UnboundID Metrics Engine Administration Guide

Version 3.6.0

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

Copyright

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

The contents of this publication are presented for information purposes only and is provided “as is”. While every effort has been made to ensure the accuracy of the contents, the contents are not to be construed as warranties or guarantees, expressed or implied, regarding the products or services described herein or their use or applicability. We reserve the right to modify or improve the design or specifications of such products at any time without notice.

Published: 2012-10-29

Contents

Preface.......... v
Purpose of This Guide.......... v
Audience.......... v
Related Documentation.......... v
Document Conventions.......... vi

Chapter 1: Overview of the Metrics Engine.......... 1
Overview of the Metrics Engine.......... 2
About Data Collection.......... 3

Chapter 2: Installing the Server.......... 5
Before You Begin.......... 6
Supported Operating Platforms.......... 6
Software Requirements.......... 6
Installing Java.......... 7
About the PostgreSQL DBMS.......... 8
Preparing the Operating System (Solaris).......... 9
Preparing the Operating System (Linux).......... 11
Running as a Non-Root User.......... 15
Installation Process Overview.......... 17
Directory Server Configuration.......... 17
Configuring the Metrics Backend.......... 17
Configuring the Processing Time Histogram Plugin.......... 18
Setting the Connection Criteria to Collect SLA Statistics by Application.......... 19
Updating the Global Configuration.......... 19
Setting Up the Database.......... 20
About Setting Up the PostgreSQL DBMS Database.......... 20
About the Metrics Engine DBMS Users.......... 22
Installing and Configuring PostgreSQL on Centos/RHEL.......... 23
Configuring Administrative Credentials for a PostgreSQL database.......... 23
Locating PostgreSQL Configuration Files.......... 24
Configuring PostgreSQL Connections.......... 24
Enabling psql for the Metrics Engine User.......... 24
Configuring the Timezone and Routine Vacuuming.......... 25
Tuning the PostgreSQL Configuration.......... 25
Installing the Metrics Engine.......... 26
Unpacking the Installation Packages.......... 26
To Install the Metrics Engine.......... 26
Configuring the Servers Monitored by the Metrics Engine.......... 28
About the monitored-servers Tool.......... 28
About Adding Individual Servers Using dsconfig.......... 28
To Configure the Servers Monitored by the Metrics Engine.......... 28
To Add Individual Monitored Servers Using dsconfig.......... 29
Running the Metrics Engine.......... 29
Starting the Metrics Engine.......... 29
To Stop the Metrics Engine.......... 31
To Restart the Metrics Engine.......... 31
Installing the Metrics Engine Management Console.......... 31

To Install the Metrics Engine Management Console Out of the Box....32
Logging into the Metrics Engine Management Console.... 33
Fine-Tuning the Metrics Engine Management Console.... 34
Upgrading the Metrics Engine Management Console.... 36
Backing Up the Metrics Engine DBMS.... 36
About Backing Up DBMS Data.... 36
Before You Begin Your Backup....39
How to Backup the Database....40
How to Restore the Database....40
Performing a Full Backup.... 41
How to Export and Import the Database.... 41
Uninstalling the Metrics Engine....42
To Uninstall the Metrics Engine in Interactive Mode.... 42
Uninstalling the Metrics Engine in Non-Interactive Mode....42
Uninstalling the Metrics Engine Management Console....43
Cleaning Up the PostgreSQL DBMS After Uninstall.... 44
Upgrading and Reverting the Metrics Engine.... 44
Metrics Engine DBMS Schema Considerations.... 45
Overview of the Upgrade Process....45
To Upgrade the Metrics Engine....45
To Revert the Metrics Engine....46

Chapter 3: Data Collection and Metrics....47

Overview of Metrics Concepts....48
About Analzying Aggregated Data.... 48
About the Types of Metrics.... 50
About Dimensions.... 51
Overview of Query Concepts....55
Selecting Query Data....55
Aggregating the Query Result....56
Formatting the Query Result....57
About the Data Collection Process.... 57
About Performance Data.... 57
About the Collection of System Monitoring Data....58
About the External Collector Daemon....59
About Monitored Server Clock Skew....60
Tuning Data Collection.... 61
Reducing the Data Collected....61
Reducing the Frequency of Data Collection....61
Reducing the Frequency of Sample Block Creation....61
Reducing Metrics Engine Impact on Directory Performance.... 62
About Data Processing on the Metrics Engine....62
Data Importing....63
Data Aggregation....63
Accessing Monitoring Data.... 65

Chapter 4: Accessing the Metrics Engine Data.... 67

About the query-metric tool.... 68
Using the Query Metric Tool....68
About the Query Metric Explore Command HTML Pages.... 69
About the Metrics Engine API....71
Metrics Engine API Reference....71
Connection Security and Authentication....71

Tuning the RESTful API Service....72
Listing Monitored Instances....73
Retrieving a Monitored Instance....74
Listing Available Metrics....74
Retrieving a Metric Definition....78
Performing a Metric Query....80
Pagination....86
Response Codes....86

