete Retention Policy** – Used in a rare event that does not require log deletion.

### Create Log Publishers

Create a new log publisher with `dsconfig`, either from the command line or in interactive mode. Retention and rotation policies must be configured for the log publisher. For more information about policy options, see [Configure Logs](#).

**Note**

Compression cannot be disabled or turned off once configured for the logger. Determine logging requirements, prior to creating and configuring them.

Perform the following steps to create a log publisher:

1. The following creates a log publisher with the `dsconfig` command that logs disconnect operations.

```
$ bin/dsconfig create-log-publisher \
  --type file-based-access --publisher-name "Disconnect Logger" \
  --set enabled:true \
  --set "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --set "rotation-policy:Size Limit Rotation Policy" \
  --set "retention-policy:File Count Retention Policy" \
  --set log-connects:false \
  --set log-requests:false --set log-results:false \
  --set log-file:logs/disconnect.log
```

To configure compression on the logger, add this option to the previous command:

```
--set compression-mechanism: gzip
```

2. To view log publishers, enter the following command:

```
$ bin/dsconfig list-log-publishers
```

## Error Log Publisher

The Error Log reports errors, warnings, and informational messages about events that occur during the course of the server's operation. Each entry in the error log records the following properties (some are disabled by default and must be enabled):

**Time Stamp** – Displays the date and time of the operation in the format

DD/Month/YYYY:HH:MM:SS <offset from UTC time>.

**Category**– Specifies the message category that is loosely based on the server components.

**Severity** – Specifies the message severity of the event, which defines the importance of the message in terms of major errors that need to be quickly addressed. The default severity levels are `fatal-error`, `notice`, `severe-error`, and `severe-warning`.

**Message ID** – Specifies the numeric identifier of the message.

**Message** – Stores the error, warning, or informational message.

The following example displays an error log for the Metrics Engine. The log is enabled by default and is accessible in the `<server-root>/logs/errors` file.

```
[21/Oct/2012:05:15:23.048 -0500] category=RUNTIME_INFORMATION severity=NOTICE
msgID=20381715 msg="JVM Arguments: '-Xmx8g', '-Xms8g', '-XX:MaxNewSize=1g',
'-XX:NewSize=1g', '-XX:+UseConcMarkSweepGC', '-XX:+CMSConcurrentMTEnabled',
'-XX:+CMSParallelRemarkEnabled', '-XX:+CMSParallelSurvivorRemarkEnabled',
'-XX:+CMSScavengeBeforeRemark', '-XX:RefDiscoveryPolicy=1',
'-XX:ParallelCMSThreads=4', '-XX:CMSMaxAbortablePrecleanTime=3600000',
'-XX:CMSInitiatingOccupancyFraction=80', '-XX:+UseParNewGC', '-XX:+UseMembar',
```

## Chapter 3: Managing the Metrics Engine

```
'-XX:+UseBiasedLocking', '-XX:+UseLargePages', '-XX:+UseCompressedOops',
'-XX:PermSize=128M', '-XX:+HeapDumpOnOutOfMemoryError',
'-Dcom.unboundid.directory.server.scriptName=setup'"
[21/Oct/2012:05:15:23.081 -0500] category=EXTENSIONS severity=NOTICE
msgID=1880555611 msg="Administrative alert type=server-starting
id=4178daee-ba3a-4be5-8e07-5ba17bf30b71
class=com.unboundid.directory.server.core.MetricsEngine
msg='The Metrics Engine is starting'"
[21/Oct/2012:05:15:23.585 -0500] category=CORE severity=NOTICE
msgID=1879507338 msg="Starting group processing for backend api-users"
[21/Oct/2012:05:15:23.586 -0500] category=CORE severity=NOTICE
msgID=1879507339 msg="Completed group processing for backend api-users"
[21/Oct/2012:05:15:23.586 -0500] category=EXTENSIONS severity=NOTICE
msgID=1880555575 msg="'Group cache (2 static group(s) with 0 total
memberships and 0 unique members, 0 virtual static group(s),
1 dynamic group(s))' currently consumes 7968 bytes and can grow to a maximum
of an unknown number of bytes"
[21/Oct/2012:05:16:18.011 -0500] category=CORE severity=NOTICE
msgID=458887 msg="The Metrics Engine (UnboundID Metrics Engine 4.5.1.0
build 20121021003738Z, R12799) has started successfully"
```

Use `dsconfig` to modify the default File-Based Error Log, as in the following command:

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Error Logger" \
  --set include-product-name:true --set include-instance-name:true \
  --set include-startup-id:true
```

## Backend Monitor Entries

Each UnboundID Corp server exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers through SNMP, the Web Console, JConsole, LDAP command-line tools, and the Stats Logger.

The Monitor Backend contains an entry per component or activity being monitored. The list of all monitor entries can be seen using the `ldapsearch` command as follows:

```
$ bin/ldapsearch --hostname server1.example.com \
  --port 1389 \
  --bindDN "uid=admin,dc=example,dc=com" \
  --bindPassword secret \
  --baseDN "cn=monitor" "(objectclass=*)" cn
```

The following table lists a subset of monitor entries.

Monitoring Components	
Component	Description
Active Operations	Provides information about the operations currently being processed by the server including the number of operations, information on each operation, and the number of active persistent searches.

## Monitoring Components

Component	Description
Backends	Provides general information about the state of a server backend, including the entry count. If the backend is a local database, there is a corresponding database environment monitor entry with information on cache usage and on-disk size.
Client Connections	Provides information about all client connections to the server including a name followed by an equal sign and a quoted value, such as <code>connID="15"</code> , <code>connectTime="20100308223038Z"</code> .
Connection Handlers	Provides information about the available connection handlers on the server including the LDAP and LDIF connection handlers.
Disk Space Usage	Provides information about the disk space available to various components of the server.
General	Provides general information about the state of the server, including product name, vendor name, and server version.
Index	Provides information on each index including the number of preloaded keys and counters for read, write, remove, open-cursor, and read-for-search actions. These counters provide insight into how useful an index is for a given workload.
HTTP/HTTPS Connection Handler Statistics	Provides statistics about the interaction that the associated HTTP connection handler has had with its clients, including the number of connections accepted, average requests per connection, average connection duration, total bytes returned, and average processing time by status code.
JVM Stack Trace	Provides a stack trace of all threads processing within the JVM.
LDAP Connection Handler Statistics	Provides statistics about the interaction that the associated LDAP connection handler has had with its clients, including the number of connections established and closed, bytes read and written, LDAP messages read and written, and operations initiated, completed, and abandoned.
Processing Time Histogram	Categorizes operation processing times into a number of user-defined buckets of information, including the total number of operations processed, overall average response time (ms), and number of processing times between 0ms and 1ms.
System Information	Provides general information about the system and the JVM on which the server is running, including system host name, operation system, JVM architecture, Java home, and Java version.
Version	Provides information about the server version, including build ID, and revision number.
Work Queue	<p>Provides information about the state of the server work queue, which holds requests until they can be processed by a worker thread, including the requests rejected, current work queue size, number of worker threads, and number of busy worker threads.</p> <p>The work queue configuration has a <code>monitor-queue-time</code> property set to <code>true</code> by default. This logs messages for new operations with a <code>qtime</code> attribute included in the log messages. Its value is expressed in milliseconds and represents the length of time that operations are held in the work queue.</p>

## Disk Space Usage Monitor

The disk space usage monitor provides information about the amount of usable disk space available for server components. It also provides the ability to generate administrative alerts, as well as take action if the amount of usable space drops below the defined thresholds.

The disk space usage monitor evaluates the free space at locations registered through the `DiskSpaceConsumer` interface. Disk space monitoring excludes disk locations that do not have server components registered. However, other disk locations may still impact server performance, such as the operating system disk, if it becomes full. When relevant to the server, these locations include the server root, the location of the `config` directory, the location of every log file, all JE backend directories, the location of the changelog, the location of the replication environment database, and the location of any server extension that registers itself with the `DiskSpaceConsumer` interface.

All values must be specified as absolute values or as percentages. A mix of absolute values and percentages cannot be used. The following thresholds are available:

- **Low space warning** – This threshold defines either a percentage or an absolute amount of usable space. If the amount of usable space drops below this threshold, the server generates an administrative alert. It generates alerts at regular intervals, based on configuration settings, until the amount of usable space is increased, or as the amount of usable space is further reduced.
- **Low space error** – This threshold is also defined as either a percentage or an absolute size. Once the amount of usable space drops below this threshold, the server will generate an alert notification and will begin rejecting all operations requested by non-root users with "UNAVAILABLE" results. Once the server enters this mode, some action must be taken before the server will resume normal operations. This threshold must be less than or equal to the low space warning threshold. If they are equal, the server will begin rejecting requests from non-root users immediately upon detecting low usable disk space.
- **Out of space error** – This threshold can also be defined as a percentage or an absolute size. Once the amount of usable space drops below this threshold, the server will generate a final administrative alert and will shut itself down. This threshold must be less than or equal to the low space error threshold. If they are equal, the server will shut itself down rather than rejecting requests from non-root users.

## Notifications and Alerts

Each UnboundID sever provides delivery mechanisms for account status notifications, administrative alerts, and alarms using SMTP, JMX, or SNMP. Alerts, alarms, and events reflect state changes within the server that may be of interest to a user or monitoring service. Account status notifications are only delivered to the account owner.

Alert handler implementations include:

**Error Log Alert Handler** – Sends administrative alerts to the configured server error logger (s).

**Exec Alert Handler** – Executes a specified command on the local system if an administrative alert matching the criteria for this alert handler is generated by the server. Information about the administrative alert is made available to the executed application as arguments provided by the command.

**Groovy Scripted Alert Handler** – Provides alert handler implementations defined in a dynamically-loaded Groovy script that implements the `ScriptedAlertHandler` class defined in the Server SDK.

**JMX Alert Handler** – Sends administrative alerts to clients using the Java Management Extensions (JMX) protocol. UnboundID uses JMX for monitoring entries and requires that the JMX connection handler be enabled.

**SMTP Alert Handler** – Sends administrative alerts to clients via email using the SMTP. The server requires that one or more SMTP servers be defined in the global configuration.

**SNMP Alert Handler** – Sends administrative alerts to clients using the Simple Network Monitoring Protocol (SNMP). The server must have an SNMP agent capable of communicating through SNMP.

**SNMP Subagent Alert Handler** – Sends SNMP traps to a master agent in response to administrative alerts generated within the server.

**Third Party Alert Handler** – Provides alert handler implementations created in third-party code using the Server SDK.

A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`

## Configure Alert Handlers

Alert handlers can be configured with the `dsconfig` tool. UnboundID servers support JMX, SMTP, and SNMP. Use the `--help` option for a list of configuration options. The following is a sample command to create and enable an SMTP Alert handler from the command line:

```
$ bin/dsconfig create-alert-handler \
  --handler-name "SMTP Alert Handler" \
  --type smtp \
  --set enabled:true \
  --set "sender-address:alerts@example.com" \
  --set "recipient-address:administrators@example.com" \
  --set "message-subject:Directory Admin Alert \%%alert-type\%%" \
  --set "message-body:Administrative alert:\n\%%alert-message\%%"
```

## The Alerts Backend

UnboundID servers generate administrative alerts under the `cn=alerts` branch. The backend makes it possible to obtain admin alert information over LDAP for use with remote monitoring.



## Chapter 3: Managing the Metrics Engine

The backend's primary job is to process search operations for alerts. It does not support add, modify, or modify DN operations of entries.

The alerts persist on disk in the `config/alerts.ldif` file so that they can survive server restarts. By default, the alerts remain on disk for seven days before being removed. However, administrators can configure the number of days for alert retention using the `dsconfig` tool. The administrative alerts of Warning level or worse that have occurred in the last 48 hours are viewable from the output of the `status` command-line tool and in the Web Console.

### View Information in the Alerts Backend

Use `ldapsearch` to view the administrative alerts:

```
$ bin/ldapsearch --port 1389 --bindDN "cn=Directory Manager" \
  --bindPassword secret --baseDN cn=alerts "(objectclass=*)"
dn: cn=alerts
objectClass: top
objectClass: ds-alert-root
cn: alerts

dn: ds-alert-id=3d1857a2-e8cf-4e80-ac0e-ba933be59eca,cn=alerts
objectClass: top
objectClass: ds-admin-alert
ds-alert-id: 3d1857a2-e8cf-4e80-ac0e-ba933be59eca
ds-alert-type: server-started
ds-alert-severity: info
ds-alert-type-oid: 1.3.6.1.4.1.32473.2.11.33
ds-alert-time: 20110126041442.622Z
ds-alert-generator: com.unboundid.directory.server.core.metrics.engine
ds-alert-message: The server has started successfully
```

### Modify the Alert Retention Time

Use `dsconfig` to change the maximum time information about generated alerts retained in the alerts backend. After this time, the information is purged from the server. The minimum retention time is 0 milliseconds, which immediately purges the alert information.

```
$ bin/dsconfig set-backend-prop --backend-name "alerts" \
  --set "alert-retention-time: 2 weeks"
```

View the property using `dsconfig`:

```
$ bin/dsconfig get-backend-prop --backend-name "alerts" \
  --property alert-retention-time
```

```
Property : Value(s)
-----:-----
alert-retention-time : 2 w
```

## Configure Duplicate Alert Suppression

Use `dsconfig` to configure the maximum number of times an alert is generated within a particular time frame for the same condition. The `duplicate-alert-time-limit` property specifies the length of time that must pass before duplicate messages are sent over the administrative alert framework and the maximum number of messages should be sent.

```
$ bin/dsconfig set-global-configuration-prop \  
--set duplicate-alert-limit:2 \  
--set "duplicate-alert-time-limit:3 minutes"
```

## System Alarms, Alerts, and Gauges

An alarm represents a stateful condition of the server or a resource that may indicate a problem, such as low disk space or external server unavailability. A gauge defines a set of threshold values with a specified severity that, when crossed, cause the server to enter or exit an alarm state. Gauges are used for monitoring continuous values like CPU load or free disk space (Numeric Gauge), or an enumerated set of values such as 'server available' or 'server unavailable' (Indicator Gauge). Gauges generate alarms, when the gauge's severity changes due to changes in the monitored value. Like alerts, alarms have severity (NORMAL, WARNING, MINOR, MAJOR, CRITICAL), name, and message. Alarms will always have a `Condition` property, and may have a `Specific Problem or Resource` property. If surfaced through SNMP, a `Probable Cause` property and `Alarm Type` property are also listed. Alarms can be configured to generate alerts when the alarm's severity changes.

There are two alert types supported by the server - standard and alarm-specific. The server constantly monitors for conditions that may need attention by administrators, such as low disk space. For this condition, the standard alert is `low-disk-space-warning`, and the alarm-specific alert is `alarm-warning`. The server can be configured to generate alarm-specific alerts instead of, or in addition to, standard alerts. By default, standard alerts are generated for conditions internally monitored by the server. However, gauges can only generate alarm-alerts.

The server installs a set of gauges that are specific to the product and that can be cloned or configured through the `dsconfig` tool. Existing gauges can be tailored to fit each environment by adjusting the update interval and threshold values. Configuration of system gauges determines the criteria by which alarms are triggered. The Stats Logger can be used to view historical information about the value and severity of all system gauges.

The UnboundID servers are compliant with the International Telecommunication Union CCITT Recommendation X.733 (1992) standard for generating and clearing alarms. If configured, entering or exiting an alarm state can result in one or more alerts. An alarm state is exited when the condition no longer applies. An `alarm_cleared` alert type is generated by the system when an alarm's severity changes from a non-normal severity to any other severity. An `alarm_cleared` alert will correlate to a previous alarm when `Condition` and `Resource` property are the same. The Alarm Manager, which governs the actions performed when an alarm state is entered, is configurable through the `dsconfig` tool and Web Console.

Like the Alerts Backend, which stores information in `cn=alerts`, the Alarm Backend stores information within the `cn=alarms` backend. Unlike alerts, alarm thresholds have a state over time that can change in severity and be cleared when a monitored value returns to normal. Alarms can be viewed with the `status` tool. As with other alert types, alert handlers can be configured to manage the alerts generated by alarms. A complete listing of system alerts, alarms, and their severity is available in `<server-root>/docs/admin-alerts-list.csv`.

## Testing Alerts and Alarms

After alarms and alert handlers are configured, verify that the server takes the appropriate action when an alarm state changes by manually increasing the severity of a gauge. Alarms and alerts can be verified with the `status` tool.

### To Test Alarms and Alerts

1. Configure a gauge with `dsconfig` and set the `override-severity` property to critical. The following example uses the CPU Usage (Percent) gauge.

```
$ dsconfig set-gauge-prop \
  --gauge-name "CPU Usage (Percent)" \
  --set override-severity:critical
```

2. Run the `status` tool to verify that an alarm was generated with corresponding alerts. The `status` tool provides a summary of the server's current state with key metrics and a list of recent alerts and alarms. The sample output has been shortened to show just the alarms and alerts information.

```
$ bin/status
```

```

--- Administrative Alerts ---
Severity : Time           : Message
-----:-----:-----
Error    : 11/Aug/2014    : Alarm [CPU Usage (Percent). Gauge CPU Usage
(Percent)
          : 15:41:00 -0500      : for Host System has
          :                      : a current value of '18.58333333333332'.
          :                      : The severity is currently OVERRIDDEN in the
          :                      : Gauge's configuration to 'CRITICAL'.
          :                      : The actual severity is: The severity is
          :                      : currently 'NORMAL', having assumed this
severity
          :                      : Mon Aug 11 15:41:00 CDT 2014. If CPU use is
high,
          :                      : check the server's current workload and make
any
          :                      : needed adjustments. Reducing the load on the
system
          :                      : will lead to better response times.
          :                      : Resource='Host System']
```

```

:
: raised with critical severity
Shown are alerts of severity [Info,Warning,Error,Fatal] from the past 48
hours
Use the --maxAlerts and/or --alertSeverity options to filter this list

```

```

--- Alarms ---
Severity : Severity : Condition : Resource : Details
: Start Time : : :
-----:-----:-----:-----:-----
--
Critical : 11/Aug/2014: CPU Usage : Host System : Gauge CPU Usage
(Percent) for
: 15:41:00 : (Percent) : : Host System
: -0500 : : : : has a current value of
: : : : : '18.785714285714285'.
: : : : : The severity is
currently
: : : : : 'CRITICAL', having
assumed
: : : : : this severity Mon Aug 11
: : : : : 15:49:00 CDT 2014. If
CPU use
: : : : : is high, check the
server's
: : : : : current workload and
make any
: : : : : needed adjustments.
Reducing
: : : : : the load on the system
will
: : : : : lead to better response
times
Shown are alarms of severity [Warning,Minor,Major,Critical
Use the --alarmSeverity option to filter this list

```

## Back Up the Metrics Engine Database

The Metrics Engine stores all historical metric samples in the PostgreSQL DBMS, along with several other data tables that are used for bookkeeping and normalization of the sample data. Even a small Metrics Engine installation, which monitors three to four servers, will use sample tables that occupy 95% of the total DBMS space. While a functional backup must capture a consistent view of several tables, the size of the sample tables dictates the desired approach to a regular backup strategy.

The historical samples enable:

- Diagnosing past performance problems.
- Capacity planning and historical reporting.

- Access to data needed for a revenue stream, such as data used for billing and charge back.

Defining data that is important to the infrastructure will help determine the right backup strategy. In the case of billing, the data needed is typically small compared to the total population of the DBMS. This may be all the data needed, and the planning and resources required to backup the DBMS will be minimal.

If it's not possible to determine what data will be important in the future, backing up all DBMS data is the safest approach.

### Historical Data Storage

The Metrics Engine DBMS stores all historical sample data. It can store time-aggregated data for up to twenty years. The data in the DBMS is continually changing as long as the Metrics Engine is running.

The system that feeds data to the Metrics Engine is designed to allow the Metrics Engine to be offline for hours at a time without dropping any data. The collection points hold the data for hours, giving the Metrics Engine time for maintenance tasks. The collection points do have a limit on how long they hold data, so the Metrics Engine cannot be offline for an indeterminate time.

If the Metrics Engine is offline so long that the collection points start to delete data that has not yet been captured, then there will be gaps in the data. Aggregation still works, even with these gaps. If the data gap is four hours, four time samples will be missing in the one hour aggregation level, and no data will be missing in the one day aggregation level. However, the one day aggregation level will use only 20 hours of data rather than 24. By default, the Metrics Engine can be offline for about eight hours before any data is lost.

The Metrics Engine responds to queries that result in data with time gaps. The resulting data differentiates between data with zero value and missing data.

### Planning the Backup

Choose a time window during which the Metrics Engine can be offline and ensure that there is enough disk space to hold the new image. The exact size of a DBMS table and its corresponding backup depends on the number of monitored servers, the number of tracked applications, the collected metrics, and the retention duration for each of the aggregation levels. The following table provides values from installations used during testing.

Data From Sample Deployments		
Data	25 Monitored Servers	50 Monitored Servers
Number of tracked applications	20	20
1 second data resolution	8 hours	8 hours
1 minute data retention	14 days	14 days
1 hour data retention	52 weeks	52 week
1 day data retention	20 years	20 years

Data From Sample Deployments

Data	25 Monitored Servers	50 Monitored Servers
1 second table size	22 G	42 G
1 minute table size	8 G	18 G
1 hour table size	4 G (estimated)	9 G (estimated)
1 day data retention	4 G (estimated)	7 G (estimated)
time to backup	15 minutes (estimated)	30 minutes (estimated)
time for import catchup	10 minutes	42 minutes
size of compressed backup image	3 G (estimated)	5.5 G (estimated)
time to restore	1 hour (estimated)	2 h (estimated)

If no backups are performed and the DBMS is completely lost, reinitialize the DBMS, restart the Metrics Engine, and start collecting data again. All collected metric and event data are lost, but the configuration required to start collecting data again is retained.

## Start the DBMS Backup

Shut down the Metrics Engine before a backup or restore.

To backup the entire DBMS use the following command:

```
$ tar -cf backup.tar <path-to-postgres-data-directory>
```

## Restore a DBMS Backup

To restore the full backup to a new database, use the following command:

```
$ tar -xvf backup.tar
```

Run the command from the base directory of the PostgreSQL data directory.

For more information, documentation is available on the PostgreSQL website.

# Management Tools

The Metrics Engine provides several command-line tools to administer the server. The command-line tools are available in the `bin` directory for UNIX or Linux systems and `bat` directory for Microsoft Windows systems.

Each command-line utility provides a description of the subcommands, arguments, and usage examples needed to run the tool. View detailed argument options and examples by typing `--help` with the command.

