UnboundID® Metrics Engine

Administration Guide

Version 4.7.0

UnboundID Corp

13809 Research Blvd., Suite 500

Austin, Texas 78750

Tel: +1 512.600.7700

Email: support@unboundid.com

UnboundID

Copyright

Table of Contents

Copyright ..... **i**

Preface **1**

About UnboundID 1

Audience 1

Documentation Included with the Metrics Engine 2

Metrics Reference Documentation 2

Related Documentation 3

Chapter 1: Introduction **1**

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

Chapter 2: Installing the Metrics Engine **5**

Supported Platforms 6

Install the JDK 7

Configure a Non-Root User 8

Optimize the Solaris Operating System 8

Restrict ZFS Memory Consumption 8

Limit ZFS Transaction Group Writes 8

Configure ZFS Access to Underlying Disks 9

Configure ZFS Compression 9

Optimize the Linux Operating System 9

Set the File Descriptor Limit 9

Set the Filesystem Flushes 10

Install sysstat and pstack on Red Hat 10

Install the dstat Utility on SUSE 10

Disable Filesystem Swapping 10

Configure Identity Servers to be Monitored 10

Disk Space Requirements and Monitoring Intervals 11

Tracked Applications 11

Install the Metrics Engine 12
Add Monitored Servers to the Metrics Engine 14
Using the monitored-servers Tool 14
Using the dsconfig Tool 14
Start and Stop the Metrics Engine Server 15
Start the Metrics Engine as a Background Process 15
Start the Metrics Engine as a Foreground Process 15
Start the Metrics Engine at Boot Time 15
Stop the Metrics Engine 16
Restart the Metrics Engine 16
Uninstall the Metrics Engine 16
Install the Metrics Engine Web Console 17
Configure the Web Console 18
Configure SSL or StartTLS for the Console 19
Configure a Truststore for the Console 19
Log Into the Console 20
Upgrade the Web Console 20
Uninstall the Console 21
Chapter 3: Managing the Metrics Engine 22
Metrics Engine Error Logging 23
Logging Retention Policies 23
Logging Rotation Policies 23
Create Log Publishers 23
Error Log Publisher 24
Backend Monitor Entries 25
Disk Space Usage Monitor 26
Notifications and Alerts 27
Configure Alert Handlers 28
The Alerts Backend 28
View Information in the Alerts Backend 28
Modify the Alert Retention Time 29
Configure Duplicate Alert Suppression 29
System Alarms and Gauges 29
Testing Alerts and Alarms 30

To Test Alarms and Alerts 30
Back Up the Metrics Engine Database 32
Historical Data Storage 32
Planning the Backup 33
Start the DBMS Backup 33
Restore a DBMS Backup 34
Management Tools 34
Available Command-Line Utilities 34
The tools.property File 36
Tool-Specific Properties 36
Specify Default Properties Files 37
Server SDK Extensions 37
Chapter 4: Collecting Data and Metrics 39
Metrics Overview 40
Count Metrics 40
Continuous Metrics 40
Discrete Metrics 40
Dimensions 41
Query Overview 42
Select Query Data 43
Aggregate Query Results 43
Format Query Results 44
The query-metric Tool 44
Performance Data Collection 45
System Monitoring Data Collection 46
Stats Collector Plugin 47
System Utilization Monitors 47
External Collector Daemon 48
Server Clock Skew 48
Tune Data Collection 49
Reducing the Data Collected 49
Reducing the Frequency of Data Collection 49
Reducing the Frequency of Sample Block Creation 49
Reducing Metrics Engine Impact on Performance 49
Data Processing 50

Importing Data 50
Aggregating Data 51
Monitoring for Service Level Agreements 51
SLA Thresholds 52
Threshold Time Line 53
Configure an SLA Object 54
Chapter 5: Configuring Charts for Identity Servers 56
Available Dashboards 57
Customize the LDAP Dashboard 60
Debug Dashboard Customization 60
Preserve Customized Files 60
The Chart Builder Tool 61
Chart Presentation Details 62
Chart Builder Parameters 63
Chart Properties File 64
Available Charts for Identity Servers 64
Charts for All Servers 64
Data Store Charts 65
Proxy Server Charts 65
Sync Server Charts 65
Metrics Engine Server Charts 65
Identity Broker Charts 65
Configure Charts for the Identity Broker 66
Adding a Chart to the Identity Broker Dashboard 66
Adding the Broker Dashboard to the Identity Broker Console 67
Velocity Templates 67
Supporting Multiple Content Types 69
Velocity Context Providers 70
Velocity Tools Context Provider 71
Chapter 6: Security 72
Security Features 73
Certificates 74
Authentication Using Certificates 74
Create a Server Certificate with Keytool 75

Create a Client Certificate 77
Configure the Key and Trust Manager Providers 78
SSL and StartTLS Support 79
LDAP-over-SSL (LDAPS) Support 80
StartTLS Support 80
Configure SSL 80
Configure StartTLS 82
Authentication Mechanisms 83
Configure a SASL Mechanism Handler 83
Kerberos Configuration Considerations 91
GSSAPI Mechanism Handler Options 93
Configure Pass-Through Authentication 96
Add Attributes to Restrict Authentication 97
Configure Certificate Mappers 98
Configure the Subject Equals DN Certificate Mapper 99
Configure the Fingerprint Certificate Mapper 99
Configure the Subject Attribute to User Attribute Certificate Mapper 101
Configure the Subject DN to User Attribute Certificate Mapper 101
Chapter 7: Troubleshooting 103
Collect Support Data Tool 104
Enable JVM Debugging 104
Delays in Sample Data Availability 105
Slow Queries Based on Sample Cache Size 106
Performance Troubleshooting Example 106
Insufficient Memory Errors 109
Unexpected Query Results 109
Installation and Maintenance Issues 110
The setup Program will not Run 110
The Server will not Start 111
The Server has Shutdown 113
The Server will not Accept Client Connections 114
The Server is Unresponsive 115
Problems with the Web Console 115
Chapter 8: Metric Engine API Reference 117
Connection and Security 118

Response Codes 119
List Monitored Instances 119
EXAMPLES 120
Retrieve Monitored Instance 120
EXAMPLE: 121
List Available Metrics 121
EXAMPLES 123
Retrieve a Metric Definition 124
EXAMPLE 124
Perform a Metric Query 125
Data Set Structure 127
Chart Image 128
Google Chart Tools Datasource Protocol 130
Access Alerts 132
Retrieving Event Types 132
Retrieving Events 132
LDAP SLA 133
Retrieving the SLA Object 133
EXAMPLE 134
Pagination 135
Index **136**

Preface

The UnboundID Identity Proxy Server Guide provides procedures to install and manage an Identity Proxy Server in a multi-client environment.

About UnboundID

UnboundID Corp is a leading identity infrastructure domain solutions provider with proven experience in large-scale identity data solutions. The UnboundID solution set 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 Identity Data 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 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.

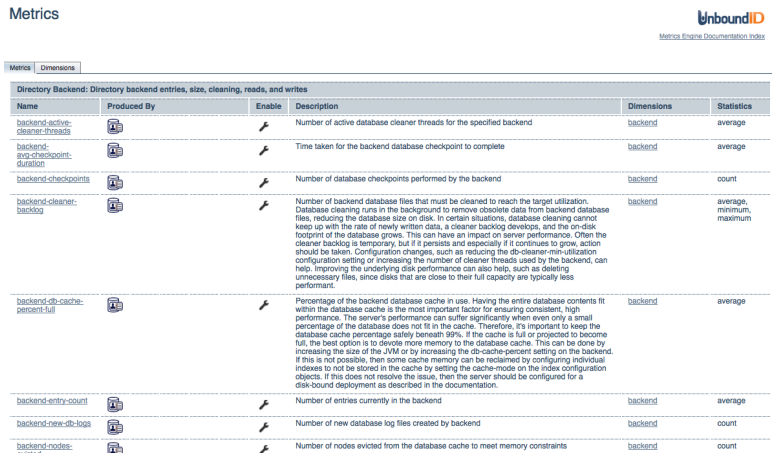
Metrics Reference Documentation

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 Metrics Documentation page, 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 Metrics Reference link opens to a list of all of the metrics collected by the Metrics Engine for each configured Identity server.



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.
- **Enable** – Provides the corresponding `dsconfig` command-line instruction to enable the metric on the producing monitored server. Hover over the wrench icon to view the `dsconfig` command.
- **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 Identity Data Store Reference (HTML)*
- *UnboundID Identity Data Store Administration Guide (PDF)*
- *UnboundID Data Sync Reference Guide (HTML)*
- *UnboundID Data Sync Administration Guide*
- *UnboundID Identity Proxy Reference (HTML)*
- *UnboundID Identity Proxy Administration Guide (PDF)*
- *UnboundID Identity Broker Reference (HTML)*

- *UnboundID Identity Broker Administration Guide (PDF)*
- *UnboundID Identity Broker Installation Guide (PDF)*
- *UnboundID Identity 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 Identity environment.

This chapter provides the following information:

Metrics Engine Overview

Metrics Engine Components

Data Collection

Charts and Dashboards

PostgresSQL DBMS Details

Metrics Engine Overview

The Metrics Engine provides insight into the transactions and performance of the UnboundID identity infrastructure. The Metrics Engine collects data from configured instances and replicas of the Identity Data Store, the Identity Proxy Server, the Identity Data Sync Server, and the Identity 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 PostgresSQL database for collected metrics. The Metrics Engine gathers data for itself and configured Identity Data Store, Identity Proxy Server, Identity Sync Server, and Identity Broker 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.

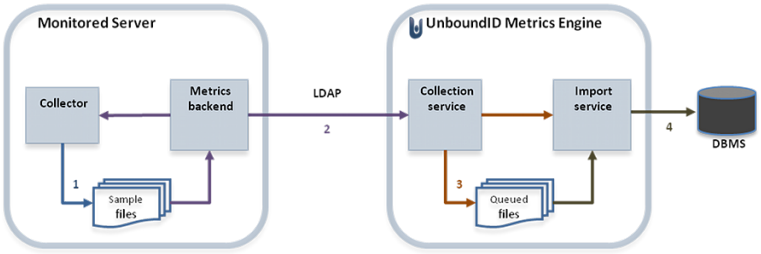
System and Status Data

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, and 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 view 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 Identity server functions. All dashboards are viewed from the Metrics Engine. The Identity Broker dashboard can be surfaced in the Identity 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 4 to 6 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

Installation of the Metrics Engine includes the following tasks:

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

Adding Monitored Servers to the Metrics Engine

Installing the Web Console

Supported Platforms

The following table lists the supported platforms and software versions.

- **Reference** – tested and confirmed that the system works as documented.
- **Yes** – the supported platform is included in UnboundID support agreements.
- **Eval Use Only** - the platform can be used to evaluate UnboundID software but should not be used for production deployments.

It is recommended that all machine resources are dedicated to the Metrics Engine Server.

Hardware Requirements

Environment	Server RAM	DBMS RAM	Disk	CPU
Small (1-6 monitored servers)	8 GB	4 GB	30 GB	4 cores
Medium (7-16 monitored servers)	20 GB	8 GB	60 GB	6 cores
Large (17-50 monitored servers)	32 GB	16 GB	180 GB	8 cores

For medium or large installations, a hardware RAID caching controller with a non-volatile write cache is preferred. The DBMS data requires multiple disk spindles.

The use of SSD storage for the DBMS files changes the system I/O performance, reducing the need to cache data or DBMS disk blocks in memory to maintain good performance. A Metrics Engine monitoring 20 servers and storing the DBMS files on SSD needs 4GB of RAM for the JVM, and 2GB of RAM for Postgres, so a system with 12GB of RAM total provides acceptable performance.

Supported Platforms & Software

Operating Systems	Supported?	Comments
RedHat Linux 5.9	Yes	
RedHat Linux 6.4	Yes	
RedHat Linux 6.5	Reference	
Solaris 10 x86 update 10	Yes	
Solaris 11.1 x86	Yes	
Solaris 11 SPARC	Yes	
AIX 7.1	Yes	
CentOS 5.9	Yes	
CentOS 6.4	Yes	
CentOS 6.5	Yes	

Operating Systems	Supported?	Comments
SUSE Enterprise 11 SP2	Yes	
Windows 2008 R2 Enterprise	Yes	
Windows Server 2012	Yes	
MacOS	Eval Use Only	

Java JDKs

JDKs	Supported?	Comments
IBM JDK 7.x 64-bit	Yes	
Oracle JDX 7.x 64-bit	Yes	
OpenJDK 7.x	Reference	

Virtual Hosts/Platforms

Virtual Hosts/Platforms	Supported?	Comments
VMWare vSphere & ESX 5.x	Yes	
IBM AIX Virtualization (LPAR, PR/SM)	Yes	

Application Servers/Servlet Containers

App Servers/Servlet Containers	Supported?	Comments
Apache Tomcat 7.x	Yes	
JBoss 7.x	Yes	

Browser Software

Auxiliary Software	Supported?
Internet Explorer 7.0+	Yes
Chrome 5.0+	Yes
Firefox 3.0+	Yes
Safari 5.0+	Yes

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 Metrics Engine'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 for use with the Metrics Engine. Most configuration changes require a reboot of the machine.

Restrict ZFS Memory Consumption

The Metrics Engine relies on database caching rather than filesystem caching 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 Metrics Engine 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 Metrics Engine 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 Metrics Engine for the change to take effect.

Set 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
   ```

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.

Install the dstat Utility on SUSE

The `dstat` utility is used by the `collect-support-data` tool and can be obtained from the OpenSuSE project website.

Disable Filesystem Swapping

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

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

Configure Identity Servers to be Monitored

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

- Identity Data Store
- Identity Proxy
- Identity Data Sync
- Idenity Broker

The Metrics Engine requires all monitored servers to be version 3.5.0 or later. 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. See the *UnboundID Data Store Administration Guide* for information about tracked applications.

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 clients application 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 Identity Data Store and Identity 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. Refer to the *UnboundID Identity Data Store Administration Guide* for Tracked Application configuration details.

The following sets the required properties 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 Metrics Engine

Use the `setup` tool to install the Metrics Engine.

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

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

   ```
   $ unzip UnboundID-Metrics-Engine-<version>.zip
   ```

3. Change to the server root directory.

   ```
   $ cd UnboundID-Metrics-Engine
   ```

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. Type **yes** and 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 (`postgres`). 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 Metrics Engine, or press **Enter** to accept the default.
14. Create the initial root user DN for the Metrics Engine, or press Enter to accept (`cn=Directory Manager`)

15. Enter and confirm a password for this account.
16. Enter an option to enable HTTP support and press **Enter**.

```
How would you like to enable support for HTTP clients?

1) HTTP
2) HTTP with SSL
3) Both HTTP and HTTP with SSL

Enter option:
```

17. Based on the option selected, enter the ports for the Metrics Engine to accept HTTP, HTTPS, or both client connections, or press **Enter** to accept the defaults:
    a. **HTTP:** `8080`
    b. **HTTPS:** `8443`
18. Enter the port on which the Metrics Engine will accept LDAP client connections, or press **Enter** to accept the default (`2389`).
19. To enable LDAPS, type **yes**, or press **Enter** to accept the default **no**.
20. If LDAPS is enabled, enter the port on which the Metrics Engine will accept LDAPS client connections, or press **Enter** to accept the default (`2636`).
21. To enable StartTLS, type **yes**, or press **Enter** to accept the default **no**.
22. 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.
23. To specify particular addresses on which this server will listen for client connections, type yes and press Enter. By default the server listens on all available network interfaces. To keep the default behavior, press Enter for (no).
24. Choose an option to assign the amount of memory that the server should allocate to the Metrics Engine and press **Enter**.
    a. **1) Aggressive** – The system is dedicated to running only this server (use 7g of 21g total memory)
    b. **2) Minimal** – Use the bare minimum memory (2G), which is useful for evaluating

product functionality at a small scale only.

c. **3) Manual** – Enter the amount of memory explicitly

25. Press **Enter** (`yes`) to start the server when configuration is complete.
26. Press **Enter** to install the Metrics Engine with the defined parameters.

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 Identity 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

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

The following example uses the `dsconfig` tool for configuring a server.

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 Metrics Engine 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. Note that 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.

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.

To avoid destroying and re-creating the JVM, use an internal restart, which can be issued over LDAP. The internal restart will keep the same Java process.

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

```
$ bin/stop-metrics-engine --restart \
  --hostname 127.0.0.1
```

Uninstall the Metrics Engine

Use the `uninstall` command-line utility to uninstall the Metrics Engine using in 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 Metrics Engine.

Perform the following steps to uninstall in interactive mode:

1. Navigate to the server root directory.