```
$ bin/dsconfig --help
```

To list the subcommands for each command:

```
$ bin/dsconfig --help-subcommands
```

To list more detailed subcommand information:

```
$ bin/dsconfig list-log-publishers --help
```

## Available Command-Line Utilities

The following command-line utilities are available, which can be run in interactive, non-interactive, or script mode.

Command Line Tools	
Command-Line Tool	Description
backup	Run full or incremental backups on one or more Metrics Engine backends. This utility also supports the use of a properties file to pass predefined command-line arguments. See <a href="#">Managing the tools.properties File</a> for more information.
base64	Encode raw data using the base64 algorithm or decode base64-encoded data back to its raw representation.
collect-support-data	Collect and package system information useful in troubleshooting problems. The information is packaged as a ZIP archive that can be sent to a technical support representative.
create-rc-script	Create an Run Control (RC) script that may be used to start, stop, and restart the server on UNIX-based systems.
config-diff	Generate a summary of the configuration changes in a local or remote server instance. The tool can be used to compare configuration settings when troubleshooting issues, or when verifying configuration settings on new servers.
dsconfig	View and edit the server configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the server and associated tools. Before launching the command, edit the properties file located in <code>config/java.properties</code> to specify the desired JVM options and <code>JAVA_HOME</code> environment variable.
ldapmodify	Perform LDAP modify, add, delete, and modify DN operations.
ldappasswordmodify	Perform LDAP password modify operations.
ldapsearch	Perform LDAP search operations.
ldif-diff	Compare the contents of two LDIF files, the output being an LDIF file needed to bring the source file in sync with the target.
ldifmodify	Apply a set of modify, add, and delete operations against data in an LDIF file.
manage-extension	Install or update extension bundles. An extension bundle is a package of extension (s) that utilize the Server SDK to extend the functionality of the server. Extension bundles are installed from a zip archive or file system directory. The server is restarted to activate the extension(s).
metric-engine-schema	Show current and required DBMS schema version information.
monitored-servers	Configure the set of servers to be monitored by this Metrics Engine and prepare external servers for monitoring.
query-metric	Explore collected monitoring data by forming queries for charts and data.
queryrate	Execute metric queries.
restore	Restore a backup of the server backend.

### Command Line Tools

Command-Line Tool	Description
revert-update	Returns a server to the version before the last update was performed.
review-license	Review or accept the product license.
server-state	View information about the current state of the server process.
setup	Perform the initial setup for the server instance.
start-metrics-engine	Start the server.
status	Display basic server information.
stop-metrics-engine	Stop or restart the server.
sum-file-sizes	Calculate the sum of the sizes for a set of files.
uninstall	Uninstall the server.
update	Update the server to a newer version by downloading and unzipping the new server install package on the same host as the server to update. Use the update tool from the new server package to update the older version of the server. During the update process, the server is stopped if running, then the update is performed. A check is performed to determine if the newly updated server starts without major errors. If it cannot start cleanly, the update is backed out and the server is returned to its prior state. See the <code>revert-update</code> tool for information on reverting an update.

## The tools.properties File

The `tools.properties` file simplifies command-line invocations by reading in a set of arguments for each tool from a text file. Each property consists of a name/value pair for a tool's arguments.

Two types of properties files are supported:

- Default properties files that can be applied to all command-line utilities.
- Tool-specific properties file that can be specified using the `--propertiesFilePath` option.

All of the server's command-line utilities can be over-written using the `config/tools.properties` file.

Create a properties file with a text editor or using the standard Java properties file format (name=value). For example, create a simple properties file that defines a set of LDAP connection parameters as follows:

```
hostname=server1.example.com
port=1389
bindDN=cn=Directory\ Manager
bindPassword=secret
```

Specify the location of the file using the `--propertiesFilePath` option. For example, specify the path to the properties file with `ldapsearch` as follows:

```
$ bin/ldapsearch --propertiesFilePath bin/mytools.properties "(objectclass=*)"
```



Properties files do not allow quotation marks around values. Any spaces or special characters should be escaped.

### Tool-Specific Properties

The server also supports properties for specific tool options using the format: `tool.option=value`. Tool-specific options have precedence over general options. For example, the following properties file uses `ldapsearch.port=2389` for `ldapsearch` requests by the client.

All other tools that use the properties file use `port=1389`.

```
hostname=server1.example.com
port=1389
ldapsearch.port=2389
bindDN=cn=Directory\ Manager
```

Another example using the `dsconfig` configuration tool is as follows:

```
hostname=server1.example.com
port=1389
bindDN=cn=Directory\ Manager
dsconfig.bindPasswordField=/ds/config/password
```

### Specify Default Properties Files

The server provides a default properties file, `tools.properties`, that applies to all command-line utilities used in client requests. The file is located in the `<server-root>/config` directory.

To use a file with a different filename in this default location, specify the path using the `--propertiesFilePath` option.

### Evaluation Order

The following evaluation ordering is used to determine options for a given command-line utility:

- All options used with a utility on the command line take precedence over any options in any properties file.
- If the `--propertiesFilePath` option is used with no other options, the server takes its options from the specified properties file.
- If no options are used on the command line including the `--propertiesFilePath` option (and `--noPropertiesFile`), the server searches for the `tools.properties` file at `<server-root>`.
- If no default properties file is found and a required option is missing, the tool generates an error.
- Tool-specific properties (for example, `ldapsearch.port=3389`) have precedence over general properties (for example, `port=1389`).

## Using the Configuration API

UnboundID servers provide a Configuration API, which may be useful in situations where using LDAP to update the server configuration is not possible. The API is consistent with the System for Cross-domain Identity Management (SCIM) 2.0 protocol and uses JSON as a text exchange format, so all request headers should allow the `application/json` content type.

The server includes a servlet extension that provides read and write access to the server's configuration over HTTP. The extension is enabled by default for new installations, and can be enabled for existing deployments by simply adding the extension to one of the server's HTTP Connection Handlers, as follows:

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --add http-servlet-extension:Configuration
```

The API is made available on the HTTPS Connection handler's `host:port` in the `/config` context. Due to the potentially sensitive nature of the server's configuration, the HTTPS Connection Handler should be used, for hosting the Configuration extension.

## Authentication and Authorization

Clients must use HTTP Basic authentication to authenticate to the Configuration API. If the username value is not a DN, then it will be resolved to a DN value using the identity mapper associated with the Configuration servlet. By default, the Configuration API uses an identity mapper that allows an entry's UID value to be used as a username. To customize this behavior, either customize the default identity mapper, or specify a different identity mapper using the Configuration servlet's `identity-mapper` property. For example:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Configuration \
  --set "identity-mapper:Alternative Identity Mapper"
```

To access configuration information, users must have the appropriate privileges:

- To access the `cn=config` backend, users must have the `bypass-acl` privilege or be allowed access to the configuration using an ACI.
- To read configuration information, users must have the `config-read` privilege.
- To update the configuration, users must have the `config-write` privilege.

## Relationship Between the Configuration API and the dsconfig Tool

The Configuration API is designed to mirror the `dsconfig` tool, using the same names for properties and object types. Property names are presented as hyphen case in `dsconfig` and as camel-case attributes in the API. In API requests that specify property names, case is not important. Therefore, `baseDN` is the same as `baseDn`. Object types are represented in hyphen case. API paths mirror what is in `dsconfig`. For example, the `dsconfig list-connection-handlers` command is analogous to the API's `/config/connection-handlers` path. Object types that appear in the schema URNs adhere to a `type:subtype` syntax. For example, a Local

## Chapter 3: Managing the Metrics Engine

DB Backend's schema URN is `urn:unboundid:schemas:configuration:2.0:backend:local-db`. Like the `dsconfig` tool, all configuration updates made through the API are recorded in `logs/config-audit.log`.

The API includes the filter, sort, and pagination query parameters described by the SCIM specification. Specific attributes may be requested using the `attributes` query parameter, whose value must be a comma-delimited list of properties to be returned, for example `attributes=baseDN,description`. Likewise, attributes may be excluded from responses by specifying the `excludedAttributes` parameter. See [Sorting and Filtering with the Configuration API](#) for more information on query parameters.

Operations supported by the API are those typically found in REST APIs:

HTTP Method	Description	Related dsconfig Example
GET	Lists the attributes of an object when used with a path representing an object, such as <code>/config/global-configuration</code> or <code>/config/backends/userRoot</code> . Can also list objects when used with a path representing a parent relation, such as <code>/config/backends</code> .	<code>get-backend-prop</code> <code>list-backends</code> <code>get-global-configuration-prop</code>
POST	Creates a new instance of an object when used with a relation parent path, such as <code>config/backends</code> .	<code>create-backend</code>
PUT	Replaces the existing attributes of an object. A PUT operation is similar to a PATCH operation, except that the PATCH is determined by determining the difference between an existing target object and a supplied source object. Only those attributes in the source object are modified in the target object. The target object is specified using a path, such as <code>/config/backends/userRoot</code> .	<code>set-backend-prop</code> <code>set-global-configuration-prop</code>
PATCH	Updates the attributes of an existing object when used with a path representing an object, such as <code>/config/backends/userRoot</code> . See <a href="#">PATCH Example</a> .	<code>set-backend-prop</code> <code>set-global-configuration-prop</code>
DELETE	Deletes an existing object when used with a path representing an object, such as <code>/config/backends/userRoot</code> .	<code>delete-backend</code>

The OPTIONS method can also be used to determine the operations permitted for a particular path.

Object names, such as `userRoot` in the Description column, must be URL-encoded in the path segment of a URL. For example, `%20` must be used in place of spaces, and `%25` is used in place of the percent (%) character. So the URL for accessing the HTTP Connection Handler object is:

```
/config/connection-handlers/http%20connection%20handler
```

### GET Example

The following is a sample GET request for information about the `userRoot` backend:

```
GET /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json
```

The response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://localhost:5033/config/backends/userRoot"
  },
  "backendID": "userRoot2",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example2,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "10",
  "dbCacheSize": "0 b",
  "dbCheckpointIntervalBytesInterval": "20 mb",
  "dbCheckpointIntervalHighPriority": "false",
  "dbCheckpointIntervalWakeupInterval": "1 m",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "0",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
  "dbNumCleanerThreads": "0",
  "dbNumLockTables": "0",
  "dbRunCleaner": "true",
  "dbTxnNoSync": "false",
  "dbTxnWriteNoSync": "true",
  "dbUseThreadLocalHandles": "true",
  "deadlockRetryLimit": "10",
  "defaultCacheMode": "cache-keys-and-values",
  "defaultTxnMaxLockTimeout": "10 s",
  "defaultTxnMinLockTimeout": "10 s",
  "enabled": "false",
  "explodedIndexEntryThreshold": "4000",
  "exportThreadCount": "0",
```

## Chapter 3: Managing the Metrics Engine

```
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "false",
"id2childrenIndexEntryLimit": "66",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"jeProperty": [
  "je.cleaner.adjustUtilization=false",
  "je.nodeMaxEntries=32"
],
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "5000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled"
}
```

### GET List Example

The following is a sample GET request for all local backends:

```
GET /config/backends
Host: example.com:5033
Accept: application/scim+json
```

The response (which has been shortened):

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 24,
  "Resources": [
    {
      "schemas": [
        "urn:unboundid:schemas:configuration:2.0:backend:ldif"
      ],
      "id": "adminRoot",
      "meta": {
```

```

    "resourceType": "LDIF Backend",
    "location": "http://localhost:5033/config/backends/adminRoot"
  },
  "backendID": "adminRoot",
  "backupFilePermissions": "700",
  "baseDN": [
    "cn=admin data"
  ],
  "enabled": "true",
  "isPrivateBackend": "true",
  "javaClass": "com.unboundid.directory.server.backends.LDIFBackend",
  "ldifFile": "config/admin-backend.ldif",
  "returnUnavailableWhenDisabled": "true",
  "setDegradedAlertWhenDisabled": "false",
  "writabilityMode": "enabled"
},
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:trust-store"
  ],
  "id": "ads-truststore",
  "meta": {
    "resourceType": "Trust Store Backend",
    "location": "http://localhost:5033/config/backends/ads-truststore"
  },
  "backendID": "ads-truststore",
  "backupFilePermissions": "700",
  "baseDN": [
    "cn=ads-truststore"
  ],
  "enabled": "true",
  "javaClass":
"com.unboundid.directory.server.backends.TrustStoreBackend",
    "returnUnavailableWhenDisabled": "true",
    "setDegradedAlertWhenDisabled": "true",
    "trustStoreFile": "config/server.keystore",
    "trustStorePin": "*****",
    "trustStoreType": "JKS",
    "writabilityMode": "enabled"
  },
  {
    "schemas": [
      "urn:unboundid:schemas:configuration:2.0:backend:alarm"
    ],
    "id": "alarms",
    "meta": {
      "resourceType": "Alarm Backend",
      "location": "http://localhost:5033/config/backends/alarms"
    },
    ...

```

### PATCH Example

Configuration can be modified using the HTTP PATCH method. The PATCH request body is a JSON object formatted according to the SCIM patch request. The Configuration API, supports a subset of possible values for the `path` attribute, used to indicate the configuration attribute to modify.

The configuration object's attributes can be modified in the following ways. These operations are analogous to the `dsconfig modify-[object]` options.

- An operation to set the single-valued `description` attribute to a new value:

```
{
  "op" : "replace",
  "path" : "description",
  "value" : "A new backend."
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --set "description:A new backend"
```

- An operation to add a new value to the multi-valued `jeProperty` attribute:

```
{
  "op" : "add",
  "path" : "jeProperty",
  "value" : "je.env.backgroundReadLimit=0"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --add je-property:je.env.backgroundReadLimit=0
```

- An operation to remove a value from a multi-valued property. In this case, `path` specifies a SCIM filter identifying the value to remove:

```
{
  "op" : "remove",
  "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.cleaner.adjustUtilization=false
```

- A second operation to remove a value from a multi-valued property, where the `path` specifies both an attribute to modify, and a SCIM filter whose attribute is `value`:

```
{
  "op" : "remove",
  "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --remove je-property:je.nodeMaxEntries=32
```

- An option to remove one or more values of a multi-valued attribute. This has the effect of restoring the attribute's value to its default value:

```
{
  "op" : "remove",
  "path" : "id2childrenIndexEntryLimit"
}
```

is analogous to:

```
$ dsconfig set-backend-prop --backend-name userRoot \
  --reset id2childrenIndexEntryLimit
```

The following is the full example request. The API responds with the entire modified configuration object, which may include a SCIM extension attribute `urn:unboundid:schemas:configuration:messages` containing additional instructions:

Example request:

```
PATCH /config/backends/userRoot
Host: example.com:5033
Accept: application/scim+json

{
  "schemas" : [ "urn:ietf:params:scim:api:messages:2.0:PatchOp" ],
  "Operations" : [ {
    "op" : "replace",
    "path" : "description",
    "value" : "A new backend."
  }, {
    "op" : "add",
    "path" : "jeProperty",
    "value" : "je.env.backgroundReadLimit=0"
  }, {
    "op" : "remove",
    "path" : "[jeProperty eq \"je.cleaner.adjustUtilization=false\"]"
  }, {
    "op" : "remove",
    "path" : "jeProperty[value eq \"je.nodeMaxEntries=32\"]"
  }, {
    "op" : "remove",
    "path" : "id2childrenIndexEntryLimit"
  } ]
}
```

Example response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ]
}
```



```
,
"id": "userRoot2",
"meta": {
  "resourceType": "Local DB Backend",
  "location": "http://example.com:5033/config/backends/userRoot2"
},
"backendID": "userRoot2",
"backgroundPrime": "false",
"backupFilePermissions": "700",
"baseDN": [
  "dc=example2,dc=com"
],
"checkpointOnCloseCount": "2",
"cleanerThreadWaitTime": "120000",
"compressEntries": "false",
"continuePrimeAfterCacheFull": "false",
"dbBackgroundSyncInterval": "1 s",
"dbCachePercent": "10",
"dbCacheSize": "0 b",
"dbCheckpointIntervalBytes": "20 mb",
"dbCheckpointHighPriority": "false",
"dbCheckpointWakeupInterval": "1 m",
"dbCleanOnExplicitGC": "false",
"dbCleanerMinUtilization": "75",
"dbCompactKeyPrefixes": "true",
"dbDirectory": "db",
"dbDirectoryPermissions": "700",
"dbEvictorCriticalPercentage": "0",
"dbEvictorLruOnly": "false",
"dbEvictorNodesPerScan": "10",
"dbFileCacheSize": "1000",
"dbImportCachePercent": "60",
"dbLogFileMax": "50 mb",
"dbLoggingFileHandlerOn": "true",
"dbLoggingLevel": "CONFIG",
"dbNumCleanerThreads": "0",
"dbNumLockTables": "0",
"dbRunCleaner": "true",
"dbTxnNoSync": "false",
"dbTxnWriteNoSync": "true",
"dbUseThreadLocalHandles": "true",
"deadlockRetryLimit": "10",
"defaultCacheMode": "cache-keys-and-values",
"defaultTxnMaxLockTimeout": "10 s",
"defaultTxnMinLockTimeout": "10 s",
"description": "123",
"enabled": "false",
"explodedIndexEntryThreshold": "4000",
"exportThreadCount": "0",
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
```

```

"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "false",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"jeProperty": [
  "\"je.env.backgroundReadLimit=0\""
],
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "5000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled",
"urn:unboundid:schemas:configuration:messages:2.0": {
  "requiredActions": [
    {
      "property": "jeProperty",
      "type": "componentRestart",
      "synopsis": "In order for this modification to take effect,
        the component must be restarted, either by disabling and
        re-enabling it, or by restarting the server"
    },
    {
      "property": "id2childrenIndexEntryLimit",
      "type": "other",
      "synopsis": "If this limit is increased, then the contents
        of the backend must be exported to LDIF and re-imported to
        allow the new limit to be used for any id2children keys
        that had already hit the previous limit."
    }
  ]
}
}

```

### API Paths

The Configuration API is available under the `/config` path. A full listing of root sub-paths can be obtained from the `/config/ResourceTypes` endpoint:

```
GET /config/ResourceTypes
Host: example.com:5033
Accept: application/scim+json
```

Sample response (abbreviated):

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:ListResponse"
  ],
  "totalResults": 520,
  "Resources": [
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "dsee-compat-access-control-handler",
      "name": "DSEE Compat Access Control Handler",
      "description": "The DSEE Compat Access Control
        Handler provides an implementation that uses syntax
        compatible with the Sun Java System Directory Server
        Enterprise Edition access control handler.",
      "endpoint": "/access-control-handler",
      "meta": {
        "resourceType": "ResourceType",
        "location": "http://example.com:5033/config/ResourceTypes/dsee-compat-
access-control-handler"
      }
    },
    {
      "schemas": [
        "urn:ietf:params:scim:schemas:core:2.0:ResourceType"
      ],
      "id": "access-control-handler",
      "name": "Access Control Handler",
      "description": "Access Control Handlers manage the
        application-wide access control. The server's access
        control handler is defined through an extensible
        interface, so that alternate implementations can be created.
        Only one access control handler may be active in the server
        at any given time.",
      "endpoint": "/access-control-handler",
      "meta": {
        "resourceType": "ResourceType",
        "location": "http://example.com:5033/config/ResourceTypes/access-
control-handler"
      }
    }
  ]
}
```

```

    }
  },
  {
    ...

```

The response's `endpoint` elements enumerate all available sub-paths. The path `/config/access-control-handler` in the example can be used to get a list of existing access control handlers, and create new ones. A path containing an object name like `/config/backends/{backendName}`, where `{backendName}` corresponds to an existing backend (such as `userRoot`) can be used to obtain an object's properties, update the properties, or delete the object.

Some paths reflect hierarchical relationships between objects. For example, properties of a local DB VLV index for the `userRoot` backend are available using a path like `/config/backends/userRoot/local-db-indexes/uid`. Some paths represent singleton objects, which have properties but cannot be deleted nor created. These paths can be differentiated from others by their singular, rather than plural, relation name (for example `global-configuration`).

## Sorting and Filtering Configuration Objects

The Configuration API supports SCIM parameters for filter, sorting, and pagination. Search operations can specify a SCIM filter used to narrow the number of elements returned. See the SCIM specification for the full set of operations for SCIM filters. Clients may also specify sort parameters, or paging parameters. As previously mentioned, clients may specify attributes to include or exclude in both get and list operations.

GET Parameters for Sorting and Filtering

GET Parameter	Description
filter	Values can be simple SCIM filters such as <code>id eq "userRoot"</code> or compound filters like <code>meta.resourceType eq "Local DB Backend"</code> and <code>baseDn co "dc=example,dc=com"</code> .
sortBy	Specifies a property value by which to sort.
sortOrder	Specifies either <code>ascending</code> or <code>descending</code> alphabetical order.
startIndex	1-based index of the first result to return.
count	Indicates the number of results per page.

## Updating Properties

The Configuration API supports the HTTP PUT method as an alternative to modifying objects with HTTP PATCH. With PUT, the server computes the differences between the object in the request with the current version in the server, and performs modifications where necessary. The server will never remove attributes that are not specified in the request. The API responds with the entire modified object.

Request:

```

PUT /config/backends/userRoot
Host: example.com:5033

```

## Chapter 3: Managing the Metrics Engine

```
Accept: application/scim+json
{
  "description" : "A new description."
}
```

### Response:

```
{
  "schemas": [
    "urn:unboundid:schemas:configuration:2.0:backend:local-db"
  ],
  "id": "userRoot",
  "meta": {
    "resourceType": "Local DB Backend",
    "location": "http://example.com:5033/config/backends/userRoot"
  },
  "backendID": "userRoot",
  "backgroundPrime": "false",
  "backupFilePermissions": "700",
  "baseDN": [
    "dc=example,dc=com"
  ],
  "checkpointOnCloseCount": "2",
  "cleanerThreadWaitTime": "120000",
  "compressEntries": "false",
  "continuePrimeAfterCacheFull": "false",
  "dbBackgroundSyncInterval": "1 s",
  "dbCachePercent": "25",
  "dbCacheSize": "0 b",
  "dbCheckpointIntervalBytesInterval": "20 mb",
  "dbCheckpointIntervalHighPriority": "false",
  "dbCheckpointIntervalWakeupInterval": "30 s",
  "dbCleanOnExplicitGC": "false",
  "dbCleanerMinUtilization": "75",
  "dbCompactKeyPrefixes": "true",
  "dbDirectory": "db",
  "dbDirectoryPermissions": "700",
  "dbEvictorCriticalPercentage": "5",
  "dbEvictorLruOnly": "false",
  "dbEvictorNodesPerScan": "10",
  "dbFileCacheSize": "1000",
  "dbImportCachePercent": "60",
  "dbLogFileMax": "50 mb",
  "dbLoggingFileHandlerOn": "true",
  "dbLoggingLevel": "CONFIG",
  "dbNumCleanerThreads": "1",
  "dbNumLockTables": "0",
  "dbRunCleaner": "true",
  "dbTxnNoSync": "false",
  "dbTxnWriteNoSync": "true",
  "dbUseThreadLocalHandles": "true",
```

```

"deadlockRetryLimit": "10",
"defaultCacheMode": "cache-keys-and-values",
"defaultTxnMaxLockTimeout": "10 s",
"defaultTxnMinLockTimeout": "10 s",
"description": "abc",
"enabled": "true",
"explodedIndexEntryThreshold": "4000",
"exportThreadCount": "0",
"externalTxnDefaultBackendLockBehavior": "acquire-before-retries",
"externalTxnDefaultMaxLockTimeout": "100 ms",
"externalTxnDefaultMinLockTimeout": "100 ms",
"externalTxnDefaultRetryAttempts": "2",
"hashEntries": "true",
"importTempDirectory": "import-tmp",
"importThreadCount": "16",
"indexEntryLimit": "4000",
"isPrivateBackend": "false",
"javaClass": "com.unboundid.directory.server.backends.jeb.BackendImpl",
"numRecentChanges": "50000",
"offlineProcessDatabaseOpenTimeout": "1 h",
"primeAllIndexes": "true",
"primeMethod": [
  "none"
],
"primeThreadCount": "2",
"primeTimeLimit": "0 ms",
"processFiltersWithUndefinedAttributeTypes": "false",
"returnUnavailableForUntrustedIndex": "true",
"returnUnavailableWhenDisabled": "true",
"setDegradedAlertForUntrustedIndex": "true",
"setDegradedAlertWhenDisabled": "true",
"subtreeDeleteBatchSize": "5000",
"subtreeDeleteSizeLimit": "100000",
"uncachedId2entryCacheMode": "cache-keys-only",
"writabilityMode": "enabled"
}

```

## Administrative Actions

Updating a property may require an administrative action before the change can take effect. If so, the server will return 200 Success, and any actions are returned in the `urn:unboundid:schemas:configuration:messages:2.0` section of the JSON response that represents the entire object that was created or modified.

For example, changing the `jeProperty` of a backend will result in the following:

```

"urn:unboundid:schemas:configuration:messages:2.0": {
  "requiredActions": [
    {
      "property": "baseContextPath",
      "type": "componentRestart",

```

```

    "synopsis": "In order for this modification to
      take effect, the component must be restarted,
      either by disabling and re-enabling it, or by
      restarting the server"
  },
  {
    "property": "id2childrenIndexEntryLimit",
    "type": "other",
    "synopsis": "If this limit is increased, then the
      contents of the backend must be exported to LDIF
      and re-imported to allow the new limit to be used
      for any id2children keys that had already hit the
      previous limit."
  }
]
}
...

```

### Updating Servers and Server Groups

Servers can be configured as part of a server group, so that configuration changes that are applied to a single server, are then applied to all servers in a group. When managing a server that is a member of a server group, creating or updating objects using the Configuration API requires the `applyChangeTo` query parameter. The behavior and acceptable values for this parameter are identical to the `dsconfig` parameter of the same name. A value of `singleServer` or `serverGroup` can be specified. For example:

```
https://example.com:5033/config/Backends/userRoot?applyChangeTo=singleServer
```

#### **Note**

This does not apply to mirrored subtree objects, which include Topology and Cluster level objects. Changes made to mirrored objects are applied to all objects in the subtree.

### Configuration API Responses

Clients of the API should examine the HTTP response code in order to determine the success or failure of a request. The following are response codes and their meanings:

Response Code	Description	Response Body
200 Success	The requested operation succeeded, with the response body being the configuration object that was created or modified. If further actions are required, they are included in the <code>urn:unboundid:schemas:configuration:messages:2.0</code> object.	List of objects, or object properties, administrative actions.
204 No Content	The requested operation succeeded and no further information has been provided, such as in the case of a DELETE operation.	None.
400 Bad Request	The request contents are incorrectly formatted or a request is made for an invalid API version.	Error summary and optional

Response Code	Description	Response Body
		message.
401 Unauthorized	User authentication is required. Some user agents such as browsers may respond by prompting for credentials. If the request had specified credentials in an Authorization header, they are invalid.	None.
403 Forbidden	The requested operation is forbidden either because the user does not have sufficient privileges or some other constraint such as an object is edit-only and cannot be deleted.	None.
404 Not Found	The requested path does not refer to an existing object or object relation.	Error summary and optional message.
409 Conflict	The requested operation could not be performed due to the current state of the configuration. For example, an attempt was made to create an object that already exists or an attempt was made to delete an object that is referred to by another object.	Error summary and optional message.
415 Unsupported Media Type	The request is such that the Accept header does not indicate that JSON is an acceptable format for a response.	None.
500 Server Error	The server encountered an unexpected error. Please report server errors to customer support.	Error summary and optional message.

An application that uses the Configuration API should limit dependencies on particular text appearing in error message content. These messages may change, and their presence may depend on server configuration. Use the HTTP return code and the context of the request to create a client error message. The following is an example encoded error message:

```
{
  "schemas": [
    "urn:ietf:params:scim:api:messages:2.0:Error"
  ],
  "status": 404,
  "scimType": null,
  "detail": "The Local DB Index does not exist."
}
```

## Domain Name Service (DNS) Caching

If needed, two global configuration properties can be used to control the caching of hostname-to-numeric IP address (DNS lookup) results returned from the name resolution services of the underlying operating system. Use the `dsconfig` tool to configure these properties.

**network-address-cache-ttl**– Sets the Java system property `networkaddress.cache.ttl`, and controls the length of time in seconds that a hostname-to-IP address mapping can be cached. The default behavior is to keep resolution results for one hour (3600 seconds). This setting applies to the server and all extensions loaded by the server.



**network-address-outage-cache-enabled** – Caches hostname-to-IP address results in the event of a DNS outage. This is set to `true` by default, meaning name resolution results are cached. Unexpected service interruptions may occur during planned or unplanned maintenance, network outages or an infrastructure attack. This cache may allow the server to function during a DNS outage with minimal impact. This cache is not available to server extensions.

## IP Address Reverse Name Lookups

UnboundID servers do not explicitly perform numeric IP address-to-hostname lookups. However, address masks configured in Access Control Lists (ACIs), Connection Handlers, Connection Criteria, and Certificate handshake processing may trigger implicit reverse name lookups. For more information about how address masks are configured in the server, review the following information for each server:

- ACI dns: bind rules under *Managing Access Control* (Data Store and Proxy Servers)
- ds-auth-allowed-address: *Adding Operational Attributes that Restrict Authentication* (Data Store)
- Connection Criteria: *Restricting Server Access Based on Client IP Address* (Data Store and Proxy Servers)
- Connection Handlers: restrict server access using Connection Handlers (Configuration Reference Guide for all servers)

## Server SDK Extensions

Custom server extensions can be created with the UnboundID Corp Server SDK. Extension bundles are installed from a .zip archive or a file system directory. Use the `manage-extension` tool to install or update any extension that is packaged using the extension bundle format. It opens and loads the extension bundle, confirms the correct extension to install, stops the server if necessary, copies the bundle to the server install root, and then restarts the server.

### Note

The `manage-extension` tool must be used with Java extensions packaged using the extension bundle format. For more information, see the "Building and Deploying Java-Based Extensions" section of the Server SDK documentation.

The UnboundID Corp Server SDK enables creating extensions for all UnboundID Corp servers. Cross-product extensions include:

- Access Loggers
- Alert Handlers
- Error Loggers
- Key Manager Providers
- Monitor Providers

- Trust Manager Providers
- OAuth Token Handlers
- Manage Extension Plugins

---

## Chapter 4: Collecting Data and Metrics

---

The Metrics Engine polls all the monitored servers over LDAP to gather the following data:

- Status of each server.
- Alerts emitted by each server.
- Performance data exposed in the `cn=monitor` subtree of each server.

For a complete summary of the metrics and dimensions that can be exposed through the RESTful Metrics API, see the reference files located in the `docs/metrics-guide` directory. Most metrics have a count, minimum, maximum, and average.

Topics include:

[Metrics Overview](#)

[Query Overview](#)

[The query-metric Tool](#)

[Performance Data Collection](#)

[System Monitoring Data Collection](#)

[Server Clock Skew](#)

[Tune Data Collection](#)

[Data Processing](#)

[Monitoring for Service Level Agreements](#)

## Metrics Overview

A metric corresponds to a single measurement made within the server. The Metrics Engine collects three types of metrics:

- **Count metrics** – represent the number of times a specific event happens within the server. Examples of count metrics include the number of LDAP operations performed, network packets received, or new connections established.
- **Continuous-valued metrics** – measure things that always have a value. For example, these metrics include the amount of free disk space, the current number of connected clients, and the number of operations pending in the work queue.
- **Discrete metrics** – correspond to measurements that have both a value and a weight, such as the duration of an LDAP operation or the average duration of a checkpoint.

The statistics that can be applied to values depend on the metric type. Only count statistics are available for count metrics. Discrete metrics have count, average, and histogram statistics available, which expose a count of the values broken down into bucket ranges. Average, minimum, and maximum statistics are available for continuous-valued metrics.

### Count Metrics

A count metric indicates the number of times a specific event happens within the server. For example, the number of packets received on a network interface during a measurement interval is a count metric. Each measurement returns the count of the number of packets received during that measurement interval only. The sample contains the number of occurrences, whether the measurement interval is five seconds or two minutes.

Another example of a count metric is the number of megabytes of data written to a disk device during a measurement interval. Using the COUNT statistic when querying for a count metric will return the sum of the counts. Count metrics can often be converted into a rate.

### Continuous Metrics

A continuous metric is a measurement of a value where the thing being measured always has a valid value at each measurement point. For example, CPU percent busy is a continuous metric. For every sample CPU interval, a valid CPU percent busy measurement can be taken. A continuous metric differs from a count metric in that continuous metric samples cannot be added across time in a meaningful way. Instead, continuous metric samples use average, minimum, and maximum statistics. To determine how busy the CPU has been since midnight, average, rather than sum, the samples since midnight.

### Discrete Metrics

A discrete metric is a measurement that has both a value and a weight. Discrete metrics are different from continuous metrics because each measurement is weighted. A discrete metric is

analogous to a weighted average and requires that multiple measurements be taken within a single sample interval. For example, LDAP operation response time is a discrete metric, where the actual response time of each operation is averaged, and the number of LDAP operations is provided as the weight. If no LDAP operations occur in a sample interval, the value would be zero and the weight would be zero.

Some continuous and discrete metrics may also report a minimum/maximum value if the measurement is composed of multiple sub-measurements. The minimum/maximum values are aggregated by averaging, so the values reflect the median.

Some discrete metrics may also convey histogram data. Histogram data represents an additional set of measurements that take individual measurements and place them into value ranges. The Metrics Engine supports histograms with up to 15 value ranges. Histogram valued samples are unique because they give a picture of the distribution of the values, and because they more precisely answer the question of "How many samples are greater than X?"

## Dimensions

Dimensions provide a means of aggregating and subdividing metric sample values in a way that logically follows what is actually measured. For example, metrics that measure disk activity have a `disk-device` dimension. Aggregating on the `disk-device` dimension shows the average disk activity for all disks, where pivoting (splitting) by the `disk-device` dimension shows the activity for specific disks.

Every metric has a logical instance dimension, which corresponds to the server on which the sample was created. Each metric may have up to three dimensions, which are defined in the metric definition.

For example, the `sync-pipe-completed-ops` metric has two dimensions, the `pipe-name` and `pipe-result`. The `pipe-name` is the name of the sync pipe as configured for the Data Sync Server. The `pipe-result` is one of the following values:

- `exception`
- `failed`
- `failed-at-resource`
- `failed-during-mapping`
- `match-multiple-at-dest`
- `no-match-at-dest`
- `already-exists-at-dest`
- `no-change-needed`
- `out-of-scope`
- `success`
- `aborted-by-plugin`
- `failed-in-plugin`

## Chapter 4: Collecting Data and Metrics

At each measurement interval for each sync pipe on each Data Sync Server, there will be a value for each of the `pipe-result` values. So, for a single Data Sync Server with two Sync Pipes, `pipe-one` and `pipe-two`, the samples generated for each sample period look like the following. The timestamp is constrained to time-only for brevity.

```
08:15:05, sync-pipe-completed-ops, pipe-one, exception, 1
08:15:05, sync-pipe-completed-ops, pipe-one, failed, 7
08:15:05, sync-pipe-completed-ops, pipe-one, failed-at-resource, 1
08:15:05, sync-pipe-completed-ops, pipe-one, failed-during-mapping, 1
08:15:05, sync-pipe-completed-ops, pipe-one, match-multiple-at-dest, 3
08:15:05, sync-pipe-completed-ops, pipe-one, no-match-at-dest, 0
08:15:05, sync-pipe-completed-ops, pipe-one, already-exists-at-dest, 0
08:15:05, sync-pipe-completed-ops, pipe-one, no-change-needed, 1
08:15:05, sync-pipe-completed-ops, pipe-one, out-of-scope, 1
08:15:05, sync-pipe-completed-ops, pipe-one, success, 125
08:15:05, sync-pipe-completed-ops, pipe-one, aborted-by-plugin, 1
08:15:05, sync-pipe-completed-ops, pipe-one, failed-in-plugin, 0
08:15:05, sync-pipe-completed-ops, pipe-two, exception, 3
08:15:05, sync-pipe-completed-ops, pipe-two, failed, 9
08:15:05, sync-pipe-completed-ops, pipe-two, failed-at-resource, 2
08:15:05, sync-pipe-completed-ops, pipe-two, failed-during-mapping, 1
08:15:05, sync-pipe-completed-ops, pipe-two, match-multiple-at-dest, 4
08:15:05, sync-pipe-completed-ops, pipe-two, no-match-at-dest, 0
08:15:05, sync-pipe-completed-ops, pipe-two, already-exists-at-dest, 0
08:15:05, sync-pipe-completed-ops, pipe-two, no-change-needed, 1
08:15:05, sync-pipe-completed-ops, pipe-two, out-of-scope, 1
08:15:05, sync-pipe-completed-ops, pipe-two, success, 217
08:15:05, sync-pipe-completed-ops, pipe-two, aborted-by-plugin, 1
08:15:05, sync-pipe-completed-ops, pipe-two, failed-in-plugin, 0
```

Compare how busy `pipe-one` is to `pipe-two` by pivoting on `pipe-name`. This results in the following:

```
pipe-one 141
pipe-two 239
```

Pivot by `pipe-result`, to get a set of counts that show the distribution of the counts of the specific error types, as well as the success and failure. This data provides a quick way of assessing the kinds of problems encountered by the Sync Pipes.

Dimensions provide a way to pivot or aggregate along a metric-specific axis. All metrics have the `instance` pivot and the `time` pivot. Metrics that support the histogram statistic can also have a `histogram` pivot.

## Query Overview

A metric query consists of three components:

- The data used to calculate the query results.
- The aggregation method used on the data to calculate the query result.

- The format of the query result.

## Select Query Data

The data used to generate the results of a metric query are driven by the following factors:

- Metric and statistic
- Time range
- Server instances included in the result (optional)
- Included dimension values (optional)
- Histogram range (optional)

Every query returns results for a single statistic and of a single metric. A query must include the time range used to generate the results. Time ranges can either be absolute dates (in ISO-8601 format) or relative dates (such as -30m). A relative start time offset is relative to the end time. A relative end time offset is relative to the current time. When no end time is specified, the server includes results up to the current time.

The time range and the desired number of points (for pivot by time) dictates the resolution of data used to process the query. For example, the finest granularity of data, one second resolution, is only kept for a few hours. It will not be used to satisfy a query spanning multiple days.

By default, all server instances that produce the metric are used to calculate the query results. However, the metric query can be restricted to one of the following:

- A specific list of servers
- Servers of a given type, such as Data Stores
- Servers within a specific location

For metrics that include one or more dimensions, a query can be evaluated across a subset of dimension values. For example, the results returned for the `response-time` metric can be restricted to just the search and modify values of the `op-type` dimension.

For `discrete-valued` metrics that break their values down into histogram ranges, a query can count statistics applied to a subset of histogram buckets by specifying a minimum and/or maximum histogram value. For example, a query on the `response-time` metric could return a count of operations that took longer than 100 milliseconds.

## Aggregate Query Results

A metric query can return the full, raw data that matches the query parameters, so that the server can aggregate metric results across time, server instance, dimension value, or histogram value. The server aggregates results, except when the query indicates not to, by using a pivot. The mechanism for aggregating the data depends on the type of metric. A pivot directs the query processor to not aggregate one component of the query data. A pivot can be based on time, server instance, a specific dimension, or histogram ranges.

- If no pivot is specified, the query returns a single number that represents the aggregation of all matched data. For example, a query with no pivot might return the total number of operations that have completed today.
- A single pivot results in one-dimensional data, such as a time-based chart with a single line or a simple bar chart.
- Two pivots results in two-dimensional data, such as a time-based chart with a separate line for each server instance, or a stacked bar chart that shows the number of completed operations broken down by server and operation type.
- Three pivots results in three-dimensional data, such as a stacked, grouped bar chart that shows completed operations broken down by server, operation type, and result.

Beyond aggregating multiple samples into one, the data returned by a metric query can be further manipulated. For example, queries can be scaled on the count statistic to return the count of events per second, per minute, or per hour. Counts of histogram values can be returned by a percentage of the total. For example, instead of returning the raw count of operations that took longer than 50 milliseconds to complete, the results could be returned as the percentage of all operations that took longer than 50 milliseconds to complete. A value of 0.02% is more meaningful than a value of 40.

### Format Query Results

The query results can be converted into a format requested by the client, such as:

- CSV spreadsheet
- PNG or a JPG chart
- XML format
- JSON format

## The query-metric Tool

The `query-metric` tool is a client application of the Metrics Engine API that enables access to all the metrics gathered by the server. It includes subcommands that facilitate creating data queries for listing metrics, server instances, and dimension values. This tool runs in both interactive and non-interactive modes. Queries are formed using the following subcommands:

- `explore` – Creates a series of hyper-linked HTML files containing charts for a broad range of metrics. The tool generates these files by making a series of API queries for a set of servers and metrics. The tool highlights the breadth of available metrics and patterns or anomalies across multiple metrics. In interactive mode, the tool prompts for the servers and the metrics.
- `query` – Defines a query for specific data of interest. In interactive mode, the tool prompts for the server, metrics, dimensions, statistics, and pivot values. The tool can be



used to request a server generated chart image file or data formatted in XML, JSON, or CSV.

To start the tool in interactive mode, enter the following command:

```
$ query-metric
```

Or, specify a subcommand in interactive mode:

```
$ query-metric explore
```

In non-interactive mode, the tool generates charts based on command-line input. For example, the following command requests information from the local Metrics Engine listening on port 8080 and generates response-time and throughput charts for Proxy Server instances in Austin for the previous two weeks:

```
$ query-metric explore \
--httpPort 8080 \
--instanceType proxy \
--instanceLocation Austin \
--metric response-time \
--metric throughput \
--startTime -2w
```

The following command line obtains a JSON formatted data table that shows average throughput for all Proxy Server instances, over time with 100 data points. Each line in the chart represents either an application's search or modification throughput. Throughput values are represented as operations per second:

```
$ query-metric query \
--hostname localhost \
--httpPort 8080 \
--username cn=user1,cn=api-users \
--password secret \
--table json \
--metric throughput \
--instanceType proxy \
--statistic average \
--pivot op-type \
--pivot application-name \
--dimension op-type:search,modify \
--rateScaling second \
--maxIntervals 100 \
--startTime 2012-09-01T17:41Z \
--endTime 2012-09-30T17:41Z
```

To see a list of all supported options, run the help option for the `query-metric` tool:

```
$ query-metric -?
```

## Performance Data Collection

Performance data represents a majority of the data collected by the Metrics Engine. Each server may produce hundreds of kilobytes of performance data per minute, though the amount of data captured has little to no impact on the performance of the monitored system. By default, the Metrics Engine stores performance data for 20 years. Configure the volume of performance data collected by each monitored server so that the Metrics Engine can keep up with the flow.

The performance data model is a dimensional data model. Measurements can be taken on multiple simultaneous values that are distinguished by dimension values. For example, a response time metric provides the time in milliseconds it took a server to respond to an LDAP request. This `response-time` metric has two dimensions:

**Application name** – reflects the connection criteria of the request.

**Operation type** – corresponds to the LDAP operation, such as add, bind, or search.

If a server has 20 different connection criteria, each response-time sample may have 140 different values, one for each of the applications multiplied by the number of operation types.

The performance data captured on the monitored server has a record with the following fields.

**Performance Data Fields**

Name	Data Type	Description
Timestamp	Date	Time of measurement, using clock on the monitored server
Metric	String	Name of metric
Dimension	String	Values of dimensions 1 - 3
Count	Int	Number of measurements represented by this sample
Average	Double	Average value of this sample
Minimum	Double	Optional minimum value of this sample
Maximum	Double	Optional maximum value of this sample
Buckets	Int	Optional histogram data associated with this sample

When a performance record is imported into the Metrics Engine, it is normalized to reduce the size of the record. The normalized record contains the following information.

**Normalized Record in the Metrics Engine**

Name	Data Type	Description
batchID	Int	The ID of the batch of data to which this record belongs
sampleTime	Timestamp	The time the sample was captured or equivalent information after aggregation
metric_qual	Int	The ID of a structure that reflects the metric and all dimension values
definitionID	Int	ID of the histogram definition, if the data belong to a histogram-valued sample
count	Int	Number of measurements represented by this sample
avg_val	Real	Average value for this sample
min_val	Real	Minimum value for this sample

## Normalized Record in the Metrics Engine

Name	Data Type	Description
max_val	Real	Maximum value for this sample
val1-15	Long	Histogram bucket values

## System Monitoring Data Collection

All servers have the ability to monitor their health and that of the host system. Servers do not collect any performance data until they are prepared by the Metrics Engine. All of the important server and machine metrics are stored in the `cn=monitor` backend.

### Stats Collector Plugin

The Stats Collector plugin is the primary driver of performance data collection for LDAP, server response, replication, local JE databases, and host system machine metrics. Stats Collector configuration determines the sample and collection intervals, granularity of data (basic, extended, verbose), types of host system collection (cpu, disk, network) and the type of data aggregation that occurs for LDAP application statistics. The Stats Collector plugin is configured with the `dsconfig` tool and collects data using LDAP queries. For example, the `--server-info:extended` option includes collection for the following:

- CPU
- JVM memory
- Memory
- Disk information
- Network information

The following are all options for the Stats Collector plugin:

```
>>>> Configure the properties of the Stats Collector Plugin
Property                                Value(s)
-----
1)  description                          -
2)  enabled                              false
3)  local-db-backend-info                 basic
4)  replication-info                     basic
5)  entry-cache-info                     basic
6)  host-info                            cpu, disk, network
7)  included-ldap-application             If per-application LDAP stats is enabled,
then stats will be included for all
applications.
8)  sample-interval                       1 s
9)  collection-interval                   500 ms
10) ldap-info                             extended
11) server-info                           basic
12) per-application-ldap-stats             aggregate-only
```

## System Utilization Monitors

The System Utilization Monitors interface directly with the host operating system to gather statistics about CPU utilization and idle states, memory consumption, disk input and output rates, and queue depths, as well as network packet transmit and receive activity.

Utilization metrics are gathered with externally invoked operating system commands, such as `iostat` and `netstat`, using platform-specific arguments and version-specific output parsing.

Enabling the Host System monitor provider automatically gathers CPU and memory utilization, but only optionally gathers disk and network information. Disk and network interfaces are enumerated in the configuration by device names (such as `eth0` or `lo`), and by disk device names (such as `sd1`, `sdab`, `sda2`, `scsi0`).

## External Collector Daemon

The System Utilization monitor contains an embedded collector daemon that runs on systems affected by a Java process fork memory issue (RFE 5049299). When a process attempts to fork a child process, Solaris attempts to allocate the same amount of memory for the child process, which will likely fail when the parent process consumes a large amount of memory.

The embedded collector daemon is started automatically for the server and inspects the Host System Monitor provider configuration to conditionally determine whether the external daemon process is required.

The external collector daemon operates by having an internal table of repeatable commands that run on a schedule. The collector creates a simulated filesystem in the `<server-root>/logs` directory for each command type so that the Host System Monitor Provider can find the output of the most recently collected data.

Repeating commands use a subdirectory for each command type to keep results isolated from other command types and to help organize file cleanup. The filename of the output contains the sample timestamp, such as `iostats-[sampletimestamp]`. If the collector daemon fails for any reason, the Host System Monitor provider is not left reading stale system data because the expected timestamp files is missing. To handle clock-edge timing, the monitor sampler will also look for data in a filename of the previous second. Timestamp files are deleted once their data have been collected.

The collector daemon runs with no inter-process communication and can be stopped if no longer necessary.

## Server Clock Skew

Correlating metric samples from multiple servers requires that the timestamp associated with each sample from each monitored server is synchronized. The Metrics Engine tracks system time information and makes it visible in the `cn=Monitored Server <servername>`, `cn=monitor` entry.

The `system-clock-skew-seconds` attribute indicates the difference between the Metrics Engine system clock and the monitored server clock, in seconds. The larger this skew value, the less precision there is when comparing changes in data across servers.

While it is not necessary to keep the Metrics Engine clock synchronized with all of the monitored servers, it can be convenient when issuing metric queries with time ranges specified by offsets. Because the offset is computed using the Metrics Engine system clock, if this clock is very different from the monitored servers' system clocks, the start/end time of a metric query will not match the expected boundaries.

## Tune Data Collection

Collecting all of the performance data at the most granular level from all of the servers may not be possible without a significant investment in hardware for the Metrics Engine. Instead, you can tune your data collection to fit within the limits of your existing Metrics Engine hardware.

The remainder of this section describes several strategies for tuning data collection.

### Reducing the Data Collected

If not all information collected by the Metrics Engine is required, the Stats Collector Plugin's `entry-cache` property can be tuned using the `dsconfig` command-line tool. For example, to omit all metrics related to the entry cache set the `entry-cache-info` group on the monitored server:

```
$ bin/dsconfig set-plugin-prop --plugin-name "Stats Collector" \
  --set entry-cache-info:none
```

The server collects information for eight different information groups. Limit data collection to the devices of actual interest.

### Reducing the Frequency of Data Collection

Monitored servers can produce metric samples every second, which is useful for short-duration changes. These samples are less useful hours later, after the per-second data is aggregated to per-minute data. Use the `dsconfig` tool to change the base sample production rate from the default of 1 second to 10 seconds:

```
$ bin/dsconfig set-plugin-prop --plugin-name "Stats Collector" \
  --set "sample-interval:10 seconds"
```

This change reduces the total data volume by about 90 percent.

### Reducing the Frequency of Sample Block Creation

The number of sample blocks processed by the Metrics Engine can also be reduced in a given time. By default, the monitored servers produce a new block of samples every 30 seconds. Increasing this to 60 seconds, while reducing the Metrics Engine's polling rate to 60 seconds,

reduces the sample processing overhead. Change the frequency at which the monitored servers create sample blocks using the following `dsconfig` command:

```
$ bin/dsconfig set-backend-prop --backend-name metrics \  
--set sample-flush-interval:60s
```

### Reducing Metrics Engine Impact on Performance

All UnboundID Corp servers expose performance data through the `cn=monitor` DN. Performance issues occur when data is read, either directly by an LDAP client, or by enabling either the Stats Logger or Stats Collector plugins.

The Stats Logger plugin reads the configured monitors and writes the resulting values to a CSV file. The Stats Collector plugin also reads the configured monitors and writes the resulting values to a CSV file, but this file is made available for LDAP clients in `cn=metrics` DN. The Stats Collector CSV files are suitable for use by the Metrics Engine, and contain one metric value per line.

Both the Stats Logger and the Stats Collector plugins are disabled by default. When enabled, each of these plugins adds an approximate 3% CPU utilization penalty, plus a negligible amount of disk I/O and JVM heap usage.

To enable the Stats Collector plugin, use `dsconfig` as follows:

```
$ bin/dsconfig set-plugin-prop --plugin-name "Stats Collector" \  
--set enabled:true
```

The `monitored-servers` tool will enable the Stats Collector plugin on the monitored server.

## Data Processing

When blocks of samples arrive in the Metrics Engine, they are queued on disk and loaded into the database. Samples from a single server are processed in time-order, so that sample blocks with older data are always processed before a sample block containing newer data. The Metrics Engine does not do time-correlation between blocks coming from different servers. So, server A samples from two hours ago may be loaded immediately after server B samples from two minutes ago. This flexibility enables servers to be unavailable to the Metrics Engine, without affecting the overall system monitoring. Also, a query for data from server A and B may return data for server B but not server A, until the data queued for server A has been collected and imported. Samples collected from the Metrics Engine itself are processed ahead of all other servers.

### Importing Data

The Metrics Engine polls all of the monitored servers at a regular interval. When new samples are available, the Metrics Engine fetches them through LDAP. The Metrics Engine has one dedicated thread taking sample blocks and converting them to the normalized form stored in the DBMS. The import queue's size is normally near zero, but under certain conditions it may

become large. When the Metrics Engine starts, it will queue (for import) all sample blocks still on disk. Blocks that are older than two hours are discarded.

For example, if a monitored server becomes unavailable for an extended period of time, it will continue to queue blocks of samples locally. When it becomes available again, the Metrics Engine collection poll of that server will capture hundreds or thousands of sample blocks. The Metrics Engine captures the sample blocks at a much faster speed than it can import them, causing the queue to grow for a period of time. If the Metrics Engine is stopped, this problem is compounded because all monitored servers will then have a backlog of sample blocks to be imported.

## Aggregating Data

To maintain a size-limited DBMS while accumulating data over a period of years, the Metrics Engine aggregates data into four different levels. Each level contains data with less time granularity, but covering a larger period of time. Data is aggregated from a lower (greater time granularity) to a higher level as soon as enough data for aggregation is available. For example, the level 0 data has one second granularity, and the level 1 data has one minute granularity. After level 0 has collected one minute's worth of data, the data from that minute can be aggregated to level 1.

To keep the data tables for each aggregation level at a constrained size, each aggregation level has a maximum age for the samples. When the samples are older than this age, they are deleted from the level. While aggregation occurs soon after the samples arrive in the level, pruning occurs only after all samples in a block have passed their age limit.

The Metrics Engine attempts to collect data from all configured servers as efficiently as possible. However, Monitored Server availability, DBMS backlog, and Metrics Engine load can all cause the data pipeline to slow down. The data aggregation system is designed to correctly handle gaps in the data.

The resolution of the aggregation levels cannot be changed, but the maximum age of each level can be configured. The following table lists the aggregation levels.

Aggregation Levels			
Level	Resolution	Default Maximum Age	Maximum Age
0	1 second	2 hours	48 hours
1	1 minute	7 days	34 days
2	1 hour	12 months	5 years
3	1 day	20 years	20 years

## Monitoring for Service Level Agreements

The Metrics Engine provides the ability to aggregate and track performance data for one or more service level agreements (SLAs). The server aggregates the data using an SLA object that tracks the current and historical performance of LDAP operations (throughput and response times) that are tied to specifically monitored applications. The SLA object consists of a tracked application name, one or more LDAP operations to be considered, a set of servers

that contribute performance data to the SLA, and optionally, thresholds to generate alerts should the server exceed these limits.

Thresholds are optional configuration settings that enable the monitoring of performance data. Each threshold sets a limit that indicates a warning condition where the server's performance is nearing a limit and/or a critical condition. When the monitored server enters or ends a warning or critical state, the Metrics Engine generates an alert. The generated alerts are the same as those created by the Data Store and Proxy Servers and can be routed through the Alert Handler to a monitoring console or administrator.

The SLA object can report the aggregate performance of all configured servers. The SLA object is configured with the following:

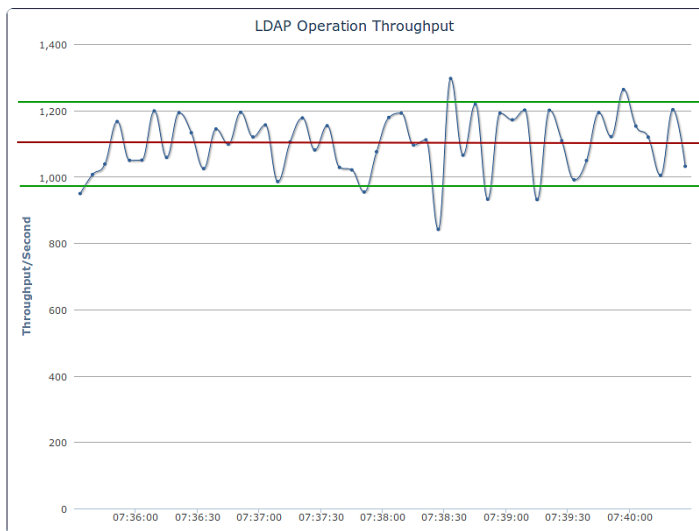
- **Designate Servers that Contribute to SLA Tracking** – The SLA object includes a Server Query component that is used to designate the servers that contribute to the SLA measurements.
- **REST API** – A REST API enables listing configured SLA objects and their current status. The Metrics Engine REST API also enables listing alerts generated by SLA thresholds, and blending the alert information with the threshold information to provide a more contextual view of the tracked applications performance.

### SLA Thresholds

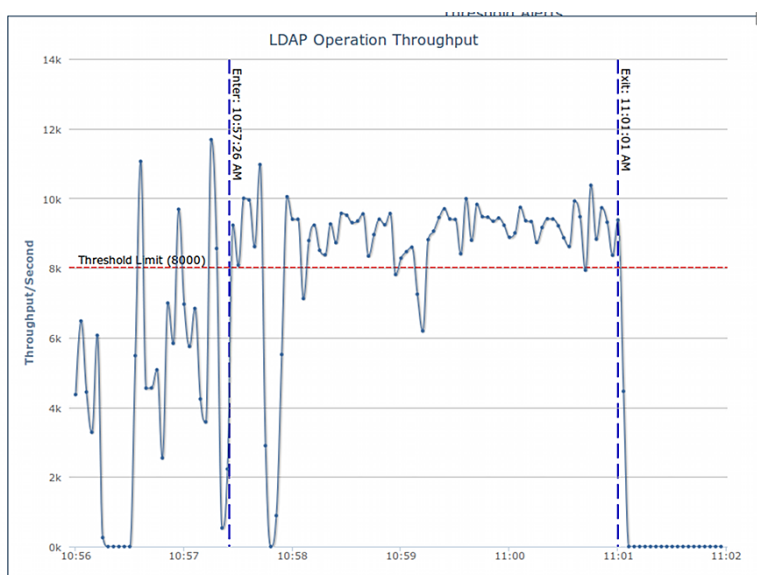
The Metrics Engine uses a Monitoring Threshold mechanism that has two components:

- **Spike Monitoring Threshold** – Used to configure a set of operational performance limits on a specific measurement, where the limit is specified as a percent change from the most recent measurement average value. A Spike Monitoring Threshold has warning and critical limits, and will enter or leave an alerted state when the monitored value exceeds either of the limits. This threshold is useful when the valid range of the measurement is not known in advance. This type of limit is useful in detecting short-term changes in a measurement that fluctuates broadly over time. The limit is applied in both positive and negative directions, so that this type of threshold can detect an upward or downward spike in the value. The following chart shows a spike monitoring threshold, where the red line is the average throughput/second and the green lines are the limits, showing the average window for the throughput.



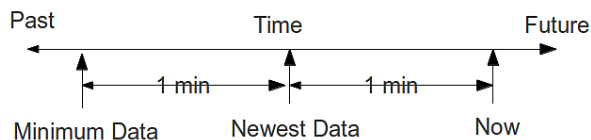


- **Static Level Monitoring Threshold** – Used to configure performance limits on a specific measurement, where the specified limits are fixed values that do not change over time. A Static Level Monitoring Threshold has warning and critical limits, and will enter or leave an alerted state when the monitored value exceeds any of the limits. The Static Level Monitoring Threshold is configured with static numeric limits, and is useful when the expected valid range of the measurement is known in advance.



## Threshold Time Line

The Metrics Engine periodically evaluates each threshold, computing current value, current average, and alerted state. The default evaluation period is 30 seconds. Using the figure below, a threshold is evaluated at time 'Now.' The most recent data that threshold uses is one minute old (Newest Data). Each threshold evaluation requires at least one minute of new data (Minimum Data). At time 'Now,' the threshold is working with data that is between one and two minutes old (between Minimum Data and Newest Data).



The one minute delay between 'Now' and 'Newest Data' is not configurable. This delay ensures that the Metrics Engine has had enough time to poll the monitored servers and get the most recent data. The one minute delay between 'Newest Data' and 'Minimum Data' is configurable on a per-threshold basis for Spike-valued thresholds, but one minute is the minimum window. Generally, time between when a monitored performance anomaly occurs on a monitored system, and when an alert is created will be between two and three minutes.

Because the Metrics Engine can capture metrics data with a very fine time resolution (one second data is the default), the data is often very "noisy." By default, the data is time-averaged (using five consecutive one-second samples to produce a single five-second value), and time-averaging will ultimately reduce the noise. However, "noisy" data can make it harder to choose an appropriate threshold limit value. If the limit value is too close to the noise levels, the threshold will alert due to values that have a very short time duration.

Each threshold is configured with a `minimum-time-to-trigger` property, which determines the minimum time allowed to exceed the threshold before an alert is generated, and a `minimum-time-to-exist` property that determines the time required for the threshold to exit an alerted state.

## Configure an SLA Object

The SLA object relies on existing performance metrics and only aggregates the data for specific SLAs. Configure any number of SLA objects for monitored servers. For more information, see the *UnboundID Metrics Engine Configuration Reference* (HTML) documentation in the `<server-root>/docs` directory.

The following steps use the `dsconfig` command-line tool:

1. Create a server query that specifies which servers will contribute to SLA monitoring. In this example, the command specifies the Proxy Servers located in Austin.

```
$ bin/dsconfig create-server-query \
  --query-name "Austin Proxy Servers" \
  --set server-instance-type:proxy \
  --set server-instance-location:Austin
```

2. Create a static-level monitoring threshold. In this example, the alert condition is set to entry, which means that the server will generate an alert if the server enters a warning state (alert-on-warn:true and warn-if-above:12) or critical state (critical-if-above:15). When the server leaves its alerted state, an alert is generated (alertcondition: exit). The minimum amount of time that the threshold can be exceeded before an alert is generated is set to 15 seconds (min-time-for-trigger:15s).

```
$ bin/dsconfig create-monitoring-threshold \
  --threshold-name "15ms response time" \
  --type static-level \
  --set alert-condition:entry \
  --set alert-condition:exit \
  --set alert-on-warn:true \
  --set min-time-for-trigger:15s \
  --set min-time-for-exit:15s \
  --set warn-if-above:12 \
  --set critical-if-above:15
```

3. Create another static-level monitoring threshold. In this example, the alert condition is set to entry. The server generates an alert if it enters a warning state (alert-on-warn:true and warn-if-above:4000) or critical state (critical-if-above:5000). When the server leaves its alerted state, an alert is generated (alertcondition:exit). The minimum amount of time that the threshold can be exceeded before an alert is generated is set to 15 seconds.

```
$ bin/dsconfig create-monitoring-threshold \
  --threshold-name "5k ops/sec" \
  --type static-level \
  --set alert-condition:entry \
  --set alert-condition:exit \
  --set alert-on-warn:true \
  --set min-time-for-trigger:15s \
  --set min-time-for-exit:15s \
  --set warn-if-above:4000 \
  --set critical-if-above:5000
```

4. Create an SLA object that targets an SSO application and monitors the response and throughput times for LDAP bind operations. The response time threshold is set to 15ms.

## Chapter 4: Collecting Data and Metrics

The throughput threshold is set to 5k operations per second. The targeted servers are the set of Proxy Servers, located in Austin.

```
$ bin/dsconfig create-ldap-sla \  
  --sla-name "SSO Application" \  
  --set enabled:true \  
  --set "application-name:SSO Application" \  
  --set "response-time-threshold-ms:15ms response time" \  
  --set "throughput-threshold-ops-per-second:5k ops/sec" \  
  --set ldap-op:bind \  
  --set "sla-server-query:Austin Proxy Servers"
```

---

## Chapter 5: Configuring Charts and Dashboards

---

The Metrics Engine provides a set of dashboards with series of charts for each configured UnboundID server.

Charts can be built and customized with the Metrics Engine Chart Builder tool. Dashboards and charts can be modified with Velocity templates.

Topics include:

[Available Dashboards](#)

[Available Charts for Identity Servers](#)

[The Chart Builder Tool](#)

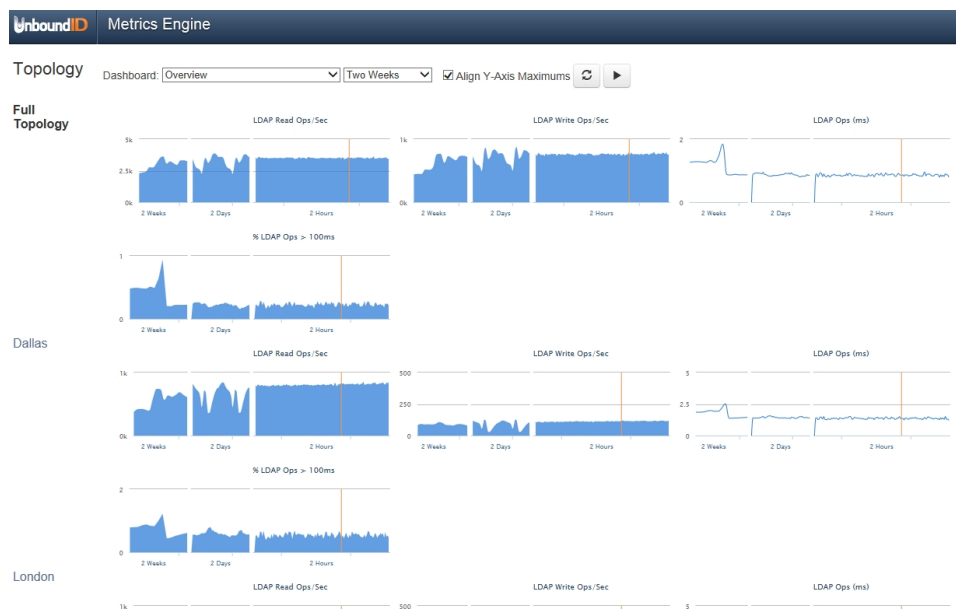
[Velocity Templates](#)

## Available Dashboards

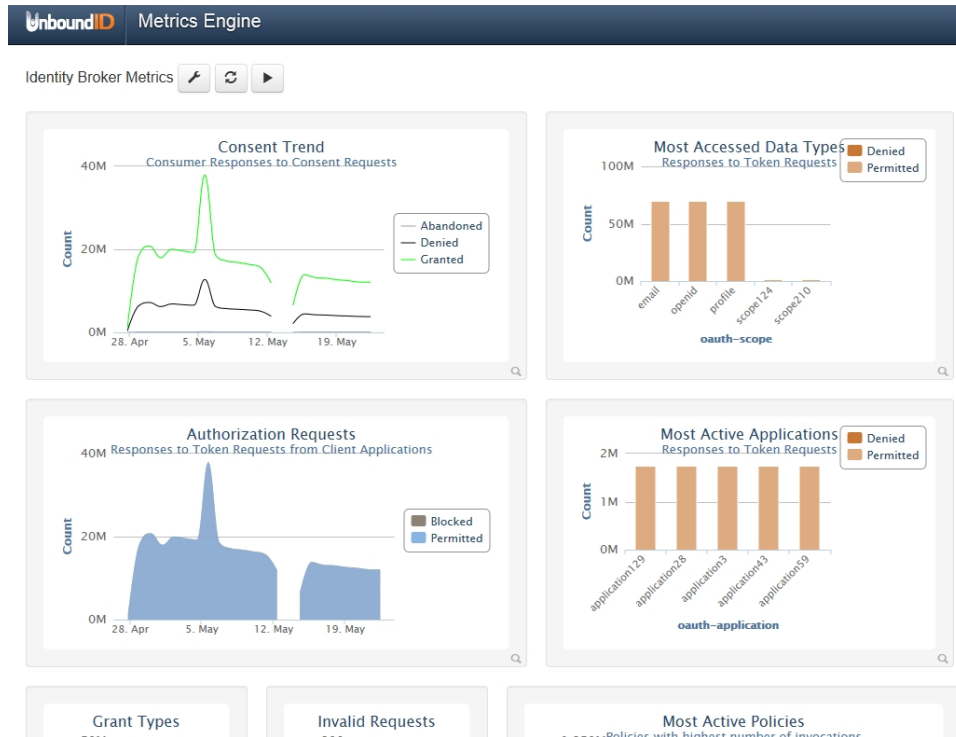
The Metrics Engine includes several dashboards that can be used to display information for all servers in a data center, specific applications, or SLA specifics. The following dashboards are available:

**ldap-dashboard** – Displays charts for Data Store, Proxy, and Data Sync servers configured with the `monitored-servers` command. Charts are also displayed for the Metrics Engine server. This dashboard is viewed from a browser at `http://<metrics-host>:<port>/view/ldap-dashboard`, and is easily customized. See [Customize the LDAP Dashboard](#). The charts can display information by:

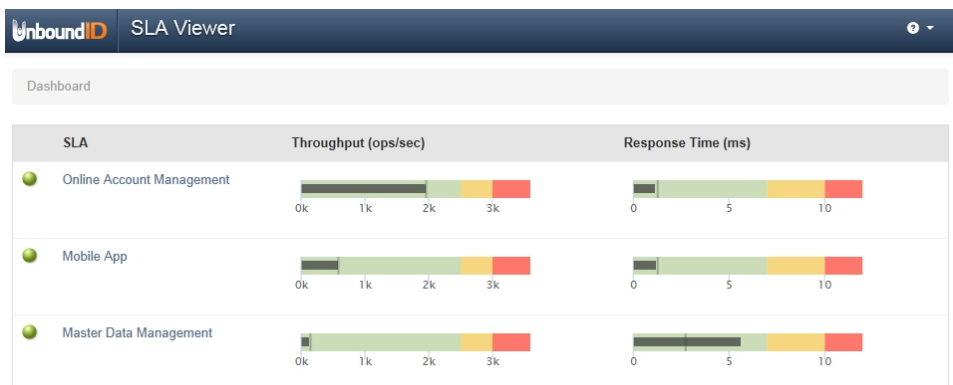
- Individual server, server location, or server type.
- Varying level of detail adjusted by server type.
- Time scale, providing either a recent or more historical data view.



**broker-dashboard** – Displays charts for configured Data Broker servers. Charts include information for data consent, applications requesting access to data, authorization, and policy activity. This dashboard is viewed from a browser at `http://<metrics-host>:<port>/view/broker-dashboard`, and can also be displayed in the Data Broker Console. See [Configure Charts for the Data Broker](#).



**sla-viewer** – Displays throughput and response time graphs, and status for configured SLAs. This dashboard is viewed from a browser at <http://<metrics-host>:<port>/view/sla-viewer>. See [Monitoring for Service Level Agreements](#) for information about configuring SLAs.

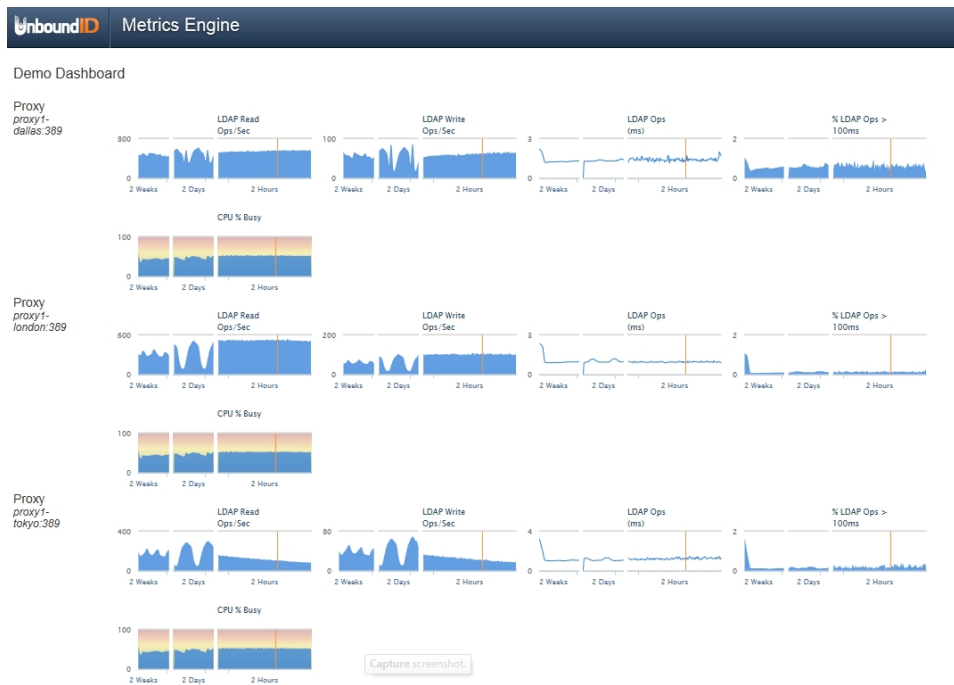


**sla-viewer-details** – Displays SLA Viewer data and additional charts for response time and time ranges from the sla-viewer dashboard. Data can be viewed per server and includes server details.

## Chapter 5: Configuring Charts and Dashboards



**demo-dashboard** – Demonstrates how to display a set of charts for multiple servers and how to vary that set of charts per server type. This dashboard is viewed from a browser at `http://<metrics-host>:<port>/view/demo-dashboard`. This dashboard can be used as a starting point for custom dashboards.





A dashboard readme file provides general instructions for customizing any dashboard, and is located in:

```
<server-root>/config/dashboard/dashboard.README
```

Custom style sheets can be created and referenced in the dashboard template or styles can be configured for all charts. See [Chart Presentation Details](#) for information. The Metrics Engine default style sheets should not be modified.

## Customize the LDAP Dashboard

Dashboards are defined by Velocity templates. After servers are configured, the LDAP dashboard displays all metrics from monitored servers. See [Velocity Templates](#) for information about templates and template components.

Perform the following to configure the LDAP dashboard:

- The configuration file for the LDAP dashboard is `<server-root>/config/velocity/templates/_ldap-dashboard-config.vm`. This file should not be changed, but can be used as a guide for customization.

### Note

Files within this directory that begin with an underscore ( `_` ) are templates that are referenced by each of the dashboards. The `_ldap-dashboard-config.vm` template is the only file that contains all of the dashboard configuration inside the file. Configuration of other templates requires configuration of a corresponding dashboard file as well.

- The `_ldap-dashboard-config.vm` file references a template file that can be customized in `<server-root>/velocity/templates/_ldap-dashboard-config-overrides.vm`. This is the file that should be customized.
- Both files contain configuration instructions. The following can be customized in the LDAP dashboard overrides file:
  - The charts that display for each server type and their styles. See [Available Server Charts](#).
  - The charts that display for a data center and their styles.
  - The charts that display for an application type and their styles.
  - The default time resolution (two weeks, is the default for data displayed).
  - The size of the charts.

## Debug Dashboard Customization

A debug option can be used in any Velocity template for exploring available information in the Velocity Context. This information includes the servers that are monitored and the metrics that are available. This option is included in the `ldap-dashboard` and `demo-dashboard` files:

```
## Uncomment this to have a window popup with detail of what's in the Velocity Context.
```

```
##parse("_debug.vm")
##debug()
```

See [Velocity Templates](#) for more information.

### Preserve Customized Files

Any files that are customized should be copied from the `config/velocity` subdirectories to the same subdirectory of the velocity directory under the server root (`<server-root>/velocity`). The files in `config/velocity` should not be modified. They are updated when the product is updated.

By default, any file of the same name under `<server-root>/velocity` will be loaded in place of `<server-root>/config/velocity`. This enables the preservation of customized files after a product upgrade.

After a product upgrade, review the files in `config/velocity` to determine if any changes should be incorporated into customized templates.

## The Chart Builder Tool

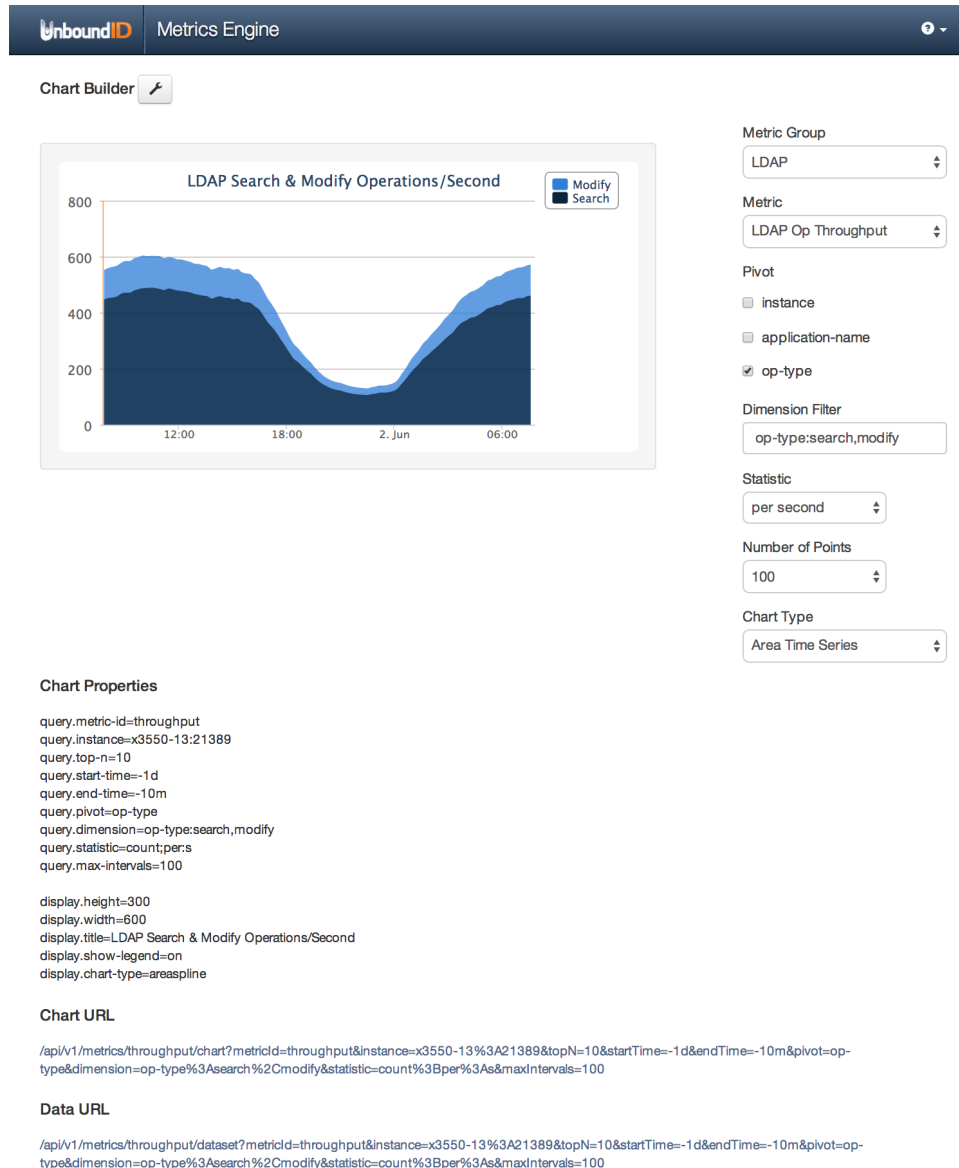
The Chart Builder tool is used to create performance charts for all configured servers. The Broker dashboard can be surfaced on the Data Broker Console Metrics page. See [Configure Charts for Data Broker](#) for more information.

As the settings in the Chart Builder are changed, the builder gathers the data from the Metrics Engine using the Metrics Engine REST API. Once configured, the dashboard page asynchronously fetches metric data for all charts, with each chart rendering when its data is returned. While most metric queries respond quickly (50-100ms), some queries may take longer. If the lag seems too long, consider making changes to the query to reduce the amount of data gathered.

Selecting specific instances and using dimension filters can decrease query time. The Chart Builder tool and the underlying libraries constrain a chart to a single metric. The size of each chart is determined by the library default size (300x300) and can be overridden in the chart properties file. There are times when the legends and labeling of a chart dictate the minimum size for a chart.

The Chart Builder tool ( `chart-builder.vm` ) is shipped with the Metrics Engine and is enabled after installation at the following URL:

```
https://<metrics-engine-host>:<port>/view/chart-builder
```



The metrics parameters used to build the chart can be saved to a properties file and added to a dashboard. If not using the Chart Builder tool, the `_chart-definition.template` file in `<server-root>/config/dashboard/charts` provides instructions about manually creating charts and adding them to a dashboard.

## Chart Presentation Details

Chart presentation details can be configured per chart or for all charts in the `_chart-definition.template` file. A properties file can be created for common styles and referenced in this file. Instructions for adding custom styles are included in the file.

The following is a sample of the chart parameters that are available:

- Colors used in the data series.
- Enable and disable a legend.
- Location (top/bottom/left/right) of the legend.
- Background color.
- Thickness of the time-series lines (absolute or as a function of the # of plotted series).
- Macro expansion in the specified title.
- Sub-title (with macro expansion that includes metric-name and current date/time).

### Chart Builder Parameters

Use the Chart Builder tool to build or adjust system and performance charts. When the configuration is set, copy the parameters into a properties file, and add the chart to a dashboard.

**Chart Builder Parameters**

Parameters	Description
Metric Group	Selects a specific group of metrics to be considered for charting.
Metric	Displays the specific metric. Open the drop-down list and hover over a metric to view a description of the particular metric.
Pivot	Splits the chart result into multiple series based on the pivot dimension chosen.
Dimension Filter	Filters the data based on the dimension(s) entered, such as the type of operations that can be viewed for an LDAP operations metric.
Statistic	<p>Displays the type of "measurement" that may exist for each metric. For example, each response-time sample contains:</p> <ul style="list-style-type: none"> <li>• number of operations (count)</li> <li>• average time-per-op (average)</li> <li>• histogram-of-operation-time (histogram)</li> </ul> <p>On a per-sample basis, the Metrics Engine stores the following: count, average, minimum, maximum, and histogram. Any metric can have one or more of five statistics, but not all statistics are equally valuable. Note the following points:</p> <ul style="list-style-type: none"> <li>• The minimum and maximum statistics may be of limited value, because as they are time-averaged, they go to extremes (min of minimums and max of maximums).</li> <li>• The count and histogram statistics have high fidelity over time because they time-aggregate perfectly.</li> <li>• The average statistic loses fidelity over time, because as the time-window for averaging gets larger, the highs and lows are lost.</li> </ul>
Number of Points	If the number of points is set to 1, all chart types, except time series, may be used. If the number of points is > 1, then only time series charts may be used.
Chart Type	Displays the chart based on the type:

### Chart Builder Parameters

Parameters	Description
	<ul style="list-style-type: none"> <li>• Area Time Series</li> <li>• Bar Chart</li> <li>• Column Chart</li> <li>• Pie Chart</li> <li>• Stacked Bar Chart</li> <li>• Stacked Column Chart</li> <li>• Time Series</li> </ul>
Chart Properties	Displays the generated chart properties. Copy the query properties into a properties file.
Chart URL	The URL to display a static image of the chart. This can be used to call the chart into a third-party client application.
Data URL	The API for getting the data. This can be used to call the chart into a third-party client application.

## Chart Properties File

Each dashboard uses a Velocity template (`<name>.vm`) and a set of chart properties files to render the charts. As charts are configured with the Chart Builder tool, the tool generates the corresponding properties for each customized item. The metrics configuration can be copied into a properties file and added to a dashboard. If no values are specified for a given property, the property will use a default value from the `<server-root>/config/dashboard/charts/_chart-definition.template` file. All properties and their descriptions are listed in this file.

The properties in the chart definition file are broken into two groups: properties that start with `display` affect the display of the data, and properties that start with `query` affect the metric query. When building a new chart, just copy the query parameters into a properties file. In general, display options should be referenced from common styles defined in the `_chart-definition.template` file, or the styles defined for a dashboard.

## Available Charts for Identity Servers

The following are the default charts that display on the LDAP Dashboard for each configured server. These and additional charts for server and system metrics reside in `<server-root>/config/dashboard/charts`. They can be modified or used to create new charts.

### Charts for All Servers

The following charts are displayed on the LDAP Dashboard for Data Store and Proxy servers:

- LDAP Read Operations Per Second
- LDAP Write Operations Per Second
- System CPU
- System Memory Percent Free

## Chapter 5: Configuring Charts and Dashboards

- LDAP Response Time
- LDAP Response Time Outliers
- LDAP Worker Thread Percent Busy
- LDAP Average Operations in Progress
- LDAP Average Queue Size
- LDAP Open Connections
- LDAP New Connections
- System Network Read MB
- System Network Write MB
- System Disk Busy
- System Disk Service Wait
- System Disk Read MB
- System Disk Write MB

### Data Store Charts

The following charts are displayed on the dashboard for the Data Store:

- Replication Backlog
- Replication Oldest Change
- Replication Unresolved Naming Conflicts
- Backend Entry Count
- Backend Cache Percent Full
- Backend Size on Disk
- Backend Cleaner Backlog

### Proxy Server Charts

The following charts are displayed on the dashboard for the Proxy Server:

- External Server Total Operations
- External Server Failed Operations
- External Server Health

### Data Sync Server Charts

The following charts are displayed on the dashboard for the Data Sync Server:

- Sync Pipe Unretrieved Changes
- Sync Pipe Percent Busy
- Sync Pipe Completed Operations Success
- Sync Pipe Completed Operations Failed
- Sync Pipe Completed Operations By Type

### Metrics Engine Server Charts

The following charts are displayed on the dashboard for the Metrics Engine Server:

- Metrics Queries Per Minute
- Metrics Query Time
- Metrics Query Time Max
- Metrics Query Time Histogram
- Metrics Cache Miss Count
- Metrics Cache Expired Count
- Metrics Cache Evicted Count
- Metrics Import Delay

- Metrics Cache Entry Count
- Metrics Cache Hit Count
- Metrics Load Time
- Metrics DBMS Cluster Time

## Data Broker Charts

The following charts are displayed on the dashboard for the Data Broker Server:

- Consent Trend
- Invalid Requests
- Authorization Requests
- Oauth Grant Types
- Most Accessed Data Types
- Most Active Applications

## Velocity Templates

The Metrics Engine exposes Velocity pages through an HTTP Servlet Extension. If the HTTP Connection Handler is enabled, the Velocity extension is enabled.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "HTTPS Connection Handler" \
  --add http-servlet-extension:Velocity
```

Velocity template files contain presentation content and variables that are replaced when the content is requested. Variables are expressed using a `$` followed by an identifier that refers to an object put into a context (VelocityContext) by the server.

Velocity extensions can be configured to expose a number of objects in the context using the `expose-*` properties:

- **expose-request-attributes** – Indicates whether HTTP request attributes are accessible to templates using the `$subid_request` variable. In general, request attributes are added by server components processing the HTTP request. Also the HTTP request parameters map is available as `$subid_request.parameters`. Request parameters are supplied by the requester, usually in the request URL query string or in the body of the request itself.
- **expose-session-attributes** – Indicates whether HTTP session attributes are accessible to templates using the `$subid_session` variable. Like request attributes, session attributes are also added by server components processing the HTTP request. The lifetime of these attributes persists until the user's session has ended.
- **expose-server-context** – Indicates whether a Server SDK server context is accessible to templates using the `$subid_server` variable. The server context provides access to properties and additional information about the server. See the *Unbound ID Server SDK* documentation for more details.

The following are other properties of the Velocity HTTP Servlet Extension:

- **description** – A description of the extension.
- **cross-origin-policy** – Defines a cross origin policy for this extension.

- **base-context-path** – URL base context for the Velocity Servlet.
- **static-content-directory** – In addition to templates, the Velocity Servlet will serve miscellaneous static content related to the templates. By default this is `config/velocity/statics`.
- **require-authentication** – Requires credentials to access Velocity content.
- **identity-mapper** – Maps user credentials to backend entries. If the `require-authentication` property is set, use this property to map bind credentials from a users backend. This is set to `Exact Match` by default. Metrics Engine Velocity template authentication should share the `api-users` LDIF backend used by the REST API. Details are available in the Metrics Engine REST API servlet configuration, and in the [Connection and Security](#) section of the Metrics Engine REST API Reference chapter.
- **static-custom-directory** – If static content is customized, it resides in `velocity/statics` by default.
- **template-directory** – The template directory from which templates are read. By default this is `config/velocity/templates`. This directory also serves as a default for Template Loaders that do not have a template directory specified.
- **static-context-path** – URL path beneath the base context where static content can be accessed.
- **allow-context-override** – Indicates whether context providers may override existing context objects with new values.
- **mime-types-file** – Specifies a file that is used to map file extensions of static content to a Content Type to be returned with requests.
- **default-mime-type** – The default Content Type for HTTP responses. Additional content types are supported by defining one or more additional Velocity Template Loaders.

The VelocityContext object can be further customized by configuring additional Velocity context providers. The dot notation used for context references can be extended to access properties and methods of objects in context using Java Bean semantics. For example, if the HTTP request URL includes a `name` query string parameter like:

```
http://example.com:8080/view/hello?name=Joe
```

An HTML template like the following could be used to generate a page containing a friendly greeting to the requestor:

```
<html>
  <body>
    Hello $ubid_request.parameters.name
  </body>
</html>
```

A pop-up window displays a table on the page that lists all variables that are in the Velocity Context. References like `$ubid_request` can appear in the template file and be replaced when



the template is rendered. This information can be used to check which variables are permitted to be in the template along with the variable values.

#### Note

For security, all template substitutions are HTML escaped by default. To substitute unescaped content, a variable name ending with "WithHtml" must be used. For example, `$addressWithHtml`, would substitute the contents of the `$addressWithHtml` variable into the page generated from the HTML template without escaping it.

A debug option can be used in any Velocity template for verifying available information in the Velocity Context:

```
parse ("_debug.vm")
debug ()
```

If a variable is added to a template for something that does not exist, the rendered page will contain a literal string of the unfulfilled variable (for example `$undefined_variable`).

By default, the Velocity Servlet Extension expects to access content in subdirectories of the server's `config/velocity` directory:

- **templates** – This directory contains Velocity template files that are used to generate pages in response to client requests.
- **statics** – This directory contains static content such as cascading style sheets, HTML, and Javascript files as well as images and third-party libraries.