```
$ cd UnboundID-Metrics-Engine
```

2. Start the uninstall command:

```
$ ./uninstall
```

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

Install the Metrics Engine Web Console

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

Note The Management Console supports JBoss 7.1.1 or later.

The following steps use Apache Tomcat as the container.

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

```
3RD-PARTY-LICENSE.TXT
LICENSE.TXT
README
metricsengconsole.war
```

4. Create a `metricsengconsole` directory in the `apache-tomcat-<version>/webapps` directory and copy the `metricsengconsole.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/metricsengconsole
$ cp metricsengconsole.war apache-tomcat-
<version>/webapps/metricsengconsole
```

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

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

6. The `WEB-INF/web.xml` file can be edited to point to the correct Metrics Engine instance. Uncomment the necessary parameters and change the host and port to match the server. For example, the server or servers that the console uses to authenticate with can be listed using the following parameters:

```
<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. Start the Metrics Engine 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`.)

9. 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
```

10. Open a browser to `http://hostname:8080/metricsengconsole`. By default, Tomcat listens on port 8080 for HTTP requests.

Configure the Web Console

The 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 `metricsengconsole/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 so that it will communicate with all of its servers over SSL or StartTLS.

1. Open the `metricsengconsole/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 Metrics Engine.

```
<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 `metricsengconsole/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-Metrics-Engine
   ```

2. Start the Metrics Engine.

   ```
   $ start-metrics-engine
   ```

3. Start the Apache Tomcat application server.

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

4. Open a browser to `http://hostname:8080/metricsengconsole/`.
5. Type the root user DN (or any authorized administrator user name) and password, and then click **Login**.
6. On the Management 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 Console:

1. Shut down the console and servlet container.
2. In the current deployment of the Console, move the `webapps/metricsengconsole/WEB-INF/web.xml` file to another location.

3. Download and deploy the latest version of the Console. See Install the Metrics Engine 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 Console:

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

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

4. Remove the `webapps/metricsengconsole` directory with the following command:

```
$ rm -rf webapps/metricsengconsole
```

5. 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.

This section includes the following information:

Metrics Engine Error Logging

Backend Monitor Entries

Configure Alert Handlers

The Alerts Backend

System Alarms and Gauges

Backup the Metrics Engine Database

Management Tools

Server SDK Extensions

Metrics Engine Error Logging

The Metrics Engine provides error loggers that provide 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.

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 Metrics Engine rotates the current log and starts a new log.

Log retention is only effective if a log rotation policy is configured. When a retention limit is reached, the Metrics Engine removes the oldest archived log prior to creating 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 7 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 (`bin/dsconfig`). 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 Metrics Engine'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 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',
```

```
'-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

The Metrics Engine exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers, including the Metrics Engine, through SNMP, the Management Console, JConsole, LDAP command-line tools, and the Periodic 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 Metrics Engine including the number of operations, information on each operation, and the number of active persistent searches.
Backends	Provides general information about the state of a Metrics Engine backend, includ-

Monitoring Components

Component	Description
	ing 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 Metrics Engine including a name followed by an equal sign and a quoted value, such as `connID="15", connectTime="20100308223038Z"`.
Connection Handlers	Provides information about the available connection handlers on the Metrics Engine including the LDAP and LDIF connection handlers.
Disk Space Usage	Provides information about the disk space available to various components of the Metrics Engine.
General	Provides general information about the state of the Metrics Engine, 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), number of processing times between 0ms and 1ms, etc.
System Information	Provides general information about the system and the JVM on which the Metrics Engine is running, including system host name, operation system, JVM architecture, Java home, Java version, etc.
Version	Provides information about the Metrics Engine version, including build ID, version, revision number, etc.
Work Queue	Provides information about the state of the Metrics Engine 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, number of busy worker threads, etc.

Disk Space Usage Monitor

The disk space usage monitor provides information about the amount of usable disk space available for Metrics Engine 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.

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 Metrics Engine 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 may also be defined as a percentage or an absolute size. Once the amount of usable space drops below this threshold, the Metrics Engine 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

In addition to standard error logging, the Metrics Engine provides delivery mechanisms for account status notifications and administrative alerts using SMTP, JMX, or SNMP. Alerts 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.

The Metrics Engine provides a number of alert handler implementations, including:

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 Metrics Engine. 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. The Metrics Engine supports 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

The Metrics Engine stores recently generated administrative alerts under the `cn=alerts` branch. The backend makes it possible to obtain admin alert information over LDAP for use with remote monitoring. 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 Metric's Engine 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 Metrics Engine 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 Metrics Engine. 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 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 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.

<p>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 '18.583333333333332'.
         :                : 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. The REST API can be used to extract the data on a regular basis and archive it in a set of CSV files. 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 you have 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
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 for PostgreSQL backup is provided at `http://www.postgresql.org/docs/9.2/static/backup-file.html`.

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 Metrics Engine provides the following command-line utilities, which can be run directly 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 Managing the tools.properties File 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 Metrics Engine on UNIX-based systems.
dsconfig	View and edit the Metrics Engine configuration.
dsframework	Manage administrative server groups or the global administrative user accounts that

Command Line Tools

Command-Line Tool	Description
	are used to configure servers within server groups.
dsjavaproperties	Configure the JVM arguments used to run the Metrics Engine and associated tools. Before launching the command, edit the properties file located in `config/java.properties` to specify the desired JVM options and `JAVA_HOME` environment variable.
ldapmodify	Perform LDAP modify, add, delete, and modify DN operations in the Metrics Engine.
ldappasswordmodify	Perform LDAP password modify operations in the Metrics Engine.
ldapsearch	Perform LDAP search operations in the Metrics Engine.
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 UnboundID Metrics Engine. Extension bundles are installed from a zip archive or file system directory. The Metrics Engine will be restarted if running to activate the extension(s).
metric-engine-schema	Show current and required UnboundID Metrics Engine 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 Metrics Engine backend.
revert-update	Returns a server to the version before the last update was performed.
review-license	Review and/or indicate your acceptance of the product license.
server-state	View information about the current state of the Metrics Engine process.
setup	Perform the initial setup for the Metrics Engine instance.
start-metrics-engine	Start the Metrics Engine.
status	Display basic server information.
stop-metrics-engine	Stop or restart the Metrics Engine.
sum-file-sizes	Calculate the sum of the sizes for a set of files.
summarize-config	Generate a configuration summary of either a remote or local Metrics Engine instance. By default, only basic components and properties will be included. To include advanced components, use the `--advanced` option.
uninstall	Uninstall the Metrics Engine.
update	Update the Metrics Engine 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. Before upgrading a server, make sure that it is capable of starting without severe or fatal errors. During the update process, the server is stopped if running, then the update is

Command Line Tools

Command-Line Tool	Description
	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 revert-update tool for information on reverting an update.

The tools.property 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.

The Metrics Engine supports two types of properties file: default properties files that can be applied to all command-line utilities, or tool-specific properties file that can be specified using the `--propertiesFilePath` option. All of the Metrics Engine'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 Metrics Engine 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.bindPasswordFile=/ds/config/password
```

Specify Default Properties Files

The Metrics Engine 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 Metrics Engine uses the following evaluation ordering 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 Metrics Engine takes its options from the specified properties file.
- If no options are used on the command line including the `--propertiesFilePath` option (and `--noPropertiesFile`), the Metrics Engine 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`).

Server SDK Extensions

Custom server extensions can be created with the UnboundID 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 Server SDK enables creating extensions for the Identity Data Store, Identity Proxy, Metrics Engine, Identity Broker, and Identity Data Sync 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 data indicates the most current status of each monitored server.
- Alert data reflects the alerts emitted by each server.
- Performance data exposes the `cn=monitor` subtree of each product 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.

This section includes the following information:

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.
- **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.
- **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.

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 5 seconds or 2 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 Synchronization 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`

At each measurement interval for each sync pipe on each Synchronization Server, there will be a value for each of the `pipe-result` values. So, for a single Synchronization Server with two sync pipes, `pipe-one` and `pipe-two`, the samples generated for each sample period look like the following. Note that 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 Identity 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 the format requested by the client. Results can be returned in one of the following formats:

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

The query-metric Tool

The `query-metric` tool enables access to all the metrics gathered by the server. The `query-metric` tool is a client application of the Metrics Engine API. 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, for a 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 may 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 and operation type. The application name reflects the connection criteria of the request. The operation type corresponds to the LDAP operation, such as add, bind, or search. So, 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
max_val	Real	Maximum value for this sample
val1-15	Long	Histogram bucket values

System Monitoring Data Collection

All Identity servers have the ability to monitor the health of the server and 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 applic-
ations.
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 in synchronized. The more time skew there is between monitored servers and the Metrics Engine, the less accurate is the time correlation across samples from different servers.

The Metrics Engine track 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 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

Identity servers all 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 Periodic Stats Logger or Stats Collector plugins.

The Periodic 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 Periodic 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 2 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 via 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 first 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 1 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 you can configure the maximum age of each level. The following table describes 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 contributes 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 Identity Data Store and Identity Proxy servers and can be routed through the Alert Handler to a monitoring console or administrator.

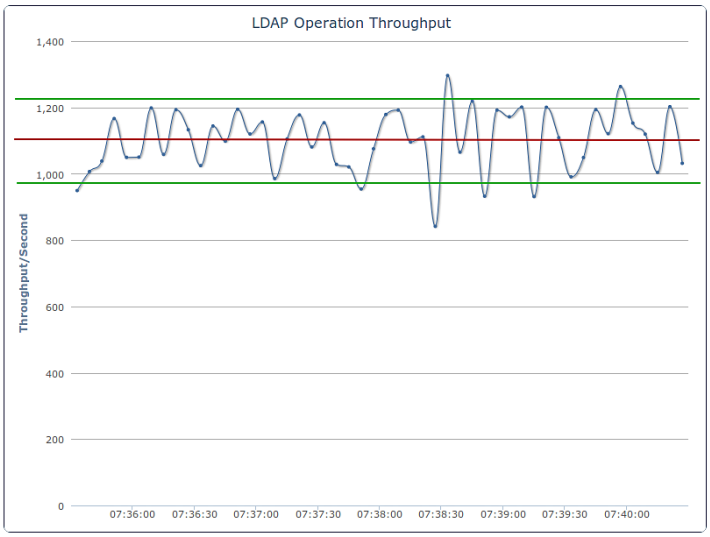
The SLA object can report the aggregate performance of all configured servers. The SLA object is configured with the following:

- **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.

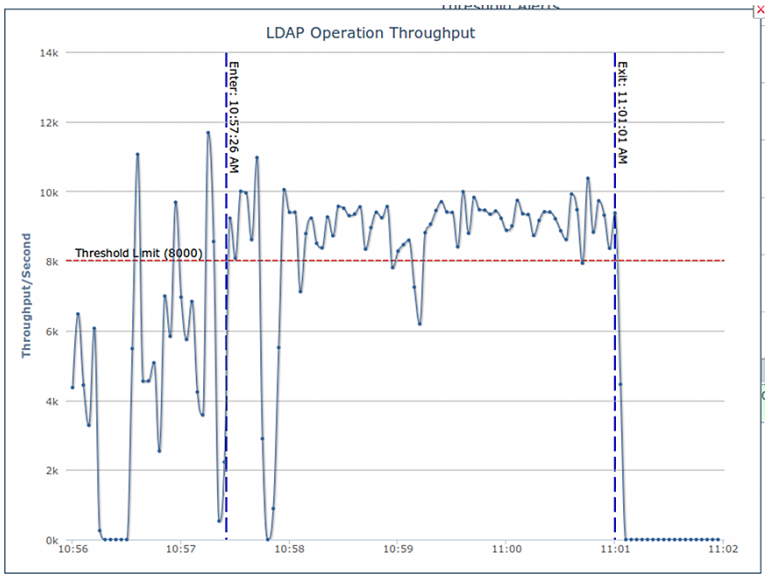
SLA Thresholds

The Metrics Engine uses a Monitoring Threshold mechanism that has two configuration 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.

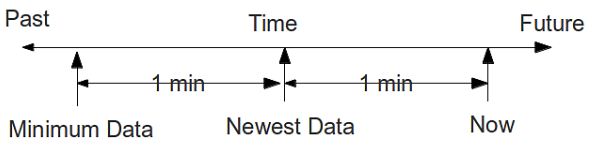


- **Static Level Monitoring Threshold**. Used to configure performance limits on a specific measurement, where the specified limits are fixed values that do not change over time. A Static Level Monitoring Threshold has warning and critical limits, and will enter or leave an alerted state when the monitored 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 1 minute old (Newest Data). Each threshold evaluation requires at least 1 minute of new data (Minimum Data). At time 'Now,' the threshold is working with data that is between 1 and 2 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 1 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 2 and 3 minutes.

Because the Metrics Engine can capture metrics data with a very fine time resolution (1 second data is the default), the data is often very "noisy." By default, the data is time-averaged (using 5 consecutive 1-second samples to produce a single 5-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 Metrics Engine Configuration Reference (HTML) documentation in the `<server-root>/docs` directory.

The following steps use the `dsconfig` command-line tool. These tasks can also be done with `dsconfig` in interactive mode.

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 (`alert-condition: 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 the 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`. 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 for Identity Servers

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

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

This chapter includes the following information:

Available Dashboards

Available Charts for Identity Servers

The Chart Builder Tool

Configure Charts for Identity Broker

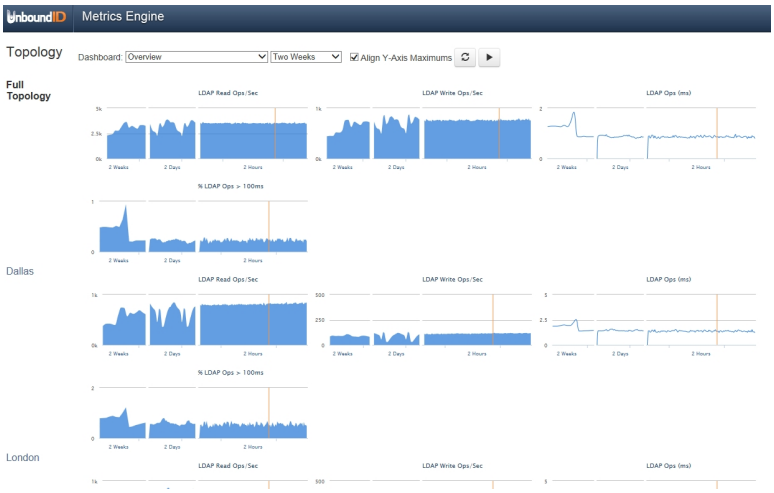
Velocity Templates

Available Dashboards

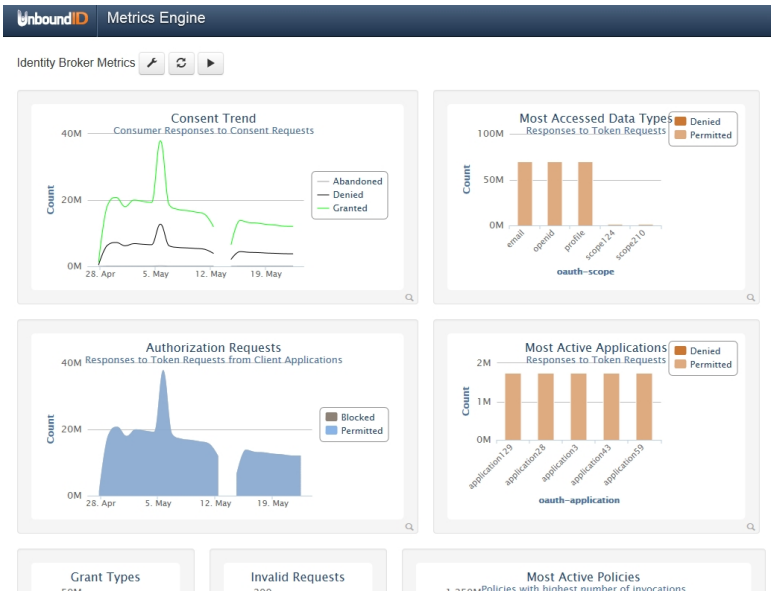
The Metrics Engine includes several dashboards that can be used to display information for all servers in a data center, specific applications, or SLA specifics. The following dashboards are available:

ldap-dashboard – Displays charts for Identity Data Store, Identity Proxy, and Identity 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:

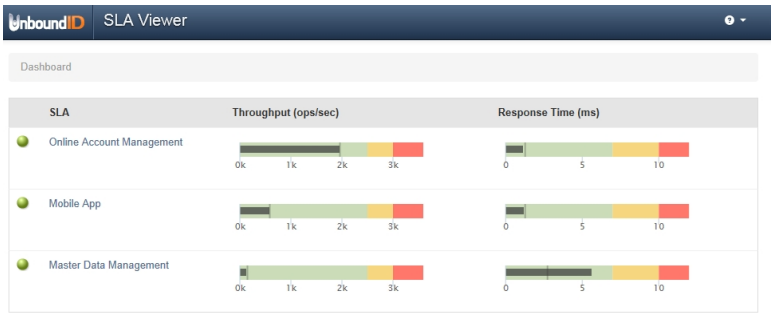
- Individual server, server location, or server type.
- Varying level of detail adjusted by server type.
- Time scale, providing either a recent or more historical data view.



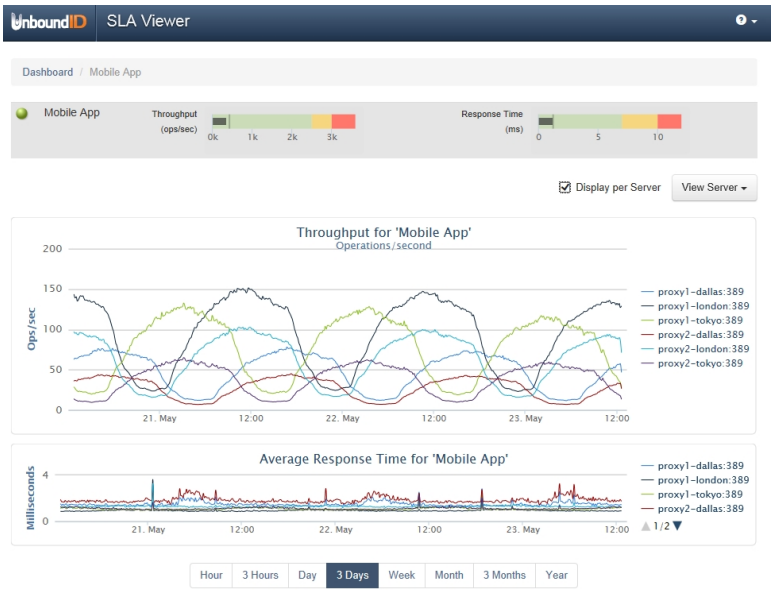
broker-dashboard – Displays charts for configured Identity Broker servers. Charts include information for data consent, applications requesting access to data, authorization, and policy activity. This dashboard is viewed from a browser at `http://<metrics-host>:<port>/view/broker-dashboard`, and can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



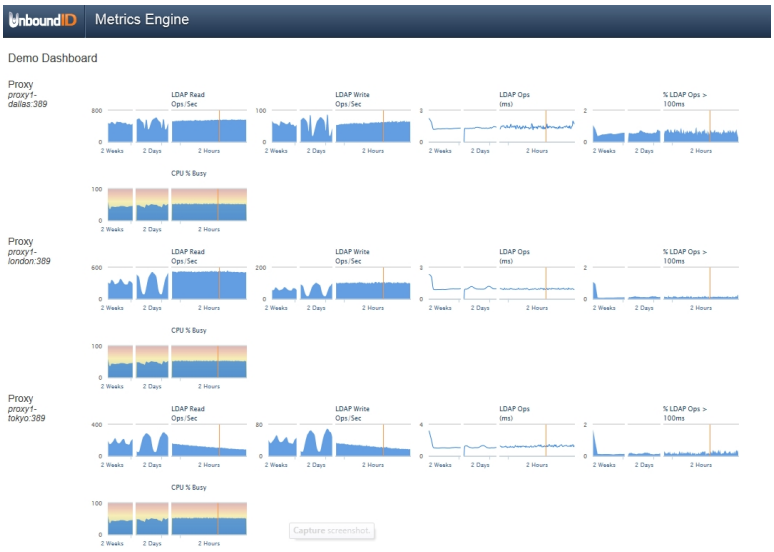
sla-viewer – Displays throughput and response time graphs, and status for configured SLAs. This dashboard is viewed from a browser at `http://<metrics-host>:<port>/view/sla-viewer`. See Monitoring for Service Level Agreements for information about configuring SLAs.



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.



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.

If 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 can 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

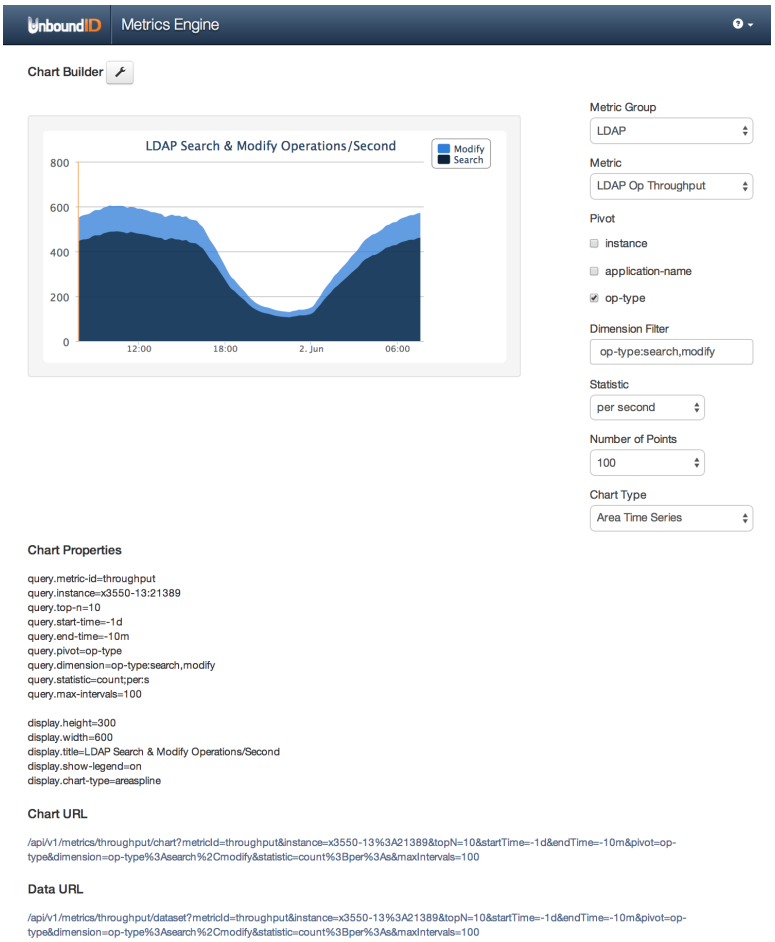
The Chart Builder tool is used to create performance charts for all configured servers. The Broker dashboard can be surfaced on the Identity Broker Console Metrics page. See Configure Charts for Identity Broker for more information.

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 styes and referencing 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. If you open the drop-down list and hover over a metric, you can 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	Displays the type of "measurement" that may exist for each metric. For example, each response-time sample contains: • number of operations (count) • average time-per-op (average) • histogram-of-operation-time (histogram) 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: • 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). • The count and histogram statistics have high fidelity over time because they time-aggregate perfectly. • The average statistic loses fidelity over time, because as the time-window for averaging gets larger, the highs and lows are lost.
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: • Area Time Series • Bar Chart • Column Chart • Pie Chart • Stacked Bar Chart

Chart Builder Parameters

Parameters	Description
	• Stacked Column Chart • Time Series
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
- LDAP Response Time
- LDAP Response Time Outliers
- LDAP Worker Thread Percent Busy
- LDAP Average Operations in Progress
- LDAP Average Queue Size
- System CPU
- System Memory Percent Free
- System Network Read MB
- System Network Write MB
- System Disk Busy
- System Disk Service Wait
- System Disk Read MB

- LDAP Open Connections
- LDAP New Connections
- System Disk Write MB

Data Store Charts

The following charts are displayed on the LDAP 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 LDAP Dashboard for the Proxy Server:

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

Sync Server Charts

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

- Metrics Queries Per Minute
- Metrics Query Time
- Metrics Query Time Max
- Metrics Query Time Histogram
- Metrics Cache Entry Count
- Metrics Cache Hit Count
- Metrics Cache Miss Count
- Metrics Cache Expired Count
- Metrics Cache Evicted Count
- Metrics Import Delay
- Metrics Load Time
- Metrics DBMS Cluster Time

Identity Broker Charts

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

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

Configure Charts for the Identity Broker

Metrics charting and the Identity Broker Dashboard use the same library. When configured, charts for the Broker dashboard display on the Identity Broker Console Metrics page. The dashboard page asynchronously fetches metric data for all charts. While most metric queries respond quickly (50-100ms), some queries may take longer. If a chart takes too long, consider making changes to the query to reduce the amount of data it gathers.

Adding a Chart to the Identity Broker Dashboard

Perform the following steps to add a chart to the Identity Broker Console Metrics page:

1. On the Metrics Engine, start the chart builder tool:

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

2. Configure the chart.
3. Create a properties file and copy and paste the Chart Properties on the Chart Builder page into the file. The property file directly correlates the listed parameters to the Metrics Engine REST API `DataSetResource.getDataSet` method, which is accessed with a GET request to `/metrics/{metric-id}/dataset`. If no values are specified for a property, then the default values will be used. If you want to exclude a parameter, use IGNORE as a value (e.g., display.title=IGNORE). Only the query.metric-id property is required.

   ```
   query.metric-id=metric-sample-query-qualifier-count
   display.height=400
   display.width=600
   query.top-n=10
   query.start-time=-4w
   query.end-time=-1m
   display.title=Throughput & Response Time
   display.sub-title=For All Instances
   display.y-axis-title=Metric Qualifier Count
   display.series-color-map=throughput=#ff0000
   display.show-legend=true
   display.series-name-map=throughput=DS-Throughput,response-time=DS-
   ResponseTime
   display.series-colors=#008000,#ff00ff,#0000ff,#00ff00
   query.pivot=metric
   query.dimension=metric:throughput,response-time
   query.max-intervals=25
   ```

4. Copy the `broker-dashboard.vm` file in the `<server-root>/-config/velocity/templates` directory to `<server-root>/velocity/templates`. Files that are saved to this location and customized are preserved after an upgrade.
5. Modify the following section by adding a file reference. For multiple charts, remove the comments and add a chart name to the `$includedCharts` list. For this example, add a single chart definition after the `#end` statement:

```
##
## Only include the charts we want to render...
##

# set ($includedCharts = ["oauth-token-request", "oauth-exceptions",
       "oauth-granttype", "my-chart"])

# foreach( $chartDefinitionFileName in $includedCharts )
# set($chartDefinition =
  $dashboard.chartDefinitionsByFileName.get($chartDefinitionFileName))
# parse("_chart.vm")
# end

# set($chartDefinitionFileName = "my-chart")
# set($chartDefinition = $dashboard.chartDefinitionsByFileName.get
     ($chartDefinitionFileName))
# parse("_chart.vm")
```

Adding the Broker Dashboard to the Identity Broker Console

Perform the following steps to add the Broker Dashboard to the Identity Broker Console Metrics page:

1. On the Identity Broker server, change the `dashboard_url` property on the Identity Broker Console web application to reference the template file:

```
$ bin/dsconfig set-web-application-extension-prop \
  --extension-name Broker-Admin-Console \
  --set dashboard-url:https://example.com:8443/view/my-broker-dashboard
```

2. Restart the Identity Broker server or disable and re-enable the Broker Apps Connection Handler for changes to take affect.

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 `$ubid_request` variable. In general, request attributes are added by server components processing the HTTP request. Also the HTTP request parameters map is available as `$ubid_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 `$ubid_session` variable. Like request attributes, session attributes are also added by server components processing the HTTP request. The lifetime of these attributes persists until the user's session has ended.
- **expose-server-context** – Indicates whether a Server SDK server context is accessible to templates using the `$ubid_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`.
- **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 on 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 arbitrarily 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.

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 `http://velocity.apache.org/tools`.

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: Security

UnboundID servers provide a full suite of security features to protect communication with clients, to establish trust between server components (for example, for replication and administration), and to secure data.

This section includes the following information:

Security Features

Certificates

SSL and StartTLS Support

Authentication Mechanisms

Configure Certificate Mappers

Security Features

The UnboundID servers support a strong set of cryptographic mechanisms to secure communication and data. The following security-related features are available:

SSL/StartTLS Support – Used to encrypt communication between the client and the server. Administrators can configure different certificates for each connection handler, or use the same certificate for all connection handlers. Additionally, the server allows for more fine-grained control of the key material used in connecting peers in SSL handshakes and trust material for storing certificates.

Message Digest/Encryption Algorithms – One-way message digests (CRYPT, 128-bit MD5, 160-bit SHA-1, and 256-bit, 384-bit, and 512-bit SHA-2 digests with or without salt) as well as a number of reversible encryption algorithms (BASE64, 3DES, AES, RC4, and Blowfish) used for storing passwords. Passwords are not made available in unencrypted form. Encrypted password storage should only be used if using an authentication mechanism that requires the server to have access to the clear-text representation of passwords, like CRAM-MD5 or DIGEST-MD5.

SASL Mechanism Support – Includes ANONYMOUS, CRAM-MD5, DIGEST-MD5, EXTERNAL, PLAIN, and GSSAPI support. The server supports two types of one-time password (OTP) mechanisms for multi-factor authentication: UNBOUNDID-TOTP SASL and UNBOUNDID-DELIVEREDOTP SASL. The UNBOUNDID-TOTP SASL mechanism allows multi-factor authentication to the server using the time-based one-time password (TOTP) code. The UNBOUNDID-DELIVERED-OTP SASL mechanism allows multi-factor authentication to the server by delivering a one-time password to the the end user through some out-of-band channel, such as email or SMS.

Password Policy Support – Includes customizable password attributes, maximum password age, maximum password reset age, multiple default password storage schemes, account expiration, and idle account lockout. The server also supports a number of password storage schemes such as one-way digests (CRYPT, MD5, SMD5, SHA, SSHA, SSHA256, SSHA384, SSHA512) and reversible encryption (BASE64, 3DES, AES, RC4, BLOWFISH). Multiple password validators can be used such as maximum password length, similarity to current password and the set of characters used.

Full-Featured Access Control System – Used to determine whether a given operation is allowable based on a wide range of criteria. The access control system allows administrators to grant or restrict access to data, restrict the use of specific types of controls and extended operations and provides strong validation for access control rules before accepting them.

Client Connection Policies Support – Controls which clients are connected to the server, how they are connected, and what resources or operations are available to them. For example, client connection criteria can be defined to block IP addresses or domains that are known to attempt brute force attacks. Client connection policies can also be configured to restrict the type of operations, controls, extended-operations, SASL mechanisms, search filters and resource limits available to the client. For example, you can configure a client connection policy that limits the number of concurrent connections or rejects all requests on unsecured connections.

Backup Protection – Protects the integrity of backup contents using cryptographic digests and encryption. When generating a backup, a cryptographic digest of the backup contents can be generated and digitally signed. The server also has options to compress and/or encrypt the contents of the backup. When restoring the backup, the server can verify that the digest matches the content of the backup and generates an error if the backup has been changed from when it was initially written, making it tamper-evident. The server can also verify the integrity of a backup before restoring it.

Certificates

Certificates can be generated and managed using a variety of commonly available tools, such as the Java keytool utility provided with the Java SDK. The keytool utility can be used to create keystores, which hold key material used in the course of establishing an SSL session, and truststores, which are consulted to determine whether a presented certificate should be trusted.

Many companies have their own certificate authorities or have existing certificates. Follow the guidelines specific to your company's implementation.

The UnboundID servers support the following three keystore types:

Java Keystore (JKS) – In most Java SE implementations, the JKS keystore is the default and preferred keystore format.

PKCS#12 – This keystore type is a well-defined standard format for storing a certificate or certificate chain, and may be used to hold certificates already in use for other types of servers. Most other servers that provide a proprietary format for storing certificates provide a mechanism for converting those certificates to PKCS#12.

PKCS#11 – Also, known as Cryptoki is a format for cryptographic token interfaces for devices, such as cryptographic smart cards, hardware accelerators, and high performance software libraries. PKCS#11 tokens may also offer a higher level of security than other types of keystores, and many of them have been FIPS 140-2 certified.

Authentication Using Certificates

Two mechanisms for certificate-based authentication are supported:

Client Certificate Validation – The server can request that the client present its own certificate for authentication during the SSL or StartTLS negotiation process. If the client presents a certificate, then the server will use the trust manager provider configured for the associated connection handler to determine whether to continue the process of establishing the SSL or StartTLS session. If the client certificate is not accepted by the trust manager provider, the server will terminate the connection. Even if the client provides its own certificate to the server during the process of establishing an SSL or StartTLS session, the underlying LDAP connection may remain unauthenticated until the client sends an LDAP bind request over that connection.

SASL EXTERNAL Certificate Authentication – Enables a client to authenticate itself with the server by presenting a certificate presented during SSL or StartTLS negotiation, outside of

the LDAP communication. Once the client has established a secure connection to the server, it may send a SASL EXTERNAL bind request to the server to request that the server attempt to identify the client based on information contained in that certificate. The server will then use a certificate mapper to identify exactly one user entry that corresponds to the provided client certificate, and it may perform additional verification, such as requiring the client certificate be present in the `userCertificate` attribute of the user's entry. If the certificate mapper cannot identify exactly one user entry for that certificate, or if its additional validation is not satisfied, then the bind attempt will fail and the client connection will remain unauthenticated.

Create a Server Certificate with Keytool

The keytool utility enables management of public/private key pairs, x509 certificate chains and trusted certificates. The keys and certificates are stored in a keystore, which is a password-protected file with a default format of JKS. Each key and trusted certificate in the keystore is accessed by its unique alias.

The following procedure creates a keystore, generates a public/private key pair, and creates a self-signed certificate based on the key pair. This certificate can be used as the server certificate or it can be replaced by a CA-signed certificate chain with additional keytool commands.

The `-dname` option is used to specify the certificate's subject, which is generally a CN attribute with a value equal to the fully-qualified name of the server. If the `-dname` option is omitted, the utility prompts for input. The certificate is valid for 180 days.

Perform the following steps to create a server certificate using Keytool:

1. Change to the directory where the certificates will be stored.

   ```
   $ cd /ds/UnboundID-<server>/config
   ```

2. Use the keytool utility to create a private/public key pair and a keystore. The keytool utility is part of the Java SDK (`${JAVA_HOME}/bin`).

   ```
   $ keytool -genkeypair \
     -dname "CN=server.example.com,ou=Metrics Engine Certificate,
        O=Example Company,C=US"\
     -alias server-cert \
     -keyalg rsa \
     -keystore keystore \
     -keypass changeit \
     -storepass changeit \
     -storetype JKS \
     -validity 180 \
     -noprompt
   ```

   The -keypass and -storepass arguments can be omitted to cause the tool to interactively prompt for the password. Also, the key password should match the keystore password.

3. View the keystore. Notice the entry type is privateKeyEntry which indicates that the entry has a private key associated with it, which is stored in a protected format to pre-

vent unauthorized access. Also note that the **Owner** and **Issuer** are the same, indicating that this certificate is self-signed.

```
$ keytool -list -v -keystore keystore -storepass changeit
```

```
Keystore type: JKS
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: server-cert
Creation date: Sep 30, 2011
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=server.example.com, OU=Metrics Engine Certificate, O=Example
Company, C=US
Issuer: CN=server.example.com, OU=Metrics Engine Certificate, O=Example
Company, C=US
Serial number: 4ac3695f
Valid from: Wed Sep 30 09:21:19 CDT 2011 until: Mon Mar 29 09:21:19 CDT
2012
Certificate fingerprints:
MD5: 3C:7B:99:BA:95:A8:41:3B:08:85:11:91:1B:E1:18:00
SHA1: E9:7E:38:0F:1C:68:29:29:C0:B4:8C:08:2B:7C:DA:14:BF:41:DE:F5
Signature algorithm name: SHA1withRSA
Version: 3
```

4. If you are going to have your certificate signed by a Certificate Authority, skip to step 7. Otherwise export the self-signed certificate. Then examine the certificate.

```
$ keytool -export -alias server-cert -keystore keystore -rfc -file
server.crt
Enter keystore password:
Certificate stored in file <server.crt>
```

5. Import the self-signed certificate into a truststore. When prompted, type **yes** to trust the certificate.

```
$ keytool -importcert -alias server-cert -file server.crt \
  -keystore truststore -storepass changeit
```

6. View the truststore with the self-signed certificate. If using this certificate as the server certificate, this is the final step.

```
$ keytool -list -v -keystore truststore -storepass changeit
```

7. To create a production-ready certificate, continue by creating the Certificate Signing Request (CSR) by writing to the file server.csr. Follow the instructions of the third-party Certificate Authority (CA), and submit the file to a CA. The CA authenticates you and then returns a certificate reply, which you can save as signed.crt.

```
$ keytool -certreq -v -alias server-cert -keystore keystore \
  -storepass changeit -file server.csr

Certification request stored in file <server.csr>
Submit this to your CA
```

8. If working with a third-party CA, both the key and trust stores should include information about the CA's root certificate as well as any intermediate certificates used to sign the server certificate. Obtain the CA root and any intermediate certificates to set up a chain of trust in your keystore. View the trusted CA and intermediate certificates to check that the displayed certificate fingerprints
match the expected ones.

```
$ keytool -v -printcert -file root.crt
$ keytool -v -printcert -file intermediate.crt
```

9. Import the CA's root certificate in the keystore and truststore. If there are other intermediate certificates, then import them using the same commands, giving them each different aliases in the key and trust stores.

```
$ keytool -importcert -v -trustcacerts -alias cacert \
  -keystore keystore -storepass changeit -file root.crt
$ keytool -importcert -v -trustcacerts -alias cacert -keystore truststore
\
  -storepass changeit -file root.crt
```

10. Import the server certificate signed by the CA into the keystore, which will replace the existing self-signed certificate. When prompted, type **yes** to trust the certificate.

```
$ keytool -importcert -v -trustcacerts -alias server-cert -keystore
keystore
  -storepass changeit -file signed.crt
```

11. Add the certificate to the truststore.

```
$ keytool -importcert -v -trustcacerts -alias server-cert \
  -keystore truststore -storepass changeit -file signed.crt
```

Create a Client Certificate

Client certificates can be used when stronger client authentication is desired but are not required for SSL connections to be established. There are two important considerations when using client certificates:

- If a client presents its own certificate to the server, the server must be configured to trust that certificate.
- If the client certificates will be used for LDAP authentication via SASL EXTERNAL, then the certificate must contain enough information to allow the server to associate it with

exactly one user entry. The requirements for this are dependent upon the certificate mapper configured for use in the server.

To create a PKCS#12 formatted client certificate with the Keytool utility, follow the steps in Create a Server Certificate and use the following command:

```
$ keytool -genkeypair \
  -dname "CN=server.example.com,ou=Certificate,O=Example Company,C=US"\
  -alias server-cert -keyalg rsa -keystore keystore.p12 -keypass changeit \
  -storepass changeit -storetype pkcs12 -validity 180 -noprompt
```

Configure the Key and Trust Manager Providers

UnboundID servers support the following trust and key managers:

- JKS Key Manager Provider and Trust Manager Provider
- PKCS#11 Key Manager Provider and Trust Manager Provider
- PKCS#12 Key Manager Provider and Trust Manager Provider

Note A blind trust manager provider can be used for evaluation purposes, but is not recommended for a production environment.

Perform the following steps to enable a key manager and trust manager and assign a connection handler. this can be done with `dsconfig` interactively, or manually.

1. Change location to the server root:

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

2. Create a text file containing the password for the certificate keystore. It is recommended that file permissions (or filesystem ACLs) be configured so that the file is only readable by the server user.

   ```
   $ echo 'changeit' > config/keystore.pin
   $ chmod 0400 keystore.pin
   ```

3. Use the `dsconfig` tool to enable the key manager provider.

   ```
   $ bin/dsconfig set-key-manager-provider-prop \
     --provider-name <JKS, PKCS11, or PKCS12>
     --set enabled:true
     --set key-store-file:/config/<Keystore.jks, keystore.p11 or
   keystore.p12>
     --set key-store-type:<JKS, PKCS11 or PKCS12>
     --set key-store-pin-file:/config/keystore.pin
   ```

4. Use `dsconfig` to enable the trust manager provider.

   ```
   $ bin/dsconfig set-trust-manager-provider-prop \
     --provider-name <JKS, PKCS11, or PKCS12> \
     --set enabled:true \
   ```

```
  --set trust-store-file:/config/<truststore.jks, truststore.p11, or
truststore.p12>
```

5. Use `dsconfig` to enable the LDAPS connection handler. Port 636 is typically reserved for LDAPS. If the certificate alias differs from the default "server-cert", use the `--set ssl-cert-nickname:<aliasname>` option to set it, or use the `--reset sslcert-nickname` option for the server to set the alias.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "LDAPS Connection Handler" \
  --set listen-port:1636 \
  --set enabled:true \
  --set ssl-cert-nickname:1 \
  --set key-manager-provider:<JKS, PKCS11, or PKCS12> \
  --set trust-manager-provider:<JKS, PKCS11, or PKCS12>
```

6. Test the listener port for SSL-based client connection on port 1636 to return the Root DSE. Type **yes** to trust the certificate.

```
$ bin/ldapsearch --port 1636 --useSSL --baseDN "" --searchScope base \
"(objectclass=*)"
```

```
The server is using the following certificate:
Subject DN: CN=179.13.201.1, OU=Server Certificate, O=Example Company,
L=Austin, ST=Texas, C=US
Issuer DN: EMAILADDRESS=whatever@example.com, CN=Cert Auth, OU=My
Certificate Authority, O=Example Company, L=Austin, ST=Texas, C=US
Validity: Fri Sep 25 15:21:10 CDT 2013 through Sat Sep 25 15:21:10 CDT
2014

Do you wish to trust this certificate and continue connecting to the
server?
Please enter 'yes' or 'no':yes
```

7. If necessary, disable the LDAP Connection Handler so that communication can only go through SSL.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "LDAP Connection Handler" \
  --set enabled:false
```

SSL and StartTLS Support

SSL and/or StartTLS are used to secure communication with clients and other components. The supported versions of SSL or StartTLS are determined by what the underlying JVM supports. The server will automatically look at the supported protocols and attempt to determine the best one to use.

When using Oracle Java SE 1.7, version TLSv1.2 is preferred by the server. A particular protocol can be specified by setting the `com.unboundid.util.SSLUtil.defaultSSLProtocol` property.

LDAP-over-SSL (LDAPS) Support

The server provides dedicated connection handlers for LDAPS connections. LDAPS enables the client and server to establish an SSL session before messages are transferred.

StartTLS Support

StartTLS adds SSL encryption to an existing plain text LDAP connection. The client opens an unencrypted TCP connection to the server and, after processing zero or more LDAP operations over that clear-text connection, sends a StartTLS extended request to the server to indicate that the client-server communication should be encrypted.

To require the use of SSL for client connections accepted by a connection handler, set use-ssl to true for that connection handler. To allow clients to use StartTLS on a connection handler, the administrator must configure that connection handler to allow StartTLS. Because SSL and StartTLS are mutually exclusive, you cannot enable both SSL and StartTLS for the same connection handler (although you can have some connection handlers configured to use SSL and others configured to use StartTLS).

Configure SSL

If SSL was not configured at install time, then it may be enabled at any time using the following process. These instructions assume that the certificate is available in a JKS-formatted keystore, but a similar process may be used for certificates available through other mechanisms like a PKCS#12 file or a PKCS#11 token.

Perform the following steps to configure SSL:

1. Change to the server root directory.

   ```
   $ cd /ds/UnboundID-Metrics-Engine
   ```

2. Create a text file containing the password for the certificate keystore. It is recommended that file permissions (or filesystem ACLs) be configured so that the file is only readable by the Metrics Engine user.

   ```
   $ echo 'changeit' > config/keystore.pin
   $ chmod 0400 config/keystore.pin
   ```

3. Run the `dsconfig` command in interactive mode (`bin/dsconfig`).
4. Enter the connection parameters when prompted.
5. On the Configuration Console main menu, enter o (lowercase letter "o") to change the configuration objects menu.
6. Choose the **Advanced** menu option.
7. On the Metrics Engine Configuration Console main menu, enter the number corresponding to the **Key Manager Provider**.

8. On the Key Manager Provider management menu, select the option to view and edit an existing key manager.
9. On the Key Manager Provider menu, enter the option for **JKS**.
10. Make any necessary changes to the JKS key manager provider for the keystore. The provider must be enabled, and the locations of the key-store-file and key-store-pin-file must be set.

```
>>>> Configure the properties of the File Based Key Manager Provider

Property Value(s)
----------------------------------------
1) description -
2) enabled true
3) key-store-file config/keystore
4) key-store-type              JKS
5) key-store-pin               -
6) key-store-pin-file config/keystore.pin
7) private-key-pin             -
8) private-key-pin-file        -
```

11. Type **f** to save and apply the changes.
12. Return to the Advanced main menu, and enter the number corresponding to **Trust Manager Provider**.
13. On the Trust Manager Provider management menu, enter the option to view and edit an existing trust manager provider.
14. On the Trust Manager Provider menu, enter the option for **JKS**.
15. Ensure that the JKS trust manager provider is enabled and that the trust-store-file property has a value that reflects the path to the truststore file.
16. Type **f** to save and apply the changes.
17. Return to the Configuration (Standard) main menu, enter the number corresponding to the **Connection Handler** option.
18. On the Connection Handler management menu, enter the option to view and edit and existing connection handler.
19. On the Connection Handler menu, enter the option for **LDAPS Connection Handler**.
20. On the LDAP Connection Handler menu, make sure that the handler is enabled, the listen-port property reflects the port on which to listen for SSL-based connections, and the ssl-cert-nickname property reflects the alias for the target certificate in the selected key-store.
21. Type **f** to save and apply the changes.

22. Verify that the server is properly configured to accept SSL-based client connections using an LDAP-based tool like ldapsearch. For example:

```
$ bin/ldapsearch --port 1636 --useSSL --baseDN "" \
--searchScope base "(objectclass=*)"
```

```
The server is using the following certificate:
  Subject DN: CN=179.13.201.1, OU=Metrics Engine
  Certificate, O=Example Company, L=Austin, ST=Texas,
  C=US Issuer DN: EMAILADDRESS=whatever@example.com,
  CN=Cert Auth, OU=My Certificate Authority, O=Example
  Company, L=Austin, ST=Texas, C=US
  Validity: Fri Sep 25 15:21:10 CDT 2011 through Sat Sep 25 15:21:10 CDT
2012
Do you wish to trust this certificate and continue connecting to the
server?
Please enter 'yes' or 'no':yes
```

If necessary, disable the LDAP connection handler so only the LDAPS connection handler will accept connections.

Configure StartTLS

The StartTLS extended operation is used to initiate a TLS-secured communication channel over a clear-text connection, such as an insecure LDAP connection. StartTLS provides a way to use a single connection handler for both secure and insecure communication, rather than requiring a dedicated connection handler for secure communication.

1. Use `dsconfig` to configure the Connection Handler to allow StartTLS. The `allow-starttls` property cannot be set if SSL is enabled. The connection handler must also be configured with a key manager provider and a trust manager provider.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "LDAP Connection Handler" \
  --set allow-start-tls:true \
  --set key-manager-provider:JKS \
  --set trust-manager-provider:JKS
```

2. Use `ldapsearch` to test StartTLS.

```
$ bin/ldapsearch -p 1389 --useStartTLS -b "" -s base "(objectclass=*)"
```

```
The server is using the following certificate:
   Subject DN: CN=Server Cert, OU=Metrics Engine Certificate,
     O=Example Company, L=Austin, ST=Texas, C=US
   Issuer DN: EMAILADDRESS=whatever@example.com, CN=Cert Auth,
     OU=My Certificate Authority, O=Example Company, L=Austin, ST=Texas,
C=US
   Validity: Thu Oct 29 10:29:59 CDT 2013 through Fri Oct 29 10:29:59 CDT
2014

Do you wish to trust this certificate and continue connecting to the
```

```
server?
Please enter 'yes' or 'no':yes
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 6fa8f196-d112-40b4-b8d8-93d6d44d59ea
```

Authentication Mechanisms

UnboundID servers support the following authentication mechanisms:

Simple Authentication – Simple authentication allows a client to identify itself with a server using the DN and password of the target user. Because the password is provided in the clear, simple authentication is inherently insecure unless the client communication is encrypted using a mechanism like SSL or StartTLS.