## Supporting Multiple Content Types

By default, the Velocity Servlet Extension is configured to respond to HTTP requests with a content type `text/html`. Change this request type by setting the default MIME type using `dsconfig`. For example, the following can be used to set the default type to XML:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name Velocity \
  --set default-mime-type:application/xml
```

HTML requests can be supported as well as clients that seek content in other formats. Create one or more Velocity template loaders to load templates for other content types like XML or JSON.

The ability to serve multiple formats of a document to clients at the same URL is typically called *content negotiation*. HTTP clients indicate the type of content desired using the `Accept` header. A client may use a header like the following to indicate that they prefer content in XML but will fallback to HTML if necessary:

```
Accept: application/xml,text/html;q=0.9
```

The following can be used to create a Velocity template loader for XML content:

```
$ bin/dsconfig create-velocity-template-loader \
  --extension-name Velocity \
  --loader-name XML \
  --set evaluation-order-index:502 \
```

```
--set mime-type-matcher:application/xml \  
--set mime-type:application/xml \  
--set template-suffix:.vm.xml
```

Upon receiving a request, the Velocity Servlet first creates an ordered list of requested media types from most desired to least based on the value of the `Accept` header. Starting from the most desired type, it will then iterate over the defined template loaders according to the `evaluation-order-index` property from lowest value to highest.

A template loader can indicate that it can handle content for requested media type by comparing the requested type to its `mime-type-matcher` property. A loader can be configured to load templates from a specific directory or load template files having a particular suffix. For example, XML templates are expected to be named using a `.vm.xml` suffix. If a loader indicates it handles the requested content type and a template exists for the requested view, the template is loaded and used to generate a response to the client. If no loaders are found for the requested media type, the next most preferred media type (if any) is tried. If no loaders indicated that they could satisfy the requested view, the client is sent an `HTTP 404 (not found)` error. If no loaders could provide acceptable media but the requested view exists in some other format, the client is sent an `HTTP 406 (not acceptable)` error.

In this example, a template file called `hello.vm.xml` can be used to generate a response in XML:

```
<hello name="$ubid_request.parameters.name"/>
```

In this case, the response will contain an HTTP Content-Type header with the value of the `mime-type` property of the Velocity template loader.

## Velocity Context Providers

The previous examples use a value supplied as an HTTP request query string parameter to form a response. The templates contain a variable `$ubid_request.parameters.name` that was replaced at runtime with a value from the Velocity Context.

The Velocity Extension can be configured to make some information available in the Velocity Context such as the HTTP request, session, and Server SDK Server Context. Velocity Context Providers provide more flexibility in populating the Velocity Context for template use.

Here are some of the properties of a Velocity Context Provider:

- **enabled** – Indicates whether the provider will contribute content for any requests.
- **object-scope** – Indicates to the provider how often objects contributed to the Velocity Context should be re-initialized. Possible values are: `request`, `session`, or `application`.
- **included-view/excluded-view** – These properties can be used to restrict the views for which a provider contributes content. A view name is the request URL's path to the resource without the Velocity Servlet's context or a leading forward slash. If one or more views are included, the provider will service requests for just the specified views. If one

or more views are excluded, the provider will service requests for all but the excluded views.

## Velocity Tools Context Provider

Apache's Velocity Tools project is focused on providing utility classes useful in template development. The Velocity Context can be configured by specifying Velocity Tool classes to be automatically added to the Velocity Context for template development. For more information about the Velocity Tools project, see the Velocity website.

The following command can be used to list the set of Velocity Tools that are included in the Velocity Context for general use by templates:

```
$ bin/dsconfig get-velocity-context-provider-prop \  
--extension-name Velocity \  
--provider-name "Velocity Tools" \  
--property request-tool \  
--property session-tool \  
--property application-tool \  

```

---

## Chapter 6: Troubleshooting

---

There are several ways to troubleshoot issues with data gathering or with the Metrics Engine itself.

Topics include:

[Collect Support Data Tool](#)

[Enable JVM Debugging](#)

[Delays in Sample Data Availability](#)

[Slow Queries Based on Sample Cache Size](#)

[Performance Troubleshooting Example](#)

[Insufficient Memory Errors](#)

[Unexpected Query Results](#)

[Installation and Maintenance Issues](#)

## Using the collect-support-data Tool

UnboundID servers provide information about their current state and any problems encountered. If a problem occurs, run the `collect-support-data` tool in the `/bin` directory. The tool aggregates all relevant support files into a zip file that can be sent to a support provider for analysis. The tool also runs data collector utilities, such as `jps`, `jstack`, and `jstat` plus other diagnostic tools for the operating system.

The tool may only archive portions of certain log files to conserve space, so that the resulting support archive does not exceed the typical size limits associated with e-mail attachments.

The data collected by the `collect-support-data` tool may vary between systems. The data collected includes the configuration directory, summaries and snippets from the `logs` directory, an LDIF of the monitor and RootDSE entries, and a list of all files in the server root.

Perform the following steps to run this tool:

1. Navigate to the server root directory.
2. Run the `collect-support-data` tool. Include the host, port number, bind DN, and bind password.

```
$ bin/collect-support-data \  
--hostname 100.0.0.1 --port 389 \  
--bindDN "cn=Directory Manager" \  
--bindPassword secret \  
--serverRoot /opt/UnboundID-<server> \  
--pid 1234
```

3. Email the zip file to a support provider.

## Delays in Sample Data Availability

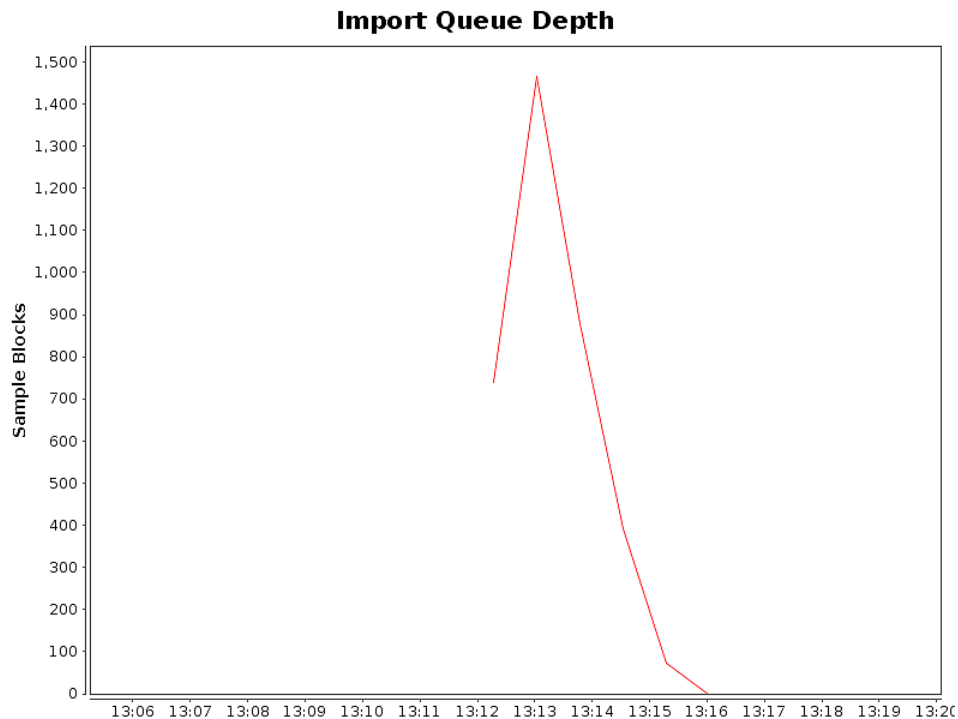
The time between when a metric sample is captured and when it is available in the Metrics Engine is a combination of queuing and polling delays. The default configuration allows the monitored server to queue samples in memory for up to 30 seconds before writing them to disk.

The Metrics Engine polls each monitored server every 30 seconds by default. In a worst case, a sample may have been captured on the monitored server 60 seconds before it has been captured and queued for import on the Metrics Engine. When all servers are running normally, 60 seconds is the upper limit of a normal delay between when a sample is captured on the monitored server and when it is available to a query on the Metrics Engine.

Use the following URL in a browser to chart the number of sample blocks queued by the Metrics Engine as a function of time over the past hour. Estimate, using the downward slope of the spike, how long it will take to clear the backlog.

```
http://<metrics-engine-host:port>/api/v1/metrics/monitor-import-queue-  
depth/chart?maxIntervals=60&startTime=-1h
```

Below is a sample from a Metrics Engine that was shut down for 10 minutes. The spike that occurs on startup results from the fact that all monitored servers continued to queue sample blocks. When the Metrics Engine restarted, it fetched them and queued them for import. About 1500 sample blocks were queued and it took the Metrics Engine about three minutes to catch up.



To monitor over LDAP, the following LDAP entry contains the equivalent information:

```
dn: cn=Aggregation,cn=monitor
  Attribute: import-queue - number of sample blocks waiting for import
  (should be close to zero)
  Attribute: import-load-delay-millis - milliseconds between when the sample
  block arrived and when it was imported (should be less than 5 seconds)
  Attribute: import-load-millis - milliseconds to load the block to DBMS
  (should be less than 50 milliseconds)
  Attribute: import-parse-millis - milliseconds to parse the block to a
  normalized form ready for import (should be less than 75 milliseconds)
```

## Slow Queries Based on Sample Cache Size

The evicted-count attribute of the sample cache sets the number of entries that have been evicted from the cache due to a lack of space. The cache may not be large enough for the query load placed on the server. Increase the size of the sample cache with the following command, which sets the maximum size to 200000:

```
$ dsconfig set-monitoring-configuration-prop \
  --set sample-cache-max-cached-series:200000
```

## Chapter 6: Troubleshooting

Some queries are so infrequent that the cached data expires due to age. The default age is 10 minutes, but this can be increased up to one hour. If the `expired-count` monitor attribute is increasing between queries, consider increasing the idle timeout as follows:

```
$ dsconfig set-monitoring-configuration-prop \
  --set sample-cache-idle-series-timeout:20m
```

## Performance Troubleshooting Example

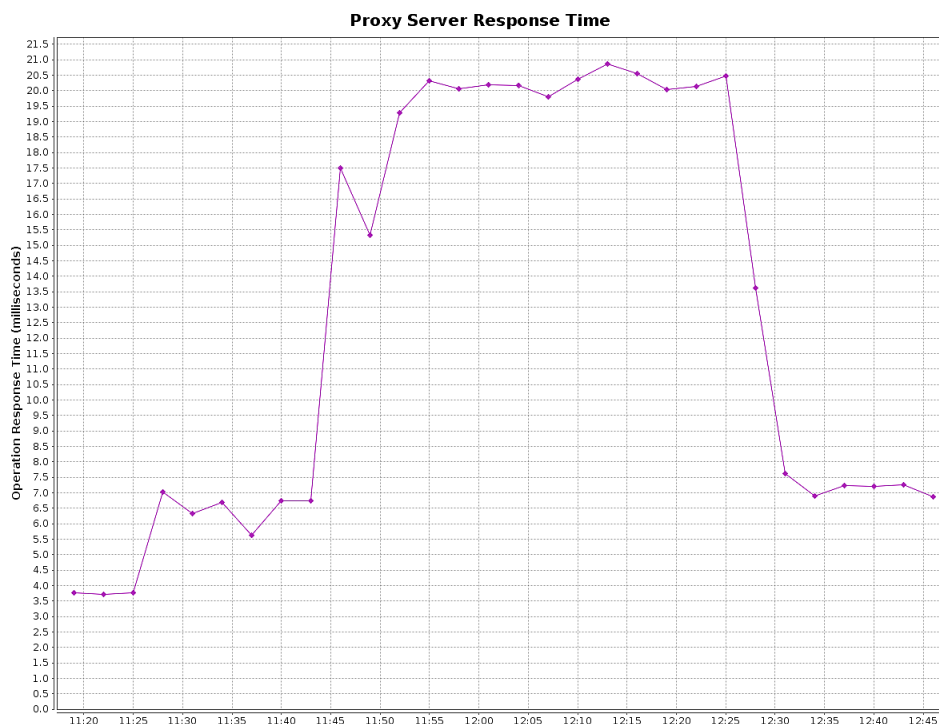
The Metrics Engine monitors itself at the same time it monitors other servers, so the historical view of the status and performance of the Metrics Engine is captured in the DBMS and is available for historical analysis.

The following example answers the question of why an application, which was performing well 30 minutes ago, now exhibits performance issues. The application in question is hosted on a pair of Proxy Servers with both servers sharing the same pair of Identity Data Stores in a round-robin configuration. All of the charts are generated with the `query-metric` tool.

A plot of the average Proxy Server response time that covers the time frame of the issue is captured using the following `query-metric` command:

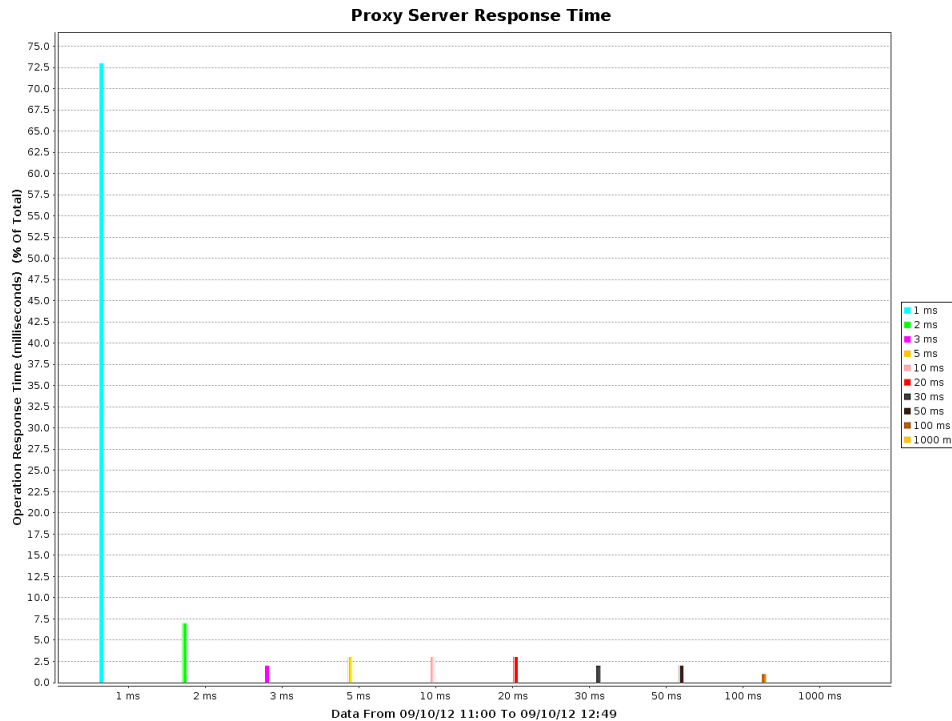
```
$ bin/query-metric query \
  --metric response-time \
  --instanceType proxy \
  --startTime -1h
```

The command displays the following chart.



## Performance Troubleshooting Example

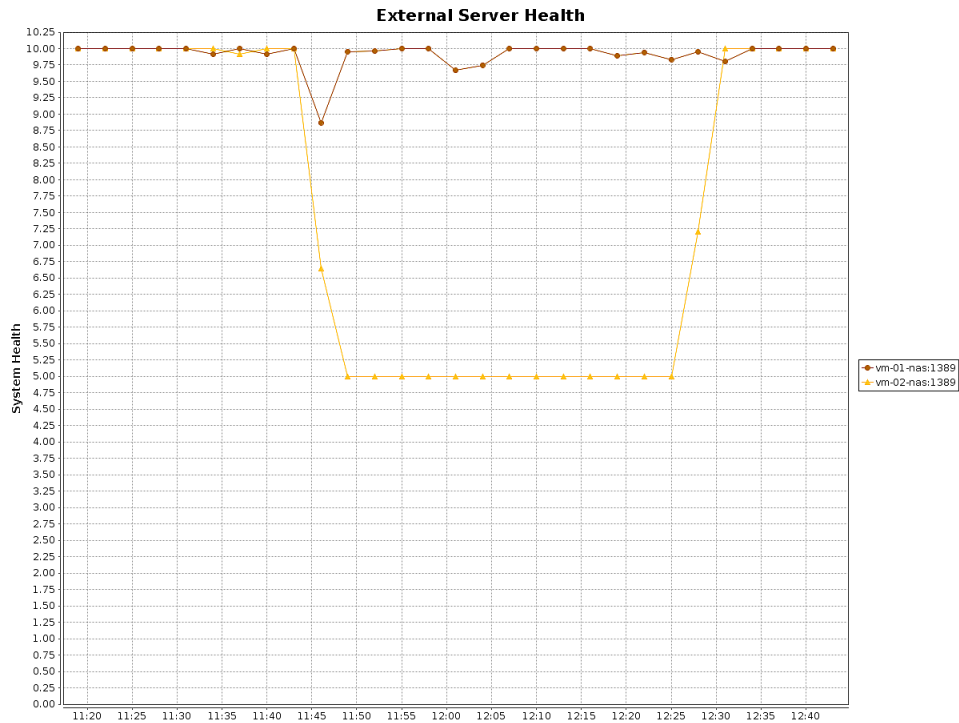
This chart shows the application response time tripled. This could mean that either a few requests took a long time, or everything slowed down. To get more information, use the `query-metric` command to get a plot of the application response time histogram over the same time. The result is the following graph.



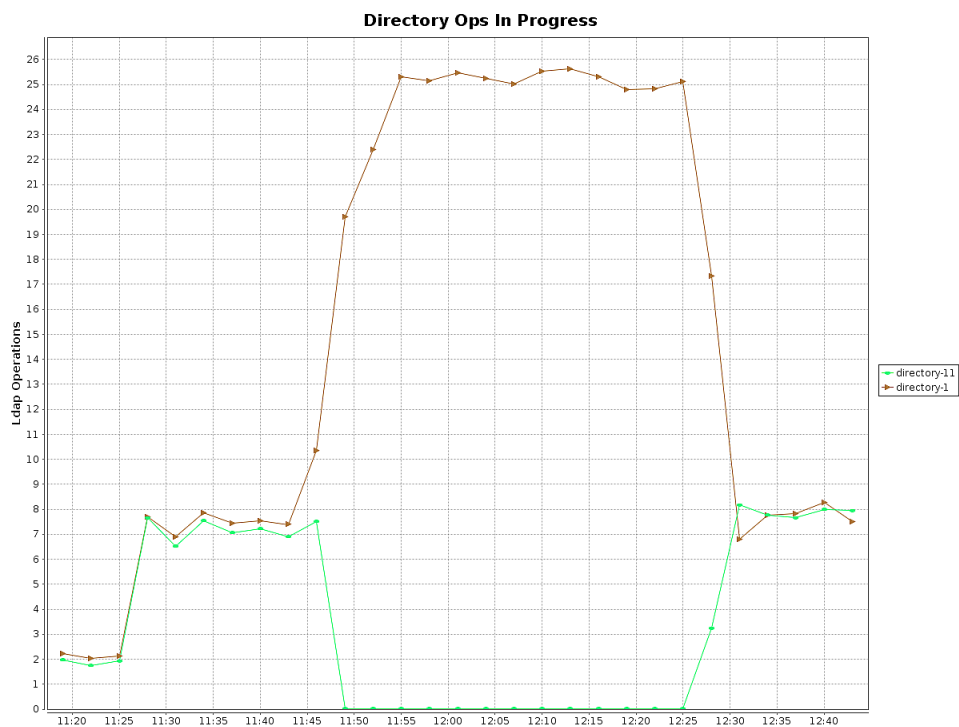
This graph shows that no requests during the increased response time period took a long time. It appears that all operations were slow. The next step might be to look at the external server health.



## Chapter 6: Troubleshooting



This chart displays an increase in response time that matches the decrease in external server health on vm-02-nas:1389. The problem appears to be on that specific Data Store. The next step is to determine what each Data Store was doing.



The chart shows that directory-11 (vm-02-nas:1389) stopped responding then corrected 30 minutes later. Finally, the most recent status for directory-11 is retrieved using the `status` command. The following is displayed:

```
--- Administrative Alerts ---
Severity : Time : Message
-----:-----:-----
-----
-----
-----
Error : 10/Sep/2015 11:47:39 -0500 : A severe backlog has been detected in the
Data Store work queue. The operation currently at the head of the queue has
been waiting for 25785 milliseconds
Error : 10/Sep/2015 11:47:25 -0500 : A severe backlog has been detected in the
Data Store work queue. The operation currently at the head of the queue has
been waiting for 11790 milliseconds
Warning : 10/Sep/2015 11:47:12 -0500 : The Data Store has detected that the
amount of usable disk space is below the configured low disk space warning
threshold for the following path(s):
: : '/home/slj/deploy/ds2' (totalBytes: 18624344064, usableBytes: 1851559936,
usablePercent: 10),
'/home/slj/deploy/ds2/changelogDb' (totalBytes: 18624344064,
: : usableBytes: 1851559936, usablePercent:
10), '/home/slj/deploy/ds2/config' (totalBytes: 18624344064, usableBytes:
1851559936, usablePercent: 10),
: : '/home/slj/deploy/ds2/db/changelog' (totalBytes: 18624344064, usableBytes:
1851559936, usablePercent: 10),
'/home/slj/deploy/ds2/db/userRoot' (totalBytes:
: : 18624344064, usableBytes: 1851559936, usablePercent: 10),
'/home/slj/deploy/ds2/logs' (totalBytes: 18624344064, usableBytes: 1851559936,
usablePercent: 10)
```

Looking at the charts and server status above, the available disk space on directory-11 went below the warning threshold for a period, resulting in the traffic shifting from two identity data stores to only one for about 30 minutes. At the end of that time, both Identity Data Stores resumed normal operations and the response time returned to normal.

## Insufficient Memory Errors

If the server shuts down due to insufficient memory errors, it is possible that the allocated heap size is not enough for the amount of data being returned. Consider increasing the heap size, or reducing the number of request handler threads using the following `dsconfig` command:

```
$ bin/dsconfig set-connection-handler-prop \
--handler-name "HTTP Connection Handler" \
--set num-request-handlers:<num-of-threads>
```

## Unexpected Query Results

The query API aggregates data samples across servers and dimension values. The samples for different servers, or even different dimension values, are imported into the Metrics Engine at different times. All metric data is imported in time-order for each server. The ordering cannot be set across servers, and samples for a specific time may arrive in stages. Therefore, a metric query that aggregates across servers or dimensions may get partial data when the query time range ends. This problem can be compounded when the monitored servers clocks are not synchronized (samples have the monitored server timestamp). The query looks at a single time range. The more [clock skew](#) between the monitored servers, the higher the probability of the results not being accurate for the range.

With the query API, the data can be pivoted (split) by server and dimension. The API enables formatting the results as an HTML table. The following sequence of API URLs return the last three minutes of data in 10-second increments:

```
http://<metrics-engine-host:port>/api/v1/metrics/throughput/datatable?
maxIntervals=30&startTime=-3m&tqx=out:html&tz=US/Central

http://<metrics-engine-host:port>/api/v1/metrics/throughput/datatable?
maxIntervals=30&startTime=-3m&tqx=out:html&tz=US/Central&pivot=instance

http://<metrics-engine-host:port>/api/v1/metrics/throughput/datatable?
maxIntervals=30&startTime=-
3m&tqx=out:html&tz=US/Central&pivot=instance&pivot=op-type
```

- The first URL aggregates all servers and LDAP operations into a single number split across time.
- The second URL splits out the data by server and time.
- The third URL splits out the data by server, LDAP operation, and time.

As dimension pivots (splits) are added, the results display more aggregations of partial data.

## Installation and Maintenance Issues

The following are common installation and maintenance issues and possible solutions.

### The setup Program will not Run

If the `setup` tool does not run properly, some of the most common reasons include the following:

**A Java Environment Is Not Available** – The server requires that Java be installed on the system prior to running the `setup` tool.

If there are multiple instances of Java on the server, run the `setup` tool with an explicitly-defined value for the `JAVA_HOME` environment variable that specifies the path to the Java installation. For example:

```
$ env JAVA_HOME=/ds/java ./setup
```

Another issue may be that the value specified in the provided `JAVA_HOME` environment variable can be overridden by another environment variable. If that occurs, use the following command to override any other environment variables:

```
$ env UNBOUNDID_JAVA_HOME="/ds/java" UNBOUNDID_JAVA_BIN="" ./setup
```

**Unexpected Arguments Provided to the JVM** – If the `setup` tool attempts to launch the `java` command with an invalid set of arguments, it may prevent the JVM from starting. By default, no special options are provided to the JVM when running `setup`, but this might not be the case if either the `JAVA_ARGS` or `UNBOUNDID_JAVA_ARGS` environment variable is set. If the `setup` tool displays an error message that indicates that the Java environment could not be started with the provided set of arguments, run the following command:

```
$ unset JAVA_ARGS UNBOUNDID_JAVA_ARGS
```

**The Server Has Already Been Configured or Started** – The `setup` tool is only intended to provide the initial configuration for the server. It will not run if it detects that it has already been run.

A previous installation should be removed before installing a new one. However, if there is nothing of value in the existing installation, the following steps can be used to run the `setup` program:

- Remove the `config/config.ldif` file and replace it with the `config/update/config.ldif.{revision}` file containing the initial configuration.
- If there are any files or subdirectories in the `db` directory, then remove them.
- If a `config/java.properties` file exists, then remove it.
- If a `lib/setup-java-home` script (or `lib\set-java-home.bat` file on Microsoft Windows) exists, then remove it.

## The Server will not Start

If the server does not start, then there are a number of potential causes.

**The Server or Other Administrative Tool Is Already Running** – Only a single instance of the server can run at any time from the same installation root. Other administrative operations can prevent the server from being started. In such cases, the attempt to start the server should fail with a message like:

```
The <server> could not acquire an exclusive lock on file
/ds/UnboundID-<server>/locks/server.lock:
The exclusive lock requested for file
/ds/UnboundID-<server>/locks/ server.lock
was not granted, which indicates that another
process already holds a shared or exclusive lock on
that file. This generally means that another instance
of this server is already running.
```

If the server is not running (and is not in the process of starting up or shutting down), and there are no other tools running that could prevent the server from being started, it is possible that a previously-held lock was not properly released. Try removing all of the files in the locks directory before attempting to start the server.

**There Is Not Enough Memory Available** – When the server is started, the JVM attempts to allocate all memory that it has been configured to use. If there is not enough free memory available on the system, the server generates an error message indicating that it could not be started.

There are a number of potential causes for this:

- If the amount of memory in the underlying system has changed, the server might need to be re-configured to use a smaller amount of memory.
- Another process on the system is consuming memory and there is not enough memory to start the server. Either terminate the other process, or reconfigure the server to use a smaller amount of memory.
- The server just shut down and an attempt was made to immediately restart it. If the server is configured to use a significant amount of memory, it can take a few seconds for all of the memory to be released back to the operating system. Run the `vmstat` command and wait until the amount of free memory stops growing before restarting the server.
- For Solaris-based systems, if the system has one or more ZFS filesystems (even if the server itself is not installed on a ZFS filesystem), it is possible that ZFS caching is holding onto a significant amount of memory and cannot release it quickly enough to start the server. Re-configure the system to limit the amount of memory that ZFS is allowed to use.
- If the system is configured with one or more memory-backed filesystems (such as `tmpfs` used for Solaris `/tmp`), determine if any large files are consuming a significant amount of memory. If so, remove them or relocate them to a disk-based filesystem.

**An Invalid Java Environment or JVM Option Was Used** – If an attempt to start the server fails with 'no valid Java environment could be found,' or 'the Java environment could not be started,' and memory is not the cause, other causes may include the following:

- The Java installation that was previously used to run the server no longer exists. Update the `config/java.properties` file to reference the new Java installation and run the `bin/dsjavaproperties` command to apply that change.
- The Java installation has been updated, and one or more of the options that had worked with the previous Java version no longer work. Re-configure the server to use the previous Java version, and investigate which options should be used with the new installation.

- If an `UNBOUNDID_JAVA_HOME` or `UNBOUNDID_JAVA_BIN` environment variable is set, its value may override the path to the Java installation used to run the server (defined in the `config/java.properties` file). Similarly, if an `UNBOUNDID_JAVA_ARGS` environment variable is set, then its value might override the arguments provided to the JVM. If this is the case, explicitly unset the `UNBOUNDID_JAVA_HOME`, `UNBOUNDID_JAVA_BIN`, and `UNBOUNDID_JAVA_ARGS` environment variables before starting the server.

Any time the `config/java.properties` file is updated, the `bin/dsjavaproperties` tool must be run to apply the new configuration. If a problem with the previous Java configuration prevents the `bin/dsjavaproperties` tool from running properly, remove the `lib/set-java-home` script (or `lib\set-java-home.bat` file on Microsoft Windows) and invoke the `bin/dsjavaproperties` tool with an explicitly-defined path to the Java environment, such as:

```
$ env UNBOUNDID_JAVA_HOME=/ds/java bin/dsjavaproperties
```

**An Invalid Command-Line Option was Used** – There are a small number of arguments that can be provided when running the `bin/start-ds` command. If arguments were provided and are not valid, the server displays an error message. Correct or remove the invalid argument and try to start the server again.

**The Server Has an Invalid Configuration** – If a change is made to the server configuration using `dsconfig` or the Web Console, the server will validate the change before applying it. However, it is possible that a configuration change can appear to be valid, but does not work as expected when the server is restarted.

In most cases, the server displays (and writes to the error log) a message that explains the problem. If the message does not provide enough information to identify the problem, the `logs/config-audit.log` file provides recent configuration changes, or the `config/archived-configs` directory contains configuration changes not made through a supported configuration interface. The server can be started with the last valid configuration using the `--useLastKnownGoodConfig` option:

```
$ bin/start-<ds> --useLastKnownGoodConfig
```

To determine the set of configuration changes made to the server since the installation, use the `config-diff` tool with the arguments `--sourceLocal --targetLocal --sourceBaseline`. The `dsconfig --offline` command can be used to make configuration changes.

**Proper Permissions are Missing** – The server should only be started by the user or role used to initially install the server. However, if the server was initially installed as a non-root user and then started by the root account, the server can no longer be started as a non-root user. Any new files that are created are owned by root.

If the user account used to run the server needs to change, change ownership of all files in the installation to that new user. For example, if the server should be run as the "ds" user in the "other" group, run the following command as root:

```
$ chown -R ds:other /ds/UnboundID-<server>
```

## The Server has Shutdown

Check the current server state by using the `bin/server-state` command. If the server was previously running but is no longer active, potential reasons may include:

- Shut down by an administrator – Unless the server was forcefully terminated, then messages are written to the error and server logs stating the reason.
- Shut down when the underlying system crashed or was rebooted – Run the `uptime` command on the underlying system to determine what was recently started or stopped.
- Process terminated by the underlying operating system – If this happens, a message is written to the system error log.
- Shut down in response to a serious problem – This can occur if the server has detected that the amount of usable disk space is critically low, or if errors have been encountered during processing that left the server without worker threads. Messages are written to the error and server logs (if disk space is available).
- JVM has crashed – If this happens, then the JVM should provide a fatal error log (a `hs_err_pid<processID>.log` file), and potentially a core file.

## The Server will not Accept Client Connections

Check the current server state by using the `bin/server-state` command. If the server does not appear to be accepting connections from clients, reasons can include the following:

- The server is not running.
- The underlying system on which the server is installed is not running.
- The server is running, but is not reachable as a result of a network or firewall configuration problem. If that is the case, connection attempts should time out rather than be rejected.
- If the server is configured to allow secure communication through SSL or StartTLS, a problem with the key manager and/or trust manager configuration can cause connection rejections. Messages are written to the server access log for each failed connection attempt.
- The server may have reached its maximum number of allowed connections. Messages should be written to the server access log for each rejected connection attempt.
- If the server is configured to restrict access based on the address of the client, messages should be written to the server access log for each rejected connection attempt.
- If a connection handler encounters a significant error, it can stop listening for new requests. A message should be written to the server error log with information about the problem. Restarting the server can also solve the issue. Another option is to create an LDIF file that disables and then re-enables the connection handler, create the

`config/auto-process-ldif` directory if it does not already exist, and then copy the LDIF file into it.

## The Server is Unresponsive

Check the current server state by using the `bin/server-state` command. If the server process is running and appears to be accepting connections but does not respond to requests received on those connections, potential reasons for this include:

- If all worker threads are busy processing other client requests, new requests are forced to wait until a worker thread becomes available. A stack trace can be obtained using the `jstack` command to show the state of the worker threads and the waiting requests.

If all worker threads are processing the same requests for a long time, the server sends an alert that it might be deadlocked. All threads might be tied up processing unindexed searches.

- If a request handler is busy with a client connection, other requests sent through that request handler are forced to wait until it is able to read data. If there is only one request handler, all connections are impacted. Stack traces obtained using the `jstack` command will show that a request handler thread is continuously blocked.
- If the JVM in which the server is running is not properly configured, it can spend too much time performing garbage collection. The effect on the server is similar to that of a network or firewall configuration problem. A stack trace obtained with the `pstack` utility will show that most threads are idle except the one performing garbage collection. It is also likely that a small number of CPUs is 100% busy while all other CPUs are idle. The server will also issue an alert after detecting a long JVM pause that will include details.
- If the JVM in which the server is running has hung, the `pstack` utility should show that one or more threads are blocked and unable to make progress. In such cases, the system CPUs should be mostly idle.
- If there is a network or firewall configuration problem, communication attempts with the server will fail. A network sniffer will show that packets sent to the system are not receiving TCP acknowledgment.
- If the host system is hung or lost power with a graceful shutdown, the server will be unresponsive.

If it appears that the problem is with the server software or the JVM, work with a support provider to diagnose the problem and potential solutions.

## Problems with the Web Console

If a problem occurs when trying to use the Web Console, reasons may include one of the following:



## Chapter 6: Troubleshooting

- The web application container that hosts the console is not running. If an error occurs while trying to start it, consult the logs for the web application container.
- If a problem occurs while trying to authenticate, make sure that the target server is online. If it is, the access log may provide information about the authentication failure.
- If a problem occurs while interacting with the server instance using the Web Console, the access and error logs for that instance may provide additional information.

---

## Chapter 7: Metrics Engine API Reference

---

The Metrics Engine REST API can be used to build custom dashboards and other applications for processing and viewing data. The API interface can be accessed using standard tools and charting packages. The Metrics Engine API is also easily accessed from a Web browser.

Topics include:

[Connection and Security](#)

[List Monitored Instances](#)

[Retrieve Monitored Instance](#)

[List Available Metrics](#)

[Retrieve a Metric Definition](#)

[Perform a Metric Query](#)

[Data Set Structure](#)

[Chart Image](#)

[Google Chart Tools Datasource Protocol](#)

[Access Alerts](#)

[LDAP SLA](#)

[Pagination](#)

## Connection and Security

No sensitive user data is collected by the Metrics Engine and stored in the DBMS. If secure access to the Metrics Engine REST API is required, enable secure HTTPS connections and require authentication. A secure HTTPS Connection Handler and authentication can be enabled using `dsconfig`, if not configured during setup.

**Note** By default, the Metrics Engine can open up to 20 simultaneous database connections. The HTTP Connection handler that runs the REST API servlet has a default value of 15 connections. If the Metrics Engine receives requests through multiple HTTP Connection Handlers, make sure that the total number of request handlers does not exceed the maximum number of database connections.

When authentication is enabled, the REST API service requires HTTP basic authentication. Requests are authenticated against entries in the `api-users` LDIF backend, or entries in `cn=Root DNs,cn=config`. Root DN users have many privileges by default. To restrict access, authenticate with users in the `api-users` backend instead, to prevent the unnecessary use of more privileged account credentials.

Enable REST API authentication by setting the `require-api-authentication` property of the Metrics HTTP Servlet Extension Configuration object as follows:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Metrics Engine REST API Servlet" \
  --set require-api-authentication:true
```

Perform the following steps to add a REST API user:

1. Create a file name `api-user1.ldif` containing one or more user entries with no privileges. Below is a sample user entry.

```
dn: cn=app-user1,cn=api-users
changeType: add
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: app-user1
uid: app-user1
sn: User1
userpassword: api1
ds-pwp-password-policy-dn: cn=Default Password Policy,cn=Password Policies,cn=config
```

The password is in clear text. It will be encrypted in the next step.

2. As a privileged user that can add API users, load the entry using `ldapmodify`:

```
$ bin/ldapmodify --filename api-user1.ldif
```

3. Authenticate using either the full DN of the user added (`cn=app-user1,cn=api-users`) or the UID (`app-user1`). The user name to DN map is governed by the `identity-mapper` setting of the Metrics REST HTTP Servlet Extension configuration object.

## Secure Error Messages

When developing an application that uses the Metrics Engine API, error messages should not be delivered from the API directly to a user. Also, the application should not depend on error messages or reason text. These messages may change over time, and their presence may depend on server configuration. Use the HTTP return code and the context of the request to create a client error message that is displayed to the user.

The Metrics Engine API has an `omit-error-message-details` Metrics HTTP Servlet Extension Configuration object, that when enabled, restricts error messages to the typical reason phrase associated with the HTTP return code (such as, 'Not Found' for an HTTP 404 error). This prevents the server from inadvertently revealing information about itself or its data. Set this property as follows:

```
$ bin/dsconfig set-http-servlet-extension-prop \
  --extension-name "Metrics Engine REST API Servlet" \
  --set omit-error-message-details:true
```

## Response Codes

The following response codes are available:

Response Code	Description
200 OK	The request was processed successfully and the requested data returned.
400 Bad Request	The request contained an error. Refer to the error message to resolve the issue.
404 Not Found	The requested resource is not found or no samples are collected for the metric.
500 Internal Server Error	An unexpected server error occurred. Refer to the error message for more info.
503 Service Not Available	The metric query service is temporary offline. Refer to the error message for more info.

The following is a sample response:

**Response Body**

```
<?xml version="1.0" encoding="UTF-8"?>
<errorResponse xmlns="com.unboundid.directory.mon.api.v1.models">
  <errors reason="unknown_data_source_id" message="There are
    no metrics defined with id connections"/>
</errorResponse>
```

## List Monitored Instances

Get a list of all monitored instances along with their current status. The default format is JSON. The servlet will use the HTTP Accept header as a hint if no specific format is specified. Results are filtered using the various `instance` query parameters.

**URL** `/api/v1/instances`  
**Method** `GET`

<b>Formats</b>	JSON, XML
<b>Query Parameters</b>	<p><code>instanceHostname</code> – Hostname(s) of the servers from which data is gathered. Multiple values are evaluated as logical ORs.</p> <p><code>instanceLocation</code> – Location(s) of the servers from which data is gathered. Multiple values are evaluated as logical ORs.</p> <p><code>instanceType</code> – Types of server(s) to get data from. Possible values are:</p> <ul style="list-style-type: none"><li>• data-store</li><li>• proxy</li><li>• sync</li><li>• metrics-engine</li></ul> <p><code>instanceVersion</code> – Version(s) of the servers to get data from. Multiple values are evaluated as logical ORs.</p>

## EXAMPLES

All instances in JSON format.

```
curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/instances.json
```

All Data Store and Proxy Server instances in XML format:

```
curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/instances.xml?
instanceType=data-store&instanceType=proxy
```

**Response Code** 200 OK

**Response Body**

```
{
  "found" : 2
  "offset" : 0, #
  "instances" : [ {
    "type" : "data-store",
    "id" : "unboundid4510",
    "hostname": "unboundid5200.example.com",
    "displayName" : "unboundid4510",
    "version": "UnboundID Data Store 5.2.0.1.0",
    "operatingSystem": "Solaris",
    "status": {
      "state": "ONLINE"
    }
  }, {
    "type" : "data-store",
    "id" : "unboundid3500",
```

```

"hostname": "unboundid3500.example.com",
"displayname" : "unboundid3500",
"version": "UnboundID Directory Server 3.5.0.0",
"operatingsystem": "Linux",
"status": {
  "state": "DEGRADED",
  "unavailablealerts": [
    "replication-backlogged"
  ]
}
} ] }

```

## Retrieve Monitored Instance

Get a specific monitored instance along with its status. The default format is JSON. The servlet will use the HTTP Accept header as a hint if no specific format is specified.

<b>URL</b>	/api/v1/instances/{instance}{.format}
<b>Method</b>	GET
<b>Formats</b>	JSON, XML
<b>Query Parameters</b>	N/A
<b>Server State</b>	<p>The Metrics Engine returns the server state status of the monitored instance, which is displayed by the <code>status</code> parameter. The <code>status</code> parameter can have one of the following values:</p> <p>OFFLINE – Server cannot be contacted.</p> <p>STARTING_UP – Server is starting.</p> <p>ONLINE – Server is available.</p> <p>DEAD_LOCKED – Server is deadlocked and not able process more operations.</p> <p>UNAVAILABLE – Server is unavailable, but not offline. The server may be in lock-down mode, but may be online.</p> <p>DEGRADED – Server is available but is incapable of providing services.</p> <p>CONNECTION_ERROR – Server could not connect or has lost connection to the host.</p>

### EXAMPLE:

Instance with ID `metrics-engine` in JSON format.

```

curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/instances/metrics-engine.json

```

<b>Response Code</b>	200 OK
----------------------	--------

**Response Body**

```
{
  "displayName": "metrics-engine",
  "hostname": "metrics-engine.example.com",
  "id" : "metrics-engine",
  "operatingSystem": "Solaris",
  "status" : {
    "state" : "ONLINE"
  },
  "type" : "metrics-engine",
  "version": "UnboundID Metrics Engine 5.2.0.1"
}
```

## List Available Metrics

Get a list of metric definitions with their units, dimensions, names, and other values. The default format is JSON. The servlet will use the HTTP Accept header if no specific format is specified.

<b>URL</b>	/api/v1/metrics{.format}
<b>Method</b>	GET
<b>Formats</b>	JSON, XML
<b>Query Parameters</b>	<p><b>name</b> – Limits the results to metrics whose names contain a matching substring. The search is not case-sensitive.</p> <p><b>type</b> – Limits the results to the metrics of the specified type. Possible values are:</p> <ul style="list-style-type: none"> <li>discreteValued</li> <li>continuousValued</li> <li>count</li> </ul> <p><b>group</b> – Limits the results to the metrics within the specified group. Possible values are:</p> <ul style="list-style-type: none"> <li>Data Store Backend</li> <li>Monitoring Data Cache</li> <li>Java Virtual Machine</li> <li>LDAP</li> <li>Entry Balancing</li> <li>Data Store Entry Cache</li> <li>External Server</li> <li>Host System</li> <li>Metric Query</li> <li>Monitoring DBMS</li> </ul>

- Monitoring Data Processing
- Replication
- Sync Pipe

**instanceType** – Limits the result to metrics that uses the specified instance types as sources. Possible values are:

- ds
- proxy
- sync
- metrics-engine

**statistic** – Limits the results to metrics that provides the specified statistics. Possible values are:

- count
- average
- maximum
- minimum
- histogram

## EXAMPLES

All metrics in JSON format.

```
curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/metrics.json
```

All count type metrics in the “Data Store Backend” group providing either count or average statistics:

```
curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/metrics.json?type=count&group=data-store%20backend&statistic=count&statistic=average
```

### Note

Spaces in parameter values may be encoded as %20 or t.

**Response Code** 200 OK

**Response Body**

```
{
  "found": 7,
  "metrics": [
    {
```



```

    "countUnit": {
      "abbreviatedName": "Chkpt",
      "pluralName": "Checkpoints",
      "singularName": "Checkpoint"
    },
    "description": "Number of database checkpoints
      performed by the backend",
    "dimensions": [
      {
        "id": "backend",
        "values": [
          "userroot"
        ]
      }
    ],
    "group": "Data Store Backend",
    "id": "backend-checkpoints",
    "instanceTypes": [
      "ds"
    ],
    "name": "Backend Checkpoints",
    "shortName": "Checkpoints",
    "statistics": [
      "count"
    ],
    "type": "count"
  },
  ...

```

## Retrieve a Metric Definition

Get a specific metric definition. The default format will be JSON if none is specified. The servlet will use the HTTP Accept header as a hint if no specific format is specified.