If both the DN and password of a simple bind request are empty (zero-length strings), the server will process it as an anonymous bind. This will have no effect if the client is not already authenticated, but it can be used to destroy any previous authentication session and revert the connection to an unauthenticated state as if no bind had ever been performed on that connection.

SASL Authentication – SASL (Simple Authentication and Security Layer, defined in RFC 4422) provides an extensible framework that can be used to add support for a range of authentication and authorization mechanisms. Several common SASL mechanisms are supported.

Configure a SASL Mechanism Handler

The `dsconfig` utility enables configuration of the following SASL mechanism handlers. Configuration can be performed through the `dsconfig` interactive menu or from the command line.

ANONYMOUS – Does not perform any authentication, but can be enabled for clients to include a trace string to identify the purpose of a connection.

CRAM-MD5 – Performs password-based authentication through an MD5 digest. The client sends a bind request to the server. The server responds with a randomly-generated challenge to protect against replay attacks. The client responds with an answer to the challenge, a clear-text password, and an authentication ID. The server encodes the password and requires that any clients have a password policy that supports two-way, reversible encryption.

By default, SASL DIGEST-MD5 uses the Exact Match Identity Mapper, which returns a success result if the authorization ID is an exact match for the value of the `uid` attribute. Other identity mappers, such as the Regular Expression Identity Mapper or a custom Identity Mapper written using the UnboundID Server SDK, can also be used.

DIGEST-MD5 – Provides authentication through a stronger MD5 digest that does not expose a clear-text password. The client sends a bind request with credentials to the server. The server sends the client a response with a set of authentication options and a special token. The client sends an encrypted response with the chosen authentication method. The server then validates the client's response. This is the required authentication mechanism for LDAP v3 servers.

EXTERNAL – Allows a client to authenticate using information about the client, which is available to the server, but is not directly provided over LDAP. On the server, SASL EXTERNAL requires the use of a client certificate provided during SSL or StartTLS negotiation. This does not require the use of passwords, although its use on a broad scale is generally only feasible in environments with a PKI deployment.

GSSAPI – Provides authentication for LDAP clients using Kerberos V. User credentials are stored in the Kerberos key distribution center (KDC) rather than the UnboundID server. When an LDAP client attempts to authenticate with the server, a three-way exchange occurs that allows the client to verify its identity to the server through the KDC.

UnboundID's support for GSSAPI is based on the Java Authentication and Authorization Service (JAAS). By default, the server automatically generates a JAAS configuration that should be appropriate for most use cases. For more complex deployments, a custom JAAS configuration can be supplied.

UnboundID servers support GSSAPI only for authenticating clients, not for securing their communication with the server.

PLAIN – Performs password-based authentication with an authentication ID, clear-text password, and optional authorization ID.

UNBOUNDID-TOTP – Provides a proprietary multifactor authentication mechanism that allows the server to use the Time-based One-Time Password (TOTP) algorithm, specified in RFC 6238. The TOTP algorithm is an extension of the Hash-based Message Authentication Code One-Time Password (HTOP) algorithm, specified in RFC 4226. The TOTP algorithm computes a temporary code using the current time and a secret key that is shared between the client application and the server.

UNBOUNDID-TOTP SASL issues a bind request that includes at least an authentication ID and a TOTP code, but may also include an authorization ID and/or a static password. The server first uses the authentication ID to identify the user that is authenticating and then retrieves the shared secret from the user's entry (stored as a base32-encoded value in the `ds-auth-totp-sharedsecret` operational attribute) and uses it with the current time to generate a TOTP code. If that matches the code that the user entered, then that confirms that the client knows the shared secret. If a static password was also provided, then the server will confirm that it matches what is stored in the `userPassword` attribute (or that specified by the password policy). By default, the server will require the client to provide a static password.

The Commercial Edition of the LDAP SDK for Java provides the necessary client-side support for the UNBOUNDID-TOTP SASL mechanism, and provides a `com.unboundid.ldap.sdk.unboundidds.OneTimePassword` class to generate HOTP and TOTP codes for testing purposes.

UNBOUNDID-DELIVERED-OTP – Provides two-factor authentication, which uses one-time passwords (OTPs) that are delivered to the end user through some out-of-band mechanism. The server provides support for e-mail (through an external SMTP external server), SMS (through the Twilio web service), and custom delivery mechanisms with the Server SDK.

The process for authenticating using this new mechanism involves two steps:

- The client sends a "deliver one-time password" extended request to the server. This request includes an authentication ID, the user's static password, and an optional set of allowed delivery mechanisms. If successful, the server generates a one-time password, stores it in the user's entry, and sends it to the user through one of the allowed mechanisms.
- Once the user has received the one-time password, the client should perform an UNBOUNDID-DELIVERED-OTP SASL bind (which may be on the same connection or a different connection used to send the "deliver one-time password" extended operation). The credentials for this SASL mechanism include an authentication ID to identify the user, an optional authorization ID (if operations performed by the client should be authorized as a different user), and the one-time password.

Unlike UNBOUNDID-TOTP SASL, there is no need to have a shared secret between the client and the server, or any special client-side software to generate the one-time password, or a need to worry about whether the client and server clocks are roughly in sync.

Configure SASL ANONYMOUS Mechanism

The LDAP client tools provided with UnboundID servers support the use of SASL ANONYMOUS. The optional "trace" SASL option can be used to specify the trace string to include in the bind request.

Perform the following steps to configure SASL ANONYMOUS:

1. Use `dsconfig` to enable the SASL ANONYMOUS mechanism.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
  --handler-name ANONYMOUS --set enabled:true
```

2. Use `ldapsearch` to view the root DSE and enter a trace string in the access log.

```
$ bin/ldapsearch --port 1389 --saslOption mech=ANONYMOUS \
  --saslOption "trace=debug trace string" --baseDN "" \
  --searchScope base "(objectclass=*)"
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 59bab79d-4429-49c8-8a88-c74a86792f26
```

3. View the access log using a text editor in the `/ds/UnboundID-<server>/logs` folder.

```
[26/Oct/2011:16:06:33 -0500] BIND RESULT conn=2 op=0 msgID=1 resultCode=0
additionalInfo="trace='debug trace string'" etime=345.663
clientConnectionPolicy="default"
```

Configure SASL PLAIN Mechanism

LDAP clients can use SASL PLAIN with the following SASL options:

`authid` – Specifies the authentication ID to use for the bind. This must be provided.

`authzid`– Specifies an optional alternate authorization ID to use for the bind.

Perform the following steps to configure SASL PLAIN:

1. Use `dsconfig` to enable the SASL PLAIN mechanism.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
--handler-name PLAIN --set enabled:true
```

2. Use `ldapsearch` to view the root DSE using the authentication ID (`authid`) with the username jdoe. Enter a password for the authentication ID.

```
$ bin/ldapsearch --port 1389 --saslOption mech=PLAIN \
--saslOption "authid=u:jdoe" --baseDN "" \
--searchScope base "(objectclass=*)"
Password for user 'u:jdoe':
```

Or specify the full DN of the user:

```
$ bin/ldapsearch --port 1389 --saslOption mech=PLAIN \
--saslOption "authid=dn:uid=jdoe,ou=People,dc=example,dc=com" \
--baseDN "" --searchScope base "(objectclass=*)"
Password for user 'dn:uid=jdoe,ou=People,dc=example,dc=com':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 59bab79d-4429-49c8-8a88-c74a86792f26
```

Configure SASL CRAM-MD5 Mechanism

CRAM-MD5 requires an authentication ID (`authid`) from the client to identify the authenticating user. The format of that authentication ID can be either:

- `dn:` followed by the distinguished name of the target user (or just `dn:` to perform an anonymous bind).
- `u:` followed by a username.

If using `u:`, an identity mapper is used to identify the target user based on that username.

Perform the following steps to configure CRAM-MD5:

1. Use `dsconfig` to enable the SASL CRAM-MD5 mechanism if it is disabled. By default, the CRAM-MD5 mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
--handler-name CRAM-MD5 --set enabled:true
```

2. For this example, create a password policy for CRAM-MD5 using a reversible password storage scheme, like 3DES.

```
$ bin/dsconfig create-password-policy \
--policy-name "Test UserPassword Policy" \
--set password-attribute:userpassword \
--set default-password-storage-scheme:3DES
```

3. Use `ldapmodify` to add the `ds-pwp-password-policy-dn` attribute to an entry to indicate the Test UserPassword Policy should be used for that entry. When finished, press CTRL-D to process the modify operation.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=Test UserPassword Policy,cn=Password
Policies,cn=config
```

```
Processing MODIFY request for uid=jdoe,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=jdoe,ou=People,dc=example,dc=com
```

4. Use `ldapmodify` to change the `userPassword` to a reversible password storage scheme. The password storage scheme is specified in the user's password policy.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: secret
```

5. Use `ldapsearch` to view the root DSE using the authentication ID (`authid`) option with the username jdoe. Enter a password for the user.

```
$ bin/ldapsearch --port 1389 --saslOption mech=CRAM-MD5 \
--saslOption "authid=u:jdoe" --baseDN "" --searchScope base "
(objectclass=*)"
Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 50567aa3-acd2-4106-a077-37a092275363
```

Configure SASL DIGEST-MD5 Mechanism

DIGEST-MD5 requires an authentication ID (`authid`) from the client to identify the authenticating user. The format of that authentication ID can be either:

- `dn:` followed by the distinguished name of the target user (or just `dn:` to perform an anonymous bind).
- `u:` followed by a username. If using `u:`, an identity mapper is used to identify the target user based on that username.

The client may also include the following properties:

- `authzID` – Specifies an optional authorization ID that should be used for operations processed on the connection.

- `realm` – The realm in which the authentication should be processed. This may or may not be required, based on the server configuration.
- `digest-uri` – The digest URI that should be used for the bind. It should generally be "ldap://" followed by the fully-qualified address for the Metrics Engine. If this is not provided, then a value will be generated.
- `qop` – The quality of protection to use for the bind request. Only `auth` is supported (indicating that the DIGEST-MD5 bind should only be used for authentication and should not provide any subsequent integrity or confidentiality protection for the connection), and if no value is provided then auth will be assumed.

Perform the following steps to configure CRAM-MD5:

1. Use `dsconfig` to enable the SASL DIGEST-MD5 mechanism if it is disabled. By default, the DIGEST-MD5 mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
--handler-name DIGEST-MD5 --set enabled:true
```

2. For this example, create a password policy using a reversible password storage scheme, like 3DES.

```
$ bin/dsconfig create-password-policy \
--policy-name "Test UserPassword Policy" \
--set password-attribute:userpassword \
--set default-password-storage-scheme:3DES
```

3. Use `ldapmodify` to add the `ds-pwp-password-policy-dn` attribute to an entry to indicate the Test UserPassword Policy should be used for that entry. When finished, press CTRL-D to process the modify operation.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=Test UserPassword Policy,cn=Password
Policies,cn=config
```

```
Processing MODIFY request for uid=jdoe,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=jdoe,ou=People,dc=example,dc=com
```

4. Use `ldapmodify` to change the `userPassword` to a reversible password storage scheme. The password storage scheme is specified in the user's password policy.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: secret
```

5. Use `ldapsearch` to view the root DSE using the authentication ID with the username jdoe. Enter a password for the authentication ID.

```
$ bin/ldapsearch --port 1389 --saslOption mech=DIGEST-MD5 \
--saslOption "authid=u:jdoe" --baseDN "" \
--searchScope base "(objectclass=*)"
Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

6. The user identified by the authentication ID requires the proxied-auth privilege to allow it to perform operations as another user.

```
$ bin/ldapmodify

dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modifyadd: ds-privilege-name
ds-privilege-name: proxied-auth
```

5. Use ldapsearch with the `authid` (required) and `authzid` option to test the mechanism.

```
$ bin/ldapsearch --port 1389 --saslOption mech=DIGEST-MD5 \
--saslOption authid=u:jdoe \
--saslOption authzid=dn:uid=admin,dc=example,dc=com \
--base "" --searchScope base "(objectclass=*)"
Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

Configure SASL DIGEST-MD5 Mechanism

DIGEST-MD5 requires an authentication ID (`authid`) from the client to identify the authenticating user. The format of that authentication ID can be either:

- `dn:` followed by the distinguished name of the target user (or just `dn:` to perform an anonymous bind).
- `u:` followed by a username. If using `u:`, an identity mapper is used to identify the target user based on that username.

The client may also include the following properties:

- `authzID` – Specifies an optional authorization ID that should be used for operations processed on the connection.
- `realm` – The realm in which the authentication should be processed. This may or may not be required, based on the server configuration.

- `digest-uri` – The digest URI that should be used for the bind. It should generally be "ldap://" followed by the fully-qualified address for the Metrics Engine. If this is not provided, then a value will be generated.
- `qop` – The quality of protection to use for the bind request. Only `auth` is supported (indicating that the DIGEST-MD5 bind should only be used for authentication and should not provide any subsequent integrity or confidentiality protection for the connection), and if no value is provided then auth will be assumed.

Perform the following steps to configure CRAM-MD5:

1. Use `dsconfig` to enable the SASL DIGEST-MD5 mechanism if it is disabled. By default, the DIGEST-MD5 mechanism is enabled.

```
$ bin/dsconfig set-sasl-mechanism-handler-prop \
--handler-name DIGEST-MD5 --set enabled:true
```

2. For this example, create a password policy using a reversible password storage scheme, like 3DES.

```
$ bin/dsconfig create-password-policy \
--policy-name "Test UserPassword Policy" \
--set password-attribute:userpassword \
--set default-password-storage-scheme:3DES
```

3. Use `ldapmodify` to add the `ds-pwp-password-policy-dn` attribute to an entry to indicate the Test UserPassword Policy should be used for that entry. When finished, press CTRL-D to process the modify operation.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: ds-pwp-password-policy-dn
ds-pwp-password-policy-dn: cn=Test UserPassword Policy,cn=Password
Policies,cn=config
```

```
Processing MODIFY request for uid=jdoe,ou=People,dc=example,dc=com
MODIFY operation successful for DN uid=jdoe,ou=People,dc=example,dc=com
```

4. Use `ldapmodify` to change the `userPassword` to a reversible password storage scheme. The password storage scheme is specified in the user's password policy.