<b>URL</b>	/api/v1/metrics/{metricId}{.format}
<b>Method</b>	GET
<b>Formats</b>	JSON, XML
<b>Query</b>	Parameters N/A

### EXAMPLE

Metric with ID backend-sequential-writes in XML format.

```

curl \
  -X GET \
  https://<metricsEngineHost>:8080/api/v1/metrics/backend-sequential-
  writes.xml

```

**Response Code** 200 OK**Response Body**

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<countMetric xmlns="com.unboundid.directory.mon.api.v1"
  id="backend-sequential-writes" name="Sequential Disk Writes"
  shortName="Sequential Writes" group="Directory Backend">
  <description>Number of Sequential I/O Disk writes
    made by backend</description>
  <instanceTypes>
    <instanceType>ds</instanceType>
  </instanceTypes>
  <statistics>
    <statistic>count</statistic>
  </statistics>
  <dimensions>
    <dimension id="backend">
      <values>
        <value>userroot</value>
      </values>
    </dimension>
  </dimensions>
  <countUnit singularName="Sequential Write"
    pluralName="Sequential Writes" abbreviatedName="Seq Wr" />
</countMetric>
```

## Perform a Metric Query

A metric query returns the collected sample data from the various monitored instances. The data returned can be presented many ways, depending on client requirements.

**Common Query Parameters**

`instanceType` – Type(s) of instances to get data from. Possible values are:

- ds
- proxy
- sync
- metrics-engine

`instanceLocation` – Location(s) of the instances from which data is collected.

`instanceHostname` – Names of the machines hosting the instances.

`instanceVersion` – Version(s) of the instances providing the data.

`instance` (multi-valued) – ID(s) of the instances from which data is collected. The instance ID is the `cn` of the external server and the same name as listed by the `status` command.

`startTime` – Include samples on or after the specified time. The time is either an absolute time in ISO 8601 format (such as 2012-08-

13T19:36:00Z) or a time relative to the `endTime` (such as -5m or -4h). By default, the start time is -5m.

`endTime` – Include samples on or before this time. The end time is either an absolute time in ISO 8601 format or a time relative to now (such as -5m or -4h). The default end time is now. Offset time values are relative to the current system clock time on the Metrics Engine.

`maxIntervals` – The number of separate intervals, between the start and end times, returned. This is considered the “resolution” of the data over time. By default, the maximum number of intervals is 1, which means all samples collected between the start and end times will be aggregated into one result according to the statistic selected.

`statistic` – Retrieve and apply this statistic to the data. Default for count based metrics is count and average for other metric types. Possible values are:

- `count`
- `average`
- `minimum`
- `maximum`
- `histogram`

`dimension` – Include only these dimension values. A colon separates the dimension name and values, which are separated by commas (for example, `op-type:add,delete`).

`pivot` – Pivot by these dimensions. A pivot keeps the data separated along different dimensional values. The value “instance” may be used to keep the data separate between different instances. For metrics that have the histogram statistic, the histogram pivot may also be used to keep the values of each histogram bucket separate.

`tz` – Specifies the timezone to be used when displaying dates. By default, it is GMT. The timezone is specified in Java Time Zone format, so “US/Central” is CST in the United States.

### Sub-parameters for the count and average statistics

Both the count and average statistics of count type metrics may have a rate scale applied to occurrences over a period of time using the `per` sub-parameter. The valid rate scaling values are:

- `s` or `second`
- `m` or `minute`
- `h` or `hour`

### Sub-parameters for the histogram statistic

The histogram statistic includes all buckets and keeps the raw value for each bucket. Graphs can be configured to show the percentage of all operations above a given threshold, such as 50 ms. These graphs are useful for looking at a small percentage of operations in a given

category. If the value falls between histogram bucket boundaries, the buckets where it falls will be included in the data. The possible values are:

- `min` - Includes in the calculation only the histogram data above the given threshold.
- `max` - Provides an upper bound on the histogram value
- `percent` - Allows the histogram values to be reported as a percentage of the overall values. Instead of returning raw counts, the value is a fraction of the total. This percentage is calculated within a pivot.

If both `min` and `max` are specified, the returned value is the sum of all buckets between and including `min` and `max`.

## Data Set Structure

The data set structure is a proprietary data structure that is space-optimized and designed to work with charting libraries like Highcharts, FusionCharts, or JFreeChart. The default format is JSON. The servlet will use the HTTP Accept header as a hint if no format is specified.

**URL**                    `/api/v1/metrics/{metricId}/dataset{.format}`  
**Method**                `GET`  
**Formats**               `JSON, XML`

### **Note**

All of the Common Query parameters apply to this resource.

Get the average response time metric for add and delete operations from 7/7/2015 for all Data Stores and Proxy Servers in Austin and Houston:

```
curl \
-X GET \
  https://<metricsEngineHost>:8080/api/v1/metrics/response-time/dataset?
instanceType=directory
  &instanceType=proxy&instanceLocation=austin&instanceLocation=houston&startTi
me=-1d
  &endTime=2015-07-07&pivot=instance&dimension=op-type:add,delete
```

Get the new connections metric and scale the value per hour in the last 5 minutes:

```
curl \
-X GET \
  https://<metricsEngineHost>:8080/api/v1/metrics/new-connections/dataset?
statistic=count;per:hour
```

Get the percentage of all occurrences in the last hour where the response-time metric has a value above 50ms:

## Chapter 7: Metrics Engine API Reference

```
curl \
-X GET \
  https://<metricsEngineHost>:8080/api/v1/metrics/response-time/dataset?
  statistic=histogram;min:50;percent&startTime=-1h
```

**Response Code** 200 OK

**Response Body** When one time interval is requested, a category dataset is returned where the first pivoted dimension values are listed as categories and each data point corresponds to a category. Subsequent pivots and histogram buckets are included as a series and subseries. This example is the result of two pivots, op-type and instance:

```
{
  "type" : "category",
  "firstSampleTime" : 1344090300000,
  "lastSampleTime" : 1344090600000,
  "metric" : {
    "type" : "discreteValued",
    "id" : "response-time",
    "name" : "Response Time",
    "shortName" : "Response Time",
    "description" : "Time for server to process an LDAP
      operation and send a response to the client",
    "group" : "LDAP",
    "instanceTypes" : [ "ds", "proxy" ],
    "statistics" : [ "average", "count", "histogram" ],
    "dimensions" : [ {
      "id" : "application-name"
    }, {
      "id" : "op-type",
      "values" : [ "Search", "ModifyDN", "Add", "Delete",
        "Compare", "Bind", "Modify" ]
    } ],
    "countUnit" : {
      "singularName" : "Operation Response Time",
      "pluralName" : "Operation Response Time",
      "abbreviatedName" : "Response Time"
    },
    "valueUnit" : {
      "singularName" : "Millisecond",
      "pluralName" : "Milliseconds",
      "abbreviatedName" : "Msec"
    }
  },
  "series" : [ {
    "label" : "unboundid35",
    "data" : [ "0", "0", "0", "0", "0", "0", "0" ]
  }, {
    "label" : "unboundid3",
    "data" : [ "0", "0", "0", "0", "0", "0", "0" ]
  } ]
}
```

```

    } ],
    "label" : "op-type",
    "categories" : [ "Search", "Delete", "Bind", "Modify",
        "Add", "ModifyDN", "Compare" ]
}

```

## Chart Image

Retrieve and display the collected metrics data. The server will generate a chart of the query result. PNG is the default format if no format is specified.

**URL** `/api/v1/metrics/{metricId}/chart{.format}`

**Method** GET

**Formats** PNG, JPEG

**Query Parameters** `width` – The width of the image. Default value is 800.

`height` –The height of the image. Default value is 600.

`showLegend` – Include a chart legend. Default value is true.

`title` – The input is a Velocity template and the `$metric` context can be used to reference anything in the Metric object.

`subTitle` – Enables adding one or more sub-titles to the chart. Velocity templates can be used.

`tz` –Specifies the time zone used for all date/times on the chart. Default is GMT.

`dateFormat` – Specifies the format of the all date/times on the chart. It uses the standard Java SimpleDateFormat strings.

`useLogScale` – Use a logarithmic scale for the range axis.

`style` – Allows customizations of various stylistic elements in the chart. The format is a semi-colon separated list of rules, where the key and value are separated by a colon. The following rules are implemented:

- `series-color` – A space separated list of colors to use for a different series in the order specified. The standard 17 CSS named colors are supported with the hex (`#445566`) and RGB (`rgb(45, 100, 255)`) notations. For example, `series-color: blue red #221122 rgb(234,255,255);`.
- `series-width` – The width of the series lines in pixels. For example, `series-width: 5px;`.
- `background-color` – The background color of the chart. For example, `background-color: white;`.
- `grid-line-width` – The width of the gridlines. If only one value is present, both the domain and range gridlines will use the same width.

Otherwise, the first value is for the domain and the second is for the range. For example, `grid-line-width: 4px 6px;`.

- `grid-line-color` – The color of the gridlines. For example, `grid-line-color: red blue;`.
- `legend-position` – The position of the legend, if present. For example, `legend-position: right;`.

### **Note**

All of the Common Query parameters apply to this resource.

For example, to get the percent CPU used by all servers over the last week, pivot by server instance as follows:

```
curl -s -o chart.png https://<MetricsEngineHost>8080/api/v1/metrics/host-system-cpu-used/chart?maxIntervals=50&startTime=-1w&pivot=instance:
```

## Google Chart Tools Datasource Protocol

Metrics data can be presented with Google's Chart Tools Datasource protocol. The Google Visualization API query language (the `tq` request parameter) is not supported. The Metrics Engine supports JSON, HTML, CSV, and TSV data formats as outlined by the Datasource protocol.

<b>URL</b>	<code>/api/v1/metrics/{metricId}/datatable</code>
<b>Method</b>	GET
<b>Formats</b>	JSON, HTML, CSV, and TSV
<b>Query Parameters</b>	<code>tqx=out:html</code> – HTML formatted output. <code>tqx=out:csv</code> – CSV formatted output. <code>tqx=out:tsv-excel</code> – TSV formatted output.  <code>tz</code> – Specifies the timezone to be used when displaying dates. The Google Visualization API assumes that the times returned are in local time. The Metrics Engine stores and returns all timestamps in GMT. This parameter specifies how the Metrics Engine presents the time. Usually, the client will pass the user's local timezone in IANA Time Zone Database format, such as "US/Central."

### **Note**

All Common Query parameters apply to this resource.

The following example gets the average response time metric for the last 5 minutes with 30 second ( $5 * 60 / 10$ ) resolution and pivoted by `op-type` and then instance in CSV format:

```
curl \
-X GET \
https://<metricsEngineHost>:8080/api/v1/metrics/response-time/datatable?
```

```
tqx=out:csv&maxIntervals=10
  &pivot=op-type&pivot=instance&tz=US/Central
```

**Response** 200 OK

**e**

**Code**

**Response Body** When only one time interval is requested, the first pivoted dimension values form the first column. For queries that request more than one time interval, the start of each time interval forms the first column. Combinations of subsequent pivoted dimension values and/or histogram buckets are included as additional columns. All date and time values are under the GMT time zone.

```
"Time","unboundid35 AVERAGE Milliseconds","unboundid3 AVERAGE
Milliseconds"
"2012-08-04T14:38:00Z","0","0"
"2012-08-04T14:39:00Z","0","0"
"2012-08-04T14:40:00Z","0","0"
"2012-08-04T14:41:00Z","0","0"
"2012-08-04T14:42:00Z","0","0"
```

The following sample illustrates using Google chart tools:

```
<html>
  <head>
    <!--Load the AJAX API-->
    <script type="text/javascript"
src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">

      // Load the Visualization API and the line chart package.
      google.load('visualization', '1.0', {'packages':['corechart']});
      // Set a callback to run when the Google Visualization API is loaded.
      google.setOnLoadCallback(drawChart);

      function drawChart() {
        var query = new google.visualization.Query
          ('https://<metricsEngineHost>:8080/
          api/v1/metrics/response-time/datatable?maxIntervals=10
          &pivot=op-type&pivot=instance');
        query.send(handleQueryResponse);
      }
      function handleQueryResponse(response) {
        if (response.isError()) {
          alert('Error in query: ' + response.getMessage() + ' '
            + response.getDetailedMessage());
          return;
        }
        var data = response.getDataTable();

        var visualization = new
          google.visualization.LineChart(document.getElementById('chart_div'));

```



```

        visualization.draw(data, null);
    }
</script>
</head>
<body>
    <!--Div that will hold the chart-->
    <div id="chart_div"></div>
</body>
</html>

```

## Access Alerts

The eventTypes and event APIs can be used to retrieve information and alerts from monitored servers. The eventTypes API provides the range of alert types that have occurred. The events API provides detail about individual alerts.

### Retrieving Event Types

The range of alerts that have been generated by monitored servers can be retrieved, with optional filtering, based on the following API definition.

**URL** /api/v1/eventTypes/[?query-parameters] - gets a list of event types

**Method** GET

**Formats** JSON, XML

**Query Parameters** instance, instanceType, startTime, and endTime. See [Performing a Metric Query](#) for a description of each parameter.

**Response Code** 200 OK

**Response Body** ["health-check-available-to-degraded", "health-check-degraded-toavailable"]

### Retrieving Events

The detailed information for one or more events can be retrieved, with optional filtering, based on the following API definition.

**URL** /api/v1/events/[?query-parameters] - gets a list of events  
/api/v1/events/{eventId} - gets a single event

**Method** GET

**Formats** JSON, XML

**Query Parameters** type – Limits the result to include only events of the specified types. See the HTML API Reference for event types.  
severity – Limits the result to include only events that have the matching severity. Valid "severity" values are: INFO, WARNING, ERROR, and FATAL.

instance, instanceType, instanceLocation, instanceHostname, instanceVersion, startTime, and endTime. See [Performing a Metric Query](#) for a description of each parameter.

limit, offset. See [Pagination](#) for a description of each parameter.

**Response Code**

200 OK

**Response Body**

```
{
  "found" : 2,
  "offset" : 0,
  "events" : [

    { "id": "9bdfdlb8-3811-4a84-b779-93553ff35f83",
      "creationDate": 1351274815559,
      "eventType": "server-starting",
      "eventSeverity": "INFO",
      "sourceProductInstance": "lockdown-test",
      "summary": "Server Starting",
      "detail": "The Directory Server is starting" },
    { "id": "9bdfdlb8-3811-4a84-b779-93553ff35f83",
      "creationDate": 1351274815559,
      "eventType": "server-starting",
      "eventSeverity": "INFO",
      "sourceProductInstance": "directory-3",
      "summary": "Server Starting",
      "detail": "The Directory Server is starting" }
  ]
}
```

## LDAP SLA

The LDAP SLA API lists the LDAP SLA objects (configuration data) and queries any single LDAP SLA object. The query of an LDAP SLA object results in the aggregated LDAP SLA configuration, scalar data containing current values for the LDAP SLA, and time-series data. Current data comes from the Threshold object. Historical data comes from a metric query. Historical data is more expensive to fetch and is only included if the client requests it. This allows an LDAP SLA query to get the configuration and current data very efficiently for clients that only need the current data. A client that needs both current and historical data can include the appropriate query parameter and get all the data in a single call.

### Retrieving the SLA Object

List the LDAP SLA objects (configuration data) and query any single LDAP SLA object. The default format will be JSON if none is specified. The servlet will use the HTTP Accept header as a hint if no specific format is specified.

**URL**

/api/v1/sla/ldap – Returns a list of all LDAP SLA configuration objects in name-order. This includes current values and status as held by the

	Threshold objects, but will only include any historical data.
	/api/v1/sla/ldap/{sla-name} – Returns a single LDAP SLA configuration object plus optional historical data.
<b>Method</b>	GET
<b>Formats</b>	JSON, XML
<b>Query Parameters</b>	<b>For the 1st URL:</b>  instance – Returns LDAP SLA's that reference the specified instance.  application-name – Returns LDAP SLA's that reference this application name.  ldap-op – Returns LDAP SLA's that reference this LDAP operation.  <b>For the 2nd URL:</b>  historical (multi-valued, optional): <ul style="list-style-type: none"><li>• time – Includes time series data.</li><li>• limits – Includes the percent of time thresholds limits that have been exceeded. Requires Threshold.</li><li>• alerts – Includes all Threshold alerts. Requires Thresholding.</li><li>• histogram – includes response-time histogram as column data)</li><li>• nines – Includes response time values that correlate to 99%, 99.9%, 99.99%, and 99.999% response-time measurements)</li></ul> startTime – (optional). The time at which the historical data starts. The default is 1hr.  endTime – (optional). The time at which the historical data ends. The default is 5m.  pivot – (optional). Historical time-series pivots by this dimension. <ul style="list-style-type: none"><li>• instance – pivot by producing server.</li><li>• ldap-op – pivot by LDAP operation.</li><li>• histogram – pivot response-time series by histogram buckets.</li></ul> maxIntervals – (optional). Number of points to include in the historical time series. The default is 100.

## EXAMPLE

Retrieving an SLA object.

```
curl -X GET http://x3550-09:8080/api/v1/sla/ldap/Acme+Identity+Portal?historical=time\n&historical=nines\n&pivot=instance\n&startTime=-15m
```

**Response Code** 200 OK**Response Body**

(JSON, sample is abbreviated)

```
{
  "name": "Acme Identity Portal",
  "applicationName": "Application 5",
  "ldapOps": ["search"],
  "servers": ["x2270-08.unboundid.lab:1389"],
  "enabled": true,
  "responseTimeState": "NORMAL",
  "throughputState": "normal",
  "currentResponseTime": 6.002752,
  "currentThroughput": 7032.794,
  "averageResponseTime": 6.212055,
  "averageThroughput": 5517.1323,
  "responseTimeWarnLimit": 8.0,
  "responseTimeCriticalLimit": 10.0,
  "throughputWarnLimit": 8000.0,
  "throughputCriticalLimit": 10000.0,
  "responseTimeSeries": {
    "type": "timeInterval",
    "firstSampleTime": 1359045070000,
    "lastSampleTime": 1359045970000,
    "rateScaling": "NONE",
    "statistic": "AVERAGE",
    "metric": {
      "type": "discreteValued",
      "id": "response-time",
      "name": "Response Time",
      "shortName": "Response Time",
      "description": "Time for server to process an LDAP operation and send a response to the client.",
      "group": "LDAP",
      "instanceTypes": ["identity-data-store", "proxy"],
      "statistics": ["average", "count", "histogram"],
      "dimensions": [{"id": "application-name", "values": ["unidentified directory application", "unidentified proxy application", "application 9", "application 5", "root user", "admin user", "application 6"]}, {"id": "op-type", "values": ["search", "modifydn", "add", "delete", "compare", "bind", "modify"]}]],
      "countUnit": {"singularName": "Operation Response Time", "pluralName": "Operation Response Time", "abbreviatedName": "Response Time"},
      "valueUnit": {"singularName": "Millisecond", "pluralName": "Milliseconds", "abbreviatedName": "Msec"}
    }
  },
  ...
}
```

## Pagination

Pagination is supported for both the metrics and instances listing URLs.

<b>Query Parameters</b>	<code>limit</code> – Specifies the maximum number of results to return. The default is to return all results.
	<code>offset</code> – Specifies how many results to skip for the first results to return.
<b>Response Parameters</b>	<code>found</code> – The number of results that satisfied the query parameters.
	<code>offset</code> – The index into the total result set where the current response begins.

---

# Index

---

## A

- aggregating data 68
- alarms 31
  - testing setup 32
- alerts
  - alarm\_cleared alert type 31
  - configure alert handlers 29
  - list of system alerts 29, 32
  - notifications and alerts 28
  - overview 31
  - testing setup 32
- alerts backend
  - alert retention time 30
  - duplicate alert suppression 31
  - overview 29
  - view information 30
- API
  - access alerts 118
  - add a REST user 104
  - chart image 115
  - connection and security 104
  - data set structure 113
  - LDAP SLA 119
  - list available metrics 108
  - list monitored instances 105
  - pagination 122
  - perform a metrics query 111
  - response codes 105
  - retrieve a metric definition 110
  - retrieve monitored instance 107

## B

- backend monitors
  - disk space usage 28
  - entries 26
- backup command 36
- base64 command 36
- broker-dashboard 75

## C

- chart builder tool
  - overview 79
- charts
  - available server charts 82
  - chart builder parameters 81
  - chart builder tool 79
  - chart image API 115
  - chart properties file 82
  - presentation details 80, 115
- cn=monitor backend 64
- collect-support-data tool 9, 36, 90
- command-line
  - available tools 35
  - default properties file 38
  - tools.properties file 37
- config-diff tool 36
- create-rc-script command 36

## D

- dashboards
  - available dashboards 75
  - configure dashboards 78
  - debug template files 78
  - metrics landing page 12
  - save custom files 79
- Data Broker charts 84

data collection

- aggregating data 68
- importing data 67
- overview 2
- performance data 63
- reduce data collected 66
- reduce frequency of collection 66
- reduce sample blocks 66
- system monitoring data 64

Data Store charts 83

Data Sync Server charts 83

demo-dashboard 77

disk space usage monitor 28

DNS caching 53

dsconfig command 36

dsframework command 36

dsjavaproperties command 36

**E**

error log publisher 25

external collector daemon 65

**G**

gauges 31

- testing related alarms and alerts 32

**H**

host system monitor provider 65

HTTP Servlet Extension object 104-105

**I**

IP address reverse name lookup 54

**J**

Java

- installing the JDK 6

JVM debugging

- during setup 97
- invalid options 98

**L**

landing page 12

ldap-dashboard 75

ldapmodify command 36

ldappasswordmodify command 36

ldapsearch command 36

ldif-diff command 36

ldifmodify command 36

Linux configuration

- filesystem swapping 9
- filesystem variables 7
- install dstat 9
- install sysstat and pstack 9
- set file descriptor limit 8
- set filesystem flushes 8

logs

- create log publisher 24
- error log publisher 25
- overview 24
- retention policies 24
- rotation policies 24

**M**

manage-extension command 36

manage-extension tool 54

memory errors 95

metric-engine-schema command 36

metrics

- continuous metrics 57
- count metrics 57
- dimensions 58

---

- discrete metrics 57
- list available metrics 108
- overview of metric types 57
- performance impact 67
- query overview 59
- sample data delays 90

Metrics Engine

- components 2
- overview 2
- start server 15
- stop server 15

Metrics Engine charts 83

monitored-servers command 36

monitored-servers tool 14

monitored servers

- add servers 14
- configure servers to monitor 9
- dsconfig tool 15
- monitored-servers tool 14
- processing time histogram 10
- stats collector 10
- tracked applications 10

monitoring entries 26

**N**

non-root user 6

normalized records 63

**P**

performance data

- overview 3

performance data fields 63

pivots 60

PostgreSQL

- backup database 33

---

- data storage 34
- install 11
- plan the backup 34
- restore the backup 35
- start the backup 35

processing time histogram plugin 10

Proxy Server charts 83

pstack utility 101

**Q**

query-metric command 36

- access metrics 61

query data

- aggregate query results 60
- pivots 60
- select query data 60
- unexpected results 96

query overview 59

queryrate command 36

**R**

REST API

- overview 2

restore command 36

revert-update command 37

review-license command 37

**S**

sample-flush-interval property 10

server-state command 37

server clock skew 65, 96

server install 11

server SDK

- extension types 54

server status 107

- example 95

---



- service level agreements 119
  - monitoring overview 68
  - SLA dashboard 76
  - SLA object 69
  - spike monitoring threshold 69-70
- setup command 37
  - troubleshooting 96
- setup tool 97
- sla-viewer-details dashboard 76
- sla-viewer dashboard 76
- SLA object 69
  - configure object 71
- Solaris configuration
  - ZFS configuration 6
- start-metrics-engine command 37
- start Metrics Engine server 16
- stats collector plugin 10, 64, 67
  - cn=monitor backend 64
- status command 37
- stop-metrics-engine command 37
- stop Metrics Engine server 16
- sum-file-sizes command 37
- supported platforms 6
- system data
  - overview 3
- system entropy 9
- system utilization monitors 65

**T**

- tools.property file 37
- tracked applications 10
- troubleshooting
  - client connections 100
  - collect support data 90

- installation 96
- memory errors 95
- performance example 92
- sample data delays 90
- server shutdown 100
- server unresponsive 101
- slow queries 91
- unexpected query results 96
- web console 101

**U**

- uninstall command 37
- uninstall server 17
- update command 37

**V**

- Velocity templates
  - multiple content types 86
  - overview 84
  - save custom files 79
  - tools context provider 88

**W**

- web console
  - configure security 20
  - configure servers 19
  - configure Tomcat 17
  - install 17
  - log into 20
  - uninstall 21
  - upgrade 21
  - URL 19

**Z**

- ZFS configuration 6-7