```
$ bin/ldapmodify
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
replace: userPassword
userPassword: secret
```

5. Use `ldapsearch` to view the root DSE using the authentication ID with the username jdoe. Enter a password for the authentication ID.

```
$ bin/ldapsearch --port 1389 --saslOption mech=DIGEST-MD5 \
--saslOption "authid=u:jdoe" --baseDN "" \
```

```
--searchScope base "(objectclass=*)"
Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

6. The user identified by the authentication ID requires the proxied-auth privilege to allow it to perform operations as another user.

```
$ bin/ldapmodify

dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modifyadd: ds-privilege-name
ds-privilege-name: proxied-auth
```

5. Use ldapsearch with the `authid` (required) and `authzid` option to test the mechanism.

```
$ bin/ldapsearch --port 1389 --saslOption mech=DIGEST-MD5 \
--saslOption authid=u:jdoe \
--saslOption authzid=dn:uid=admin,dc=example,dc=com \
--base "" --searchScope base "(objectclass=*)"
Password for user 'u:jdoe':
```

```
dn:
objectClass: ds-root-dse
objectClass: top
startupUUID: 2188e4d4-c2bb-4ab9-8e1c-848e0168c9de
```

Configure SASL GSSAPI Mechanism

While the GSSAPI specification includes a provision for protecting client-server communication, UnboundID servers currently support GSSAPI only for the purpose of authenticating clients.

Kerberos Configuration Considerations

To implement GSSAPI authentication, a Kerberos V deployment must be configured. The Kerberos deployment should take the following into consideration:

- It is recommended that the KDC be configured to use "aes128-cts" as the TKT and TGS encryption type, which is supported by all Java VMs. In Kerberos environments using the MIT libraries, make sure that the following lines are present in the [libdefaults] section of the `/etc/krb.conf` configuration file on the KDC system:

```
default_tkt_enctypes = aes128-cts
default_tgs_enctypes = aes128-cts
permitted_enctypes = aes128-cts
```

- When a client uses Kerberos to authenticate to a server, the addresses of the target server and the KDC are used in cryptographic operations. Make sure that all systems

agree on the addresses of the server and KDC systems. Make sure that DNS is configured so that the primary addresses for the KDC and server are addresses that clients will use to communicate with them.

- Kerberos authentication is time-sensitive. If system clocks are not synchronized, authentication may fail. Use NTP or some other form of time synchronization for all KDC, server, and client systems.

To authenticate itself to the Kerberos environment, the KDC should include both host and service principals for all servers. The host principal is in the form `host/directory.example.com`, and the service principal should generally be `ldap/directory.example.com`. In an MIT Kerberos environment, the kadmin utility can be used to create these principals, as follows:

```
# /usr/sbin/kadmin -p kws/admin
Authenticating as principal kws/admin with password.
Password for kws/admin@EXAMPLE.COM:
kadmin: add_principal -randkey host/directory.example.com
WARNING: no policy specified for host/directory.example.com@EXAMPLE.COM;
  defaulting to no policy
Principal "host/directory.example.com@EXAMPLE.COM" created.
kadmin: ktadd host/directory.example.com
Entry for principal host/directory.example.com with kvno 3, encryption type
AES-128
  CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/kr-
b5/krb5.keytab.
kadmin: add_principal -randkey ldap/directory.example.com
WARNING: no policy specified for ldap/directory.example.com@EXAMPLE.COM;
  defaulting to no policy
Principal "ldap/directory.example.com@EXAMPLE.COM" created.
kadmin: quit
```

On each server, the service principal for that instance must be exported to a keytab file, using a command such as:

```
# /usr/sbin/kadmin -p kws/admin
Authenticating as principal kws/admin with password.
Password for kws/admin@EXAMPLE.COM:
kadmin: ktadd -k /ds/UnboundID-<server>/config/server.keytab ldap/
directory.example.com
Entry for principal ldap/directory.example.com with kvno 4, encryption type
AES-128
  CTS mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/ds/UnboundID-Met-
rics-Engine/
config/
  server.keytab.
kadmin: quit
```

Because this file contains the credentials that the server will use to authenticate to the KDC, it is strongly recommended that appropriate protection be taken to ensure that it is only accessible to the server itself.

GSSAPI Mechanism Handler Options

The GSSAPI SASL mechanism handler provides the following configuration options:

`enabled` – Indicates whether the GSSAPI SASL mechanism handler is enabled for use in the server. By default, it is disabled.

`kdc-address` – Specifies the address that the server should use to communicate with the KDC. If this is not specified, the server will attempt to determine it from the underlying system configuration.

`server-fqdn` – Specifies the fully-qualified domain name that clients will use to communicate with the server. If this is not specified, the server will attempt to determine it from the underlying system configuration.

`realm` – Specifies the Kerberos realm that clients will use. If this is not specified, the server will attempt to determine it from the underlying system configuration.

`kerberos-service-principal` – Specifies the service principal that the server will use to authenticate with the KDC. If this is not specified, the service principal is "ldap/" followed by the fully-qualified server address.

`keytab` – Specifies the path to the keytab file that holds the credentials for the Kerberos service principal that the server uses to authenticate with the KDC. If this is not specified, the server will use the system-wide keytab.

`identify-mapper` – Specifies the identify mapper that the server will use to map a client's Kerberos principal to the entry of the corresponding user account in the server. In the default configuration, the server uses a regular expression identity mapper that looks for an entry with a `uid` value equal to the username portion of the Kerberos principal. For example, for a Kerberos principal of `jdoe@EXAMPLE.COM`, the identity mapper will perform an internal search with a filter of (`uid=jdoe`).

`enable-debug` – Indicates whether the server should write debugging information about Kerberos-related processing (including JAAS processing) that the server performs. If enabled, this information will be written to standard error in the `logs/server.out` log file.

`jaas-config` file – Specifies the path to a JAAS configuration file that the server should use. If this is not specified, the server will generate a JAAS configuration file based on the values of the other configuration properties. It is recommended that this only be used in extraordinary circumstances in which the server-generated JAAS configuration is not acceptable.

Configure SASL UNBOUNDID-TOTP Mechanism

Perform the following steps to configure the UNBOUNDID-TOTP mechanism:

1. Configure the server so that `ds-auth-totp-shared-secret` is a sensitive attribute that can only be set over a secure connection and cannot ever be retrieved from the server.

Create the sensitive attribute and reference it from the global configuration using `dsconfig`.

```
$ bin/dsconfig create-sensitive-attribute \
--attribute-name ds-auth-totp-shared-secret \
--set attribute-type:ds-auth-totp-shared-secret \
--set allow-in-returned-entries:suppress \
--set allow-in-filter:reject \
--set allow-in-compare:reject \
--set allow-in-add:secure-only \
--set allow-in-modify:secure-only

$ bin/dsconfig set-global-configuration-prop \
--add sensitive-attribute:ds-auth-totp-shared-secret
```

2. Update a user entry so that it contains a `ds-auth-totp-shared-secret` attribute with a value that holds the base32-encoded shared secret that will be used for TOTP authentication. This should be done over a secure connection (SSL or StartTLS). There is no maximum limit for the `ds-auth-totp-shared-secret` string, but there is a minimum length of 16 base32-encoded characters.

```
dn: uid=user.0,ou=People,dc=example,dc=com
changetype: modify
add: ds-auth-totp-shared-secret
ds-auth-totp-shared-secret: ONSWG4TFORRW6ZDF
```

3. Use `ldapsearch` to test the configuration.

```
$ bin/ldapsearch --saslOption mech=UNBOUNDID-TOTP \
--saslOption authID=u:user.0 \
--saslOption totpPassword=628094 \
--bindPassword password \
--baseDN "" \
--searchScope base \
"(objectClass=*)"
```

Configure SASL UNBOUNDID-DELIVERED-OTP Mechanism

Perform the following steps to configure the UNBOUNDID-DELIVERED-OTP mechanism:

1. Add support for one or more OTP delivery mechanisms. The following commands enable an SMTP external server, associate it with the global configuration, and create the delivery mechanism.

```
$ bin/dsconfig create-external-server \
--server-name "Intranet SMTP Server" \
--type smtp \
--set server-host-name:server.example.com
```

```
$ bin/dsconfig set-global-configuration-prop \
--add "smtp-server:Intranet SMTP Server"
```

```
$ bin/dsconfig create-otp-delivery-mechanism \
--mechanism-name E-Mail \
--type email \
--set enabled:true \
--set 'sender-address:otp@example.com' \
--set "email-address-attribute-type:mail" \
--set "message-subject:Your one-time password" \
--set "message-text-before-otp:Your one-time password: "
```

2. With a Twilio account, configure the server to deliver one-time passwords over SMS.

```
dsconfig create-otp-delivery-mechanism \
--mechanism-name SMS \
--type twilio \
--set enabled:true
--set twilio-account-sid:xxxxx \
--set twilio-auth-token:xxxxx \
--set "sender-phone-number:xxxxx" \
--set phone-number-attribute-type:mobile \
--set "message-text-before-otp:Your one-time password: "
```

3. With OTP delivery mechanisms established, configure the extended operation handler.

```
$ bin/dsconfig create-extended-operation-handler \
--handler-name "Deliver One-Time Password" \
--type deliver-otp \
--set enabled:true \
--set "identity-mapper:Exact Match" \
--set "password-generator:One-Time Password Generator" \
--set default-otp-delivery-mechanism:SMS \
--set default-otp-delivery-mechanism:E-Mail
```

4. Configure the SASL mechanism handler.

```
$ bin/dsconfig create-sasl-mechanism-handler \
--handler-name UNBOUNDID-DELIVERED-OTP \
--type unboundid-delivered-otp \
--set enabled:true \
--set "identity-mapper:Exact Match" \
--set "otp-validity-duration:5 minutes"
```

5. Make sure the server contains a user account with the account needed to deliver the one-time password to the user (i.e., a valid email address or mobile number).

6. Next, use the deliver one-time password extended operation to have the server generate and send a one-time password to the user. The Commercial Edition of UnboundID LDAP SDK contains support for the extended request and response needed to do this. In actual production deployments, you can create a web form to allow the user to enter the information and click a button. The server comes with a new deliver-one-time-password command-line tool that can achieve the same result.

```
$ bin/deliver-one-time-password \
--userName jdoe \
```

```
--promptForBindPassword \
--deliveryMechanism SMS
Enter the static password for the user:

Successfully delivered a one-time password via mechanism 'SMS' to '123-
456-7890'
```

If processed successfully, a text message is recieved:

```
Your one-time password: 123456
```

7. Authenticate to the server using the UNBOUNDID-DELIVERED-OTP SASL mechanism. The Commercial Edition of the LDAP SDK can help you accomplish this so that the user sees an interface. Or, you can use ldapsearch or some other tool to accomplish the same result.

```
$ bin/ldapsearch \
-o mech=UNBOUNDID-DELIVERED-OTP \
-o authID=u:jdoe \
-o otp=123456 \
-b '' \
-s base '(objectClass=*)' \
ds-supported-otp-delivery-mechanism
```

The search returns:

```
dn:
ds-supported-otp-delivery-mechanism: E-Mail
ds-supported-otp-delivery-mechanism: SMS
```

Configure Pass-Through Authentication

Pass-through authentication (PTA) is a mechanism by which one Metrics Engine receives the bind request and can consult another Metrics Engine to authenticate the bind request. Implement this functionality by configuring a PTA plug-in that enables the server to accept simple password-based bind operations.

Perform the following steps to configure PTA:

1. Use `dsconfig` to define external servers to perform the authentication. The bind DN is set to `uid=pass-throughuser, dc=example,dc=com`, which is used to bind to the target LDAP server for simple authentication. The `verify-credentials-method` property ensures that a single set of connections for processing binds and all other types of operations is in place without changing the identity of the associated connection. Multiple external servers can be configured in case one becomes unavailable.

```
$ bin/dsconfig create-external-server \
  --server-name "ds-with-pw-1.example.com:389" \
  --type unboundid-metrics-engine \
  --set server-host-name:ds-with-pw-1.example.com \
  --set server-port:389 \
  --set "bind-dn:uid=pass-through-user,dc=example,dc=com" \
```

```
  --set authentication-method:simple \
  --set verify-credentials-method:retain-identity-control
```

2. Create an instance of the PTA plug-in that will use the external server(s). The server will first try to process a local bind as the target user (`try-local-bind:true`). The `try-local-bind:true` with `override-local-password:true` means that if the local bind fails, it will try sending the request to `ds-with-pw-1.example.com:389` or another server, if configured (`server-access-mode:round-robin`). If the bind succeeds against the remote server, the local entry is updated to store the password that was used (`update-local-password:true`). The number of connections to initially establish to the LDAP external server is set to 10. The maximum number of connections maintained to the LDAP external server is 10.

```
$ bin/dsconfig create-plugin \
  --plugin-name "Pass-Through Authentication" \
  --type pass-through-authentication \
  --set enabled:true \
  --set server:ds-with-pw-1.example.com:389 \
  --set server:ds-with-pw-2.example.com:389 \
  --set try-local-bind:true \
  --set override-local-password:true \
  --set update-local-password:true \
  --set server-access-mode:round-robin \
  --set initial-connections:10 \
  --set max-connections:10
```

Note

The `try-local-bind` property works with the `override-local-password` property. If `try-local-bind` is true and `override-local-password` is set to its default value of false, then the server attempts a local bind first. If it fails because no password is set, then it will forward the bind request to a remote server. If the password was set but still fails, the server will not send the request to the remote server.

If `try-local-bind` is true and `override-local-password` is true, then a local bind will be attempted. The server will forward the request to the remote server if the local bind fails for any reason.

Add Attributes to Restrict Authentication

The UnboundID servers provide a number of operational attributes that can be added to user entries to restrict the way those users can authenticate and the circumstances under which they can be used for proxied authorization. The operational attributes are as follows:

- `ds-auth-allowed-address` – Indicates that the user should only be allowed to authenticate from a specified set of client systems. Values are listed as individual IP addresses, IP address patterns (using wildcards, CIDR notation, or subnet mask notation), individual DNS addresses, or DNS address patterns (using wildcards). If no values are listed in a user entry, no restrictions are enforced.

- `ds-auth-allowed-authentication-type` – Indicates that the user should only be allowed to authenticate in certain ways. Values include `simple` , or `sasl <mech>`. If no values are listed in a user entry, no restrictions are enforced.
- `ds-auth-require-secure-authentication` – Specifies whether the user should be required to authenticate securely. If this attribute is present with a value of `true`, the user can only authenticate over a secure connection or with a mechanism that does not expose user credentials (CRAM-MD5, DIGEST-MD5, and GSSAPI SASL).
- `ds-auth-require-secure-connection` – Specifies whether the user is required to communicate with the server over a secure connection. If this attribute is present in a user entry with a value of `true`, the user can only communicate with the server over a secure connection (SSL or StartTLS).
- `ds-auth-is-proxyable` – Indicates whether the user can be used as the target of proxied authorization (using proxied authorization v1 or v2 control, intermediate client control, or a SASL mechanism that allows an alternate authorization identity). If this attribute is present in a user entry with a value of `required`, the user is not allowed to authenticate directly with the server but only referenced by proxied authorization. If this attribute is present with a value of `prohibited`, the user is not allowed to be the target of proxied authorization but can only authenticate directly with the server. If this attribute is present with a value of `allowed`, or if it is not present, the user can authenticate directly or be the target of proxied authorization.
- `ds-auth-is-proxyable-by` – Restricts the set of accounts that may target the user for proxied authorization. If this attribute is present in a user's entry, its values must be the distinguished names of the users that can target the user for proxied authorization (if proxied authorization is allowed). If it is absent from the user's entry, then any account with appropriate rights may target the user via proxied authorization.

Configure Certificate Mappers

SASL EXTERNAL requires that a certificate mapper be configured in the server. The certificate mapper is used to identify the entry for the user to whom the certificate belongs. UnboundID servers support a number of certificate mapping options including:

- **Subject Equals DN** – Specifies that the subject of the certificate exactly match the distinguished name of the associated user entry. This option is not often practical as certificate subjects (cn=jdoe,ou=Client Cert,o=Example Company,c=Austin,st=Texas,c=US) are not typically in the same form as an entry (cn=-=jdoe,ou=People,o=Example Company, or uid=jdoe,ou=People,dc=example,dc=com).

- **Fingerprint** – Specifies that the user's entry contain an attribute (`ds-certficate-fingerprint` by default), with values SHA-1 or MD5 fingerprints of the certificate(s) that they can use to authenticate. This attribute must be indexed for equality.
- **Subject Attribute to User Attribute** – Used to build a search filter to find the appropriate user entry based on information contained in the certificate subject. For example, the default configuration expects the cn value from the certificate subject to match the cn value of the user's entry, and the e value from the certificate subject to match the mail value of the user's entry.
- **Subject DN to User Attribute** – Expects the user's entry to contain an attribute (`ds-certificate-subject-dn` by default), whose values are the subjects of the certificate (s) that they can use to authenticate. This multi-valued attribute can contain the subjects of multiple certificates. The attribute must be indexed for equality.

Configure the Subject Equals DN Certificate Mapper

The Subject Equals DN Certificate Mapper is the default mapping option for the SASL EXTERNAL mechanism. The mapper requires that the subject of the client certificate exactly match the distinguished name of the corresponding user entry. The mapper, however, is only practical if the certificate subject has the same format as the server's entries.

Perform the following steps to configure the Subject Equals DN Certificate Mapper:

Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
--handler-name EXTERNAL \
--set "certificate-mapper:Subject Equals DN"
```

Configure the Fingerprint Certificate Mapper

The Fingerprint Mapper causes the server to compute an MD5 or SHA-1 fingerprint of the certificate presented by the client and performs a search to find that fingerprint value in a user's entry (ds-certificate-fingerprint by default). The ds-certificate-fingerprint attribute can be added to the user's entry together with the ds-certificate-user auxiliary object class. For multiple certificates, the attribute can have separate values for each of the acceptable certificates. If you decide to use this attribute, you must index the attribute as it is not indexed by default.

The following example will use this certificate:

```
Alias name: client-cert
Creation date: Oct 29, 2011
Entry type: PrivateKeyEntry

Certificate chain length: 1 Certificate[1]:
Owner: CN=jdoe, OU=Client Cert, O=Example Company, L=Austin, ST=Texas, C=US
Issuer: EMAILADDRESS=whatever@example.com, CN=Cert Auth, OU=My Certificate
```

```
Authority,
O=Example Company, L=Austin, ST=Texas, C=US
Serial number: e19cb2838441dbcd
Valid from: Thu Oct 29 13:07:10 CDT 2011 until: Fri Oct 29 13:07:10 CDT 2012
Certificate fingerprints:
   MD5: 40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   SHA1: 2A:89:71:06:1A:F5:DA:FF:51:7B:3D:2D:07:2E:33:BE:C6:5D:97:13
   Signature algorithm name: SHA1withRSA
   Version: 1
```

Perform the following steps to configure the Fingerprint Certificate Mapper:

1. Create an LDIF file to add the `ds-certificate-user object class` and `ds-certificate-fingerprint` attribute to the target user's entry.

   ```
   dn: uid=jdoe,ou=People,dc=example,dc=com
   changetype: modify
   add: objectClass
   objectClass: ds-certificate-user
   -
   add: ds-certificate-fingerprint
   ds-certificate-fingerprint:
   40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

2. Then, apply the change to the entry using ldapmodify:

   ```
   $ bin/ldapmodify --filename add-cert-attr.ldif
   dn: uid=jdoe,ou=People,dc=example,dc=com
   ds-certificate-fingerprint:40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

3. Check that the attribute was added to the entry using `ldapsearch`.

   ```
   $ bin/ldapsearch --baseDN dc=example,dc=com "(uid=jdoe)" \
   ds-certificate-fingerprint
   dn:uid=jdoe,ou=People,dc=example,dc=com
   ds-certificate-fingerprint:40:73:7C:EF:1B:4A:3F:F4:9B:09:C3:50:2B:26:4A:EB
   ```

4. Create an index for the `ds-certificate-fingerprint` attribute. If the server is configured with multiple data backends, then the attribute should be indexed in each of those backends.

   ```
   $ bin/dsconfig create-local-db-index --backend-name userRoot \
   --index-name ds-certificate-fingerprint --set index-type:equality
   ```

5. Use the `rebuild-index` tool to cause an index to be generated for this attribute.

   ```
   $ bin/rebuild-index --task --baseDN dc=example,dc=com \
   --index ds-certificate-fingerprint
   ```

   ```
   [14:56:28] The console logging output is also available in
   '/ds/UnboundID-Metrics-Engine/logs/tools/rebuild-index.log'
   [14:56:29] Due to changes in the configuration, index
   dc_example_dc_com_ds-certificate-fingerprint.equality is currently
   operating in a degraded state and must be rebuilt before it can used
   [14:56:29] Rebuild of index(es) ds-certificate-fingerprint started with
   ```

```
161 total records to process
[14:56:29] Rebuild complete. Processed 161 records in 0 seconds (average
rate 1125.9/sec)
```

6. Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
--handler-name EXTERNAL \
--set "certificate-mapper:Fingerprint Mapper"
```

Configure the Subject Attribute to User Attribute Certificate Mapper

The Subject Attribute to User Attribute Certificate Mapper maps common attributes from the subject of the client certificate to the user's entry. The generated search filter must match exactly one entry within the scope of the base distinguished name for the mapper. If no match is returned or if multiple machine entries are found, the mapping fails.

Given the subject of the client certificate:

```
Owner: CN=John Doe, OU=Client Cert, O=Example Company, L=Austin, ST=Texas,
C=US
We want to match to the following user entry:
dn: uid=jdoe,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
uid: jdoe
givenName: John
sn: Doe
cn: John Doe
mail: jdoe@example.com
```

Perform the following to configure the Subject Attribute to User Attribute Certificate Mapper:

Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
--handler-name EXTERNAL \
--set "certificate-mapper:Subject Attribute to User Attribute"
```

Configure the Subject DN to User Attribute Certificate Mapper

The Subject DN to User Attribute Certificate mapper expects the user's entry to contain an attribute (`ds-certificate-subject-dn` by default) whose values match the subjects of the certificates that the user can use to authenticate. The `ds-certificate-subject-dn` attribute can be added to the user's entry together with the `ds-certificate-user` auxiliary object class. The attribute is multi-valued and can contain the subject distinguished names of multiple certificates. The certificate mapper must match exactly one entry, or the mapping will fail.

If using this attribute, add an equality index for this attribute in all data backends.

Perform the following steps to configure the Subject DN to User Attribute Certificate Mapper:

1. Create an LDIF file to add the `ds-certificate-user` object class and `ds-certificate-subject-dn` attribute to the target user's entry.

```
dn: uid=jdoe,ou=People,dc=example,dc=com
changetype: modify
add: objectClass
objectClass: ds-certificate-user
-
add: ds-certificate-subject-dn
ds-certificate-subject-dn:CN=John Doe,OU=Client Certificate,O=Example
Company,L=Austin,ST=Texas,C=US
```

2. Then, apply the change to the entry using `ldapmodify`:

```
$ bin/ldapmodify --filename add-cert-attr.ldif
```

3. Check that the attribute was added to the entry using `ldapsearch`.

```
$ bin/ldapsearch --baseDN dc=example,dc=com "(uid=jdoe)" \
```

```
ds-certificate-subject-dn
dn: uid=jdoe,ou=People,dc=example,dc=com
ds-certificate-fingerprint:CN=jdoe, OU=Client Cert, O=Example Company,
L=Austin, ST=Texas, C=US
```

4. Create an index to the `ds-certificate-subject-dn` attribute.

```
$ bin/dsconfig create-local-db-index --backend-name userRoot \
--index-name ds-certificate-subject-dn --set index-type:equality
```

5. Use the `rebuild-index` tool to ensure that the index is properly generated in all appropriate backends.

```
$ bin/rebuild-index --task --baseDN dc=example,dc=com \
--index ds-certificate-subject-dn
```

```
[15:39:19] The console logging output is also available in
'/ds/UnboundID-Metrics-Engine/logs/ tools/rebuild-index.log'
[15:39:20] Due to changes in the configuration, index
dc_example_dc_com_ds-certificate-subject-dn.equality is currently
operating in a degraded state and must be rebuilt before it can used
[15:39:20] Rebuild of index(es) ds-certificate-subject-dn started with 161
total records to process
[15:39:20] Rebuild complete. Processed 161 records in 0 seconds (average
rate 2367.6/sec)
```

6. Change the certificate mapper for the SASL EXTERNAL mechanism.

```
$ bin/dsconfig --no-prompt set-sasl-mechanism-handler-prop \
--handler-name EXTERNAL \
--set "certificate-mapper:Subject DN to User Attribute"
```

Chapter 7: Troubleshooting

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

This chapter includes the following information:

Collect Support Data Tool

Enable JVM Debugging

Performance Troubleshooting Example

Insufficient Memory Errors

Delays in Sample Data Availability

Slow Queries Based on Sample Cache Size

Unexpected Query Results

Installation and Maintenance Issues

Collect Support Data Tool

The 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 the Collect Support Data 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 127.0.0.1 --port 389 \
   --bindDN "cn=Directory Manager" --bindPassword secret \
   --serverRoot /opt/UnboundID-Metrics-Engine --pid 1234
   ```

3. Email the zip file to a support provider.

Enable JVM Debugging

Enable the JVM debugging options to track garbage collection data for the system. These options can impact JVM performance, but provide valuable data to tune the server. While the `jstat` utility with the `-gc` option can be used to obtain some information about garbage collection activity, there are additional arguments that can be added to provide additional detail, such as:

```
-XX:+PrintGCDetails
-XX:+PrintTenuringDistribution
-XX:+PrintGCApplicationConcurrentTime
-XX:+PrintGCApplicationStoppedTime
-XX:+PrintGCDateStamps
```

1. Perform the following steps to enable these options for the server:
2. On the Metrics Engine server, navigate to the `config/java.properties` file.
3. Edit the `config/java.properties` file. Add any additional arguments to the end of the line that begins with `start-metrics-engine.java-args`.
4. Save the file.

5. Run the following command for the new arguments to take effect the next time the server is started:

```
$ bin/dsjavaproperties
```

Delays in Sample Data Availability

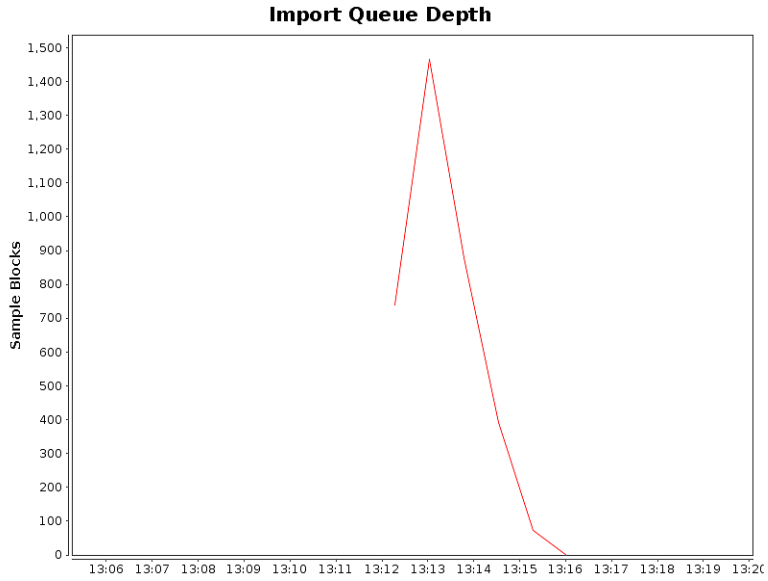
The time between when a metric sample is captured and when it is available in the Metrics Engine is a combination of queuing and polling delays. The default configuration allows the monitored server to queue samples in memory for up to 30 seconds before writing them to disk.

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 nor-
malized 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
```

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

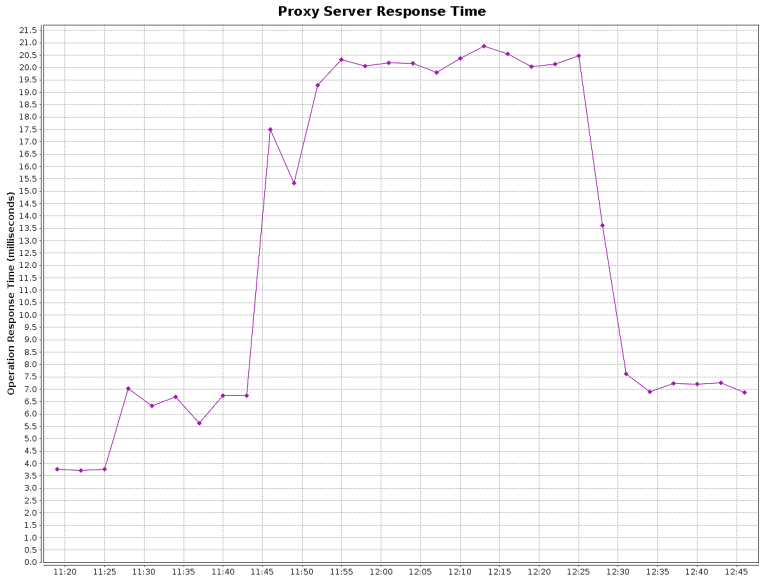
The Metrics Engine monitors itself at the same time it monitors other servers, so the historical view of the status and performance of the Metrics Engine is captured in the DBMS and is available for historical analysis.

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 Identity 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.

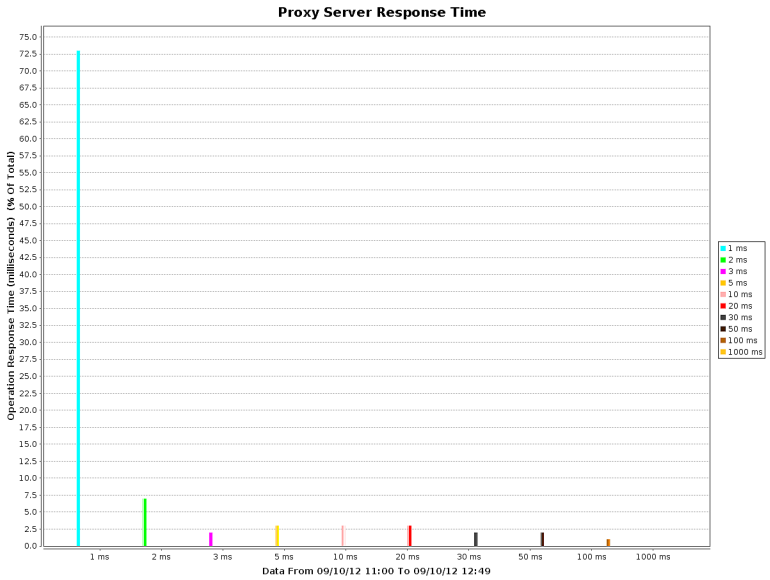
A plot of the average Identity Proxy response time that covers the time frame of the issue is captured using the following `query-metric` command:

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

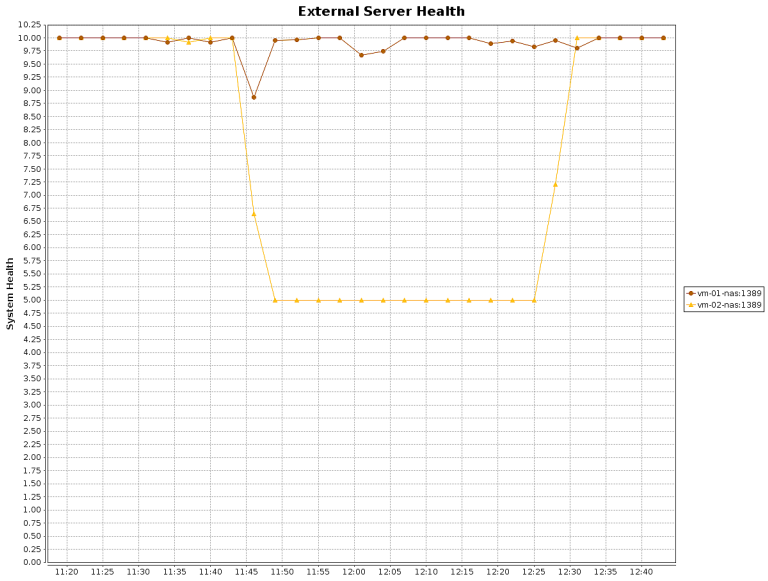
The command displays the following chart.



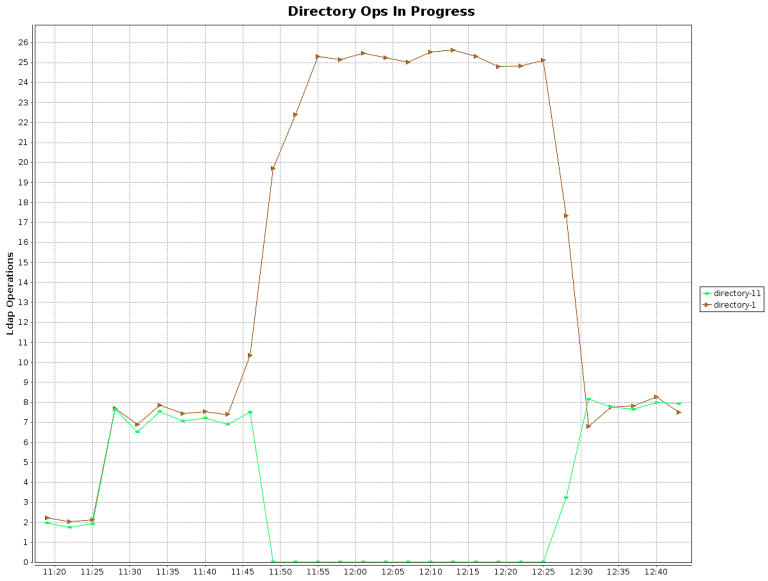
This chart shows the application response time tripled. This could mean that either a few requests took a long time, or everything slowed down. To get more information, use the `query-metric` command to get a plot of the application response time histogram over the same time. The result is the following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



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 Identity Data Store. The next step is to determine what each Identity 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/2012 11:47:39 -0500 : A severe backlog has been detected in the
Directory Server work queue. The operation currently at the head of the queue
has been waiting for 25785 milliseconds
Error : 10/Sep/2012 11:47:25 -0500 : A severe backlog has been detected in the
Directory Server work queue. The operation currently at the head of the queue
has been waiting for 11790 milliseconds
Warning : 10/Sep/2012 11:47:12 -0500 : The Directory Server 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/de-
ploy/ds2/logs' (totalBytes: 18624344064, usableBytes: 1851559936, usablePer-
cent: 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 Metrics Engine 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 by the metric queries. Consider increasing the heap size, or reducing the number of request handler threads using the following `dsconfig` command:

```
./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-
x=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 Metrics Engine 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 Metrics Engine. 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 Metrics Engine 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 Metrics Engine 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 Metrics Engine could not acquire an exclusive lock on file
/ds/UnboundID-Metrics-Engine/locks/server.lock:
The exclusive lock requested for file
/ds/UnboundID-Metrics-Engine/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 Metrics Engine 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 Metrics Engine 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, then the Metrics Engine generates an error message indicating that the server 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 Metrics Engine 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 Metrics Engine to use a smaller amount of memory.
- The Metrics Engine 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 Metrics Engine 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 Metrics Engine. 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 Metrics Engine 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 Metrics Engine 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 Metrics Engine 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 Metrics Engine 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 Metrics Engine installation to that new user. For example, if the Metrics Engine should be run as the "ds" user in the "other" group, run the following command as root:

```
chown -R ds:other /ds/UnboundID-Metrics-Engine
```

The Server has Shutdown

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

- 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 Metrics Engine does not appear to be accepting connections from clients, reasons can include the following:

- The Metrics Engine is not running.
- The underlying system on which the Metrics Engine is installed is not running.
- The Metrics Engine 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 Metrics Engine is configured to allow secure communication via 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 Metrics Engine 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 Metrics Engine is configured to restrict access based on the address of the client, then messages should be written to the server access log for each rejected connection attempt.
- If a connection handler encounters a significant error, then 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 Metrics Engine process is running and appears to be accepting connections but does not respond to requests received on those connections, then potential reasons for this behavior 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 Metrics Engine is running is not properly configured, it can spend too much time performing garbage collection. The affect on the Metrics Engine 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 Metrics Engine 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 a 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 Metrics Engine will be unresponsive.

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

Problems with the Web Console

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

- 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 Metrics Engine is online. If it is, the access log may provide information about the authentication failure.
- If a problem occurs while interacting with the Metrics Engine instance using the Console, the access and error logs for that instance may provide additional information.

Chapter 8: Metric 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, such as the Google Chart Tools. The Metrics Engine API is also easily accessed from a Web browser.

This section includes the following:

Connection and Security

Response Codes

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 Monitoring Configuration object. Set this property as follows:

```
$ bin/dsconfig set-monitoring-configuration-prop --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. Load the entry using `ldapmodify`.

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

3. Authenticate using either the `cn` or the `uid` of the user added, in this case `apiuser1`.

Response Codes

The following response codes are available.

Response Code	Description
200 0K	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.

Response Body

```
<?xml version="1.0" encoding="UTF-8"?>
<errorResponse>
   <message>There are no metrics defined with id response-
tme.
      Available metrics may be found at /metrics
   </message>
</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

Formats JSON, XML

Query Parameters

`instanceHostname` – Hostname(s) of the servers to get data from. Multiple values are evaluated as logical ORs.

`instanceLocation` – Location(s) of the servers to get data from. Multiple values are evaluated as logical ORs.

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

- `directory`
- `proxy`
- `sync`
- `metrics-engine`

`instanceVersion` – Version(s) of the servers to get data from. Multiple values are evaluated as logical ORs.

EXAMPLES

All instances in JSON format.

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

All directory and proxy instances in XML format:

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

Response Code 200 0K

Response Body

```
{
  "found" : 2
   "offset" : 0,#
 "instances" : [ {
   "type" : "directory",
   "id" : "unboundid4510",
   "hostname": "unboundid4510.example.com",
   "displayName" : "unboundid4510",
   "version": "UnboundID Directory Server 4.5.1.0",
   "operatingSystem": "Solaris",
   "status": {
      "state": "ONLINE"
      }
  }, {
     "type" : "directory",
   "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.

URL /api/v1/instances/{instance}{.format}

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

EXAMPLE:

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

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

Response Code	200 0K

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 4.5.1.0"
	}

List Available Metrics

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

URL	/api/v1/metrics{.format}
Method	GET
Formats	JSON, XML
Query Parameters	`name` – Limits the results to metrics whose names contain a matching substring. The search is not case-sensitive. `type` – Limits the results to the metrics of the specified type. Possible values are: • `discreteValued` • `continuousValued` • `count` `group` – Limits the results to the metrics within the specified group. Possible values are: • `Directory Backend` • `Monitoring Data Cache` • `Java Virtual Machine` • `LDAP` • `Entry Balancing` • `Directory Entry Cache` • `External Server` • `Host System` • `Metric Query` • `Monitoring DBMS` • `Monitoring Data Processing` • `Replication` • `Sync Pipe` `instanceType` – Limits the result to metrics that uses the specified instance types as sources. Possible values are: • `directory` • `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 "directory backend" group providing either count or average statistics:

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

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

Response Code 200 0K

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": "Directory Backend",
      "id": "backend-checkpoints",
      "instanceTypes": [
          "directory"
```

```
        ],
        "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.

URL	/api/v1/metrics/{metricId}{.format}
Method	GET
Formats	JSON, XML
Query	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 0K
Response Body	Count type metric.

```
<?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>directory</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:

- `directory`
- `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 7/7/2014 for all identity data stores and identity proxies in two locations, 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=2014-07-07&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:

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

Response Code	200 0K
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" : [ "directory", "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-sys-
tem-cpuused/
chart?maxIntervals=50&startTime=-1w&pivot=instance:
```

Google Chart Tools Datasource Protocol

Metrics data can be presented with Google's Chart Tools Datasource protocol (`https://developers.google.com/chart/interactive/docs/dev/implementing_data_source`). 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.

URL	/api/v1/metrics/{metricId}/datatable
Method	GET
Formats	JSON, HTML, CSV, and TSV
Query Parameters	`tqx=out:html` – HTML formatted output. `tqx=out:csv` – CSV formatted output. `tqx=out:tsv-excel` – TSV formatted output. `tz` – 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 Code	200 0K
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-
c="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=optype&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 0K
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 0K

Response Body	`{`
	`"found" : 2,`
	`"offset" : 0,`
	`"events" : [`

```
  {"id":"9bdfd1b8-3811-4a84-b779-93553ff35f83",
       "creationDate":1351274815559,
       "eventType":"server-starting",
       "eventSeverity":"INFO",
       "sourceProductInstance":"lockdown-test",
       "summary":"Server Starting",
       "detail":"The Directory Server is starting"},
  {"id":"9bdfd1b8-3811-4a84-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.
Method	GET
Formats	JSON, XML
Query Parameters	**For the 1st URL:** `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.

For the 2nd URL:

`historical` (multi-valued, optional):

- `time` – Includes time series data.
- `limits` – Iincludes the percent of time thresholds limits that have been exceeded. Requires Threshold.
- `alerts` – Includes all Threhold alerts. Requires Thresholding.
- `histogram` – includes response-time histogram as column data)
- `nines` – Includes response time values that correlate to 99%, 99.9%, 99.99%, and 99.999% response-time measurements)

`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.

- `instance` – pivot by producing server.
- `ldap-op` – pivot by LDAP operation.
- `histogram` – pivot response-time series by histogram buckets.

`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/ap-
i/v1/sla/ldap/Acme+Identity+Portal?historical=time
\&historical=nines\&pivot=instance\&startTime=-15m
```

Response Code 200 0K

Response Body

(JSON, sample is abbreviated)

```
{
  "name":"Acme Identity Portal",
  "applicationName":"Application 5",
  "ldapOps":["search"],
  "servers":["x2270-08.unboundid.lab:1389"],
  "enabled":true,
  "responeTimeState":"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.

Query Parameters

`limit` – Specifies the maximum number of results to return. The default is to return all results.

`offset` – Specifies how many results to skip for the first results to return.

Response Parameters

`found` – The number of results that satisfied the query parameters.

`offset` – The index into the total result set where the current response begins.

Index

A

aggregating data 51

alarms 29

- testing setup 30

alerts

- alarm_cleared alert type 30
- configure alert handlers 28
- list of system alerts 28, 30
- notifications and alerts 27
- testing setup 30

alerts backend

- alert retention time 29
- duplicate alert suppression 29
- overview 28
- view information 28

API

- access alerts 132
- add a REST user 118
- chart image 128
- connection and security 118
- data set structure 127
- LDAP SLA 133
- list available metrics 121
- list monitored instances 119
- pagination 135
- perform a metrics query 125
- response codes 119
- retrieve a metric definition 124
- retrieve monitored instance 120

authentication mechanisms 83

B

backend monitors
- disk space usage 26
- entries 25

backup command 34

base64 command 34

broker-dashboard 57
- add charts 66

C

certificates
- create with keytool 75

chart builder tool
- overview 61

charts
- add to broker-dashboard 66
- available server charts 64
- chart builder parameters 63
- chart builder tool 61
- chart image API 128
- chart properties file 64
- configure for Identity Broker 66
- presentation details 62, 128

cn=monitor backend 46

collect-support-data tool 34, 104

command-line
- available tools 34
- default properties file 37
- tools.properties file 36

create-rc-script command 34

D

dashboards
- available dashboards 57
- configure dashboards 60

debug template files 60
save custom files 60
data collection
aggregating data 51
importing data 50
overview 2
performance data 45
reduce data collected 49
reduce frequency of collection 49
reduce sample blocks 49
system monitoring data 46
Data Store charts 65
demo-dashboard 59
disk space usage monitor 26
dsconfig command 34
dsframework command 34
dsjavaproperties command 35

E

error log publisher 24
external collector daemon 48

G

gauges 29
testing related alarms and alerts 30

H

host system monitor provider 48

I

Identity Broker charts 65
configure Identity Broker 67

J

Java
installing the JDK 7
JVM debugging 104
during setup 111

invalid options 112

K

key manager 78

keytool 75, 77

L

ldap-dashboard 57

ldapmodify command 35

ldappasswordmodify command 35

ldapsearch command 35

ldif-diff command 35

ldifmodify command 35

Linux configuration

filesystem swapping 10

filesystem variables 9

install dstat 10

install sysstat and pstack 10

set file descriptor limit 9

set filesystem flushes 10

logs

create log publisher 23

error log publisher 24

overview 23

retention policies 23

rotation policies 23

M

manage-extension command 35

memory errors 109

metric-engine-schema command 35

metrics

continuous metrics 40

count metrics 40

dimensions 41

discrete metrics 40

list available metrics 121
overview of metric types 40
performance impact 49
query overview 42
sample data delays 105
Metrics Engine
components 2
install 12
install web console 17
overview 2
start server 15
stop server 15
uninstall 16
Metrics Engine charts 65
monitored-servers command 35
monitored-servers tool 14
monitored servers
add servers 14
configure servers to monitor 10
dsconfig tool 14
monitored-servers tool 14
processing time histogram 11
stats collector 11
tracked applications 11
monitoring configuration object 118
monitoring entries 25

N

non-root user 8
normalized records 46

P

performance data
overview 3
performance data fields 46

pivots 43
PostgreSQL
  backup database 32
  data storage 32
  install 12
  plan the backup 33
  restore the backup 34
  start the backup 33
processing time histogram plugin 11
Proxy Server charts 65

Q

query-metric command 35
  access metrics 44
query data
  aggregate query results 43
  pivots 43
  select query data 43
  unexpected results 109
query overview 42
queryrate command 35

R

REST API
  overview 2
restore command 35
revert-update command 35
review-license command 35

S

sample-flush-interval property 11
server-state command 35
server clock skew 48, 110
server status 121
  example 108

service level agreements 133
monitoring overview 51
SLA dashboard 58
SLA object 52
spike monitoring threshold 52-53
setup command 35
troubleshooting 110
sla-viewer-details dashboard 58
sla-viewer dashboard 58
SLA object 52
configure object 54
Solaris configuration
ZFS configuration 8
SSL support 79
start-metrics-engine command 35
start Metrics Engine server 15
StartTLS support 79
stats collector plugin 11, 47, 50
cn=monitor backend 46
status command 35
stop-metrics-engine command 35
stop Metrics Engine server 16
sum-file-sizes command 35
summarize-config command 35
supported platforms 6
application servers 7
browser requirements 7
hardware requirements 6
Java JDK requirements 7
operating systems 6
virtual hosts 7
Sync Server charts 65
system data
overview 3

system utilization monitors 47

T

tools.property file 36
tracked applications 11
troubleshooting
  client connections 114
  collect support data 104
  installation 110
  JVM debugging 104
  memory errors 109
  performance example 106
  sample data delays 105
  server shutdown 113
  server unresponsive 115
  slow queries 106
  unexpected query results 109
  web console 115
trust manager 78

U

UnboundID
  about 1
uninstall command 35
uninstall Metrics Engine 16
update command 35

V

Velocity templates
  multiple content types 69
  overview 67
  save custom files 61
  tools context provider 71

W

web console
  configure security 19

configure servers 18
configure Tomcat 17
install 17
log into 20
uninstall 21
upgrade 20
URL 18

Z

ZFS configuration 8