Chapter 5: Managing the Metrics Engine....87

Working With Logs....88
Creating New Log Publishers....88
Configuring Log Rotation....89
Configuring Log Retention....90
Managing the File-Based Error Log Publisher....90
Monitoring the Metrics Engine....91
Monitoring Disk Space Usage....93
Monitoring with JMX....94
Running JConsole....94
Monitoring the Metrics Engine Using JConsole....94
Managing Notifications and Alerts....95
Working with Administrative Alert Handlers....95
Configuring the JMX Connection Handler and Alert Handler....99
Configuring the SMTP Alert Handler....100
Configuring the SNMP Subagent Alert Handler....101
Working with the Alerts Backend....101
Command-Line Tools....102
Using the Help Option....103
Available Command-Line Utilities....103
Managing the tools.properties File....104

Chapter 6: Troubleshooting the Metrics Engine....109

Debugging the Metrics Engine....110
Working with the Troubleshooting Tools....110
Working with the Collect Support Data Tool....110
Metrics Engine Troubleshooting Tools....111
Server Version Information....111
Embedded Profiler....111
Troubleshooting Resources for Java Applications....112
Java Troubleshooting Documentation (Oracle/Sun JDK)....112
Java Troubleshooting Tools (Oracle/Sun JDK)....113
Java Diagnostic Information....116
Java Troubleshooting Tools (IBM JDK)....116
Troubleshooting Resources in the Operating System....117
Identifying Problems with the Underlying System....117
Examining CPU Utilization....117
Examining Disk Utilization....118
Examining Process Details....119
Tracing Process Execution....120
Examining Network Communication....121
Troubleshooting Performance Problems....122
Example of Interpreting Performance Data to Troubleshoot Problems....122
Long Time Before Samples Appear in Queries....125

- Slow Queries for a Particular Metric........ 126
- All Metric Queries are Slow........ 127
- Strange Query Results for Time Ranges Ending Now........ 128
- Optimizing the Layout of the Sample Data Table........ 129

Troubleshooting the Metrics Engine API........ 129

Common Problems and Potential Solutions........ 129

- The Server Will Not Run Setup........ 129
- The Server Will Not Start........ 131
- The Server Has Crashed or Shut Itself Down........ 134
- The Server Will Not Accept Client Connections........ 135
- The Server is Unresponsive........ 136
- Problems with the Directory Management Console........ 137
- Providing Information for Support Cases........ 137

Preface

This guide presents the procedures and reference material necessary to install, administer and troubleshoot the UnboundID® Metrics Engine in multi-client, high-load production environments.

Purpose of This Guide

The purpose of this guide is to provide valuable procedures and concepts that can be used to manage the UnboundID® Metrics Engine in a multi-client environment. It also provides information to monitor and set up the necessary logs needed to troubleshoot the server's performance.

Audience

The guide is intended for administrators responsible for installing, maintaining, and monitoring servers in large-scale, high load production environments. It is assumed that the reader has the following background knowledge:

- Directory Services and LDAPv3 concepts
- System administration principles and practices
- Understanding of Java VM optimization and garbage collection processes
- Application performance monitoring tools

Related Documentation

The following list shows the full documentation set that may help you manage your deployment:

- *UnboundID® Directory Server Administration Guide*
- *UnboundID® Directory Server Reference Guide (HTML)*
- *UnboundID® Directory Proxy Server Administration Guide*
- *UnboundID® Directory Proxy Server Reference Guide (HTML)*
- *UnboundID® Synchronization Server Administration Guide*
- *UnboundID® Synchronization Server Reference Guide (HTML)*
- *UnboundID Metrics Engine Administration Guide*
- *UnboundID LDAP SDK for Java API*
- *UnboundID Server SDK API*

Document Conventions

The following table shows the document convention used in this guide.

Convention	Usage
`Monospace`	Commands, filenames, directories, and file paths
`Monospace Bold`	User interface elements, menu items and buttons
Italic	Identifies file names, doc titles, terms, variable names, and emphasized text

Chapter 1 Overview of the Metrics Engine

The UnboundID® Metrics Engine provides collection and storage of performance data from your UnboundID server topology. This chapter introduces the concepts and applications associated with the UnboundID Metrics Engine.

Topics:

- *Overview of the Metrics Engine*
- *About Data Collection*

Overview of the Metrics Engine

In any large-scale directory and user identity infrastructure, there are expectation and service levels to meet for uptime, scalability transaction response time and throughput. The UnboundID® Metrics Engine gives you insight into how your identity infrastructure is performing. It collects data from the internal instrumentation of the UnboundID Directory Server, UnboundID Proxy Server and UnboundID Synchronization Server, across the instances, replicas, and data centers in your environment.

Using the instantaneous and historical data available from the Metrics Engine, you can now:

- Measure and visualize the performance of the identity infrastructure as a whole service, not just as a collection of individual servers. This data provides the ability to justify and measure the achievement of service-level agreements.
- Identify those client applications and request types that are responsible for the largest resource loads, so that improvement efforts can be applied where they have the greatest impact.
- Determine which servers have the most available capacity, so that requests or request types can be reallocated accordingly.
- Discover a server instance that is under-performing due to resource limitations or misconfiguration.
- Predict the capacity of your infrastructure to accommodate growth in request traffic and identity data.
- Produce detailed analysis of all measurement taken around any abnormal performance event to quickly identify the root cause.

The following diagram illustrates the components of the Metrics Engine and how they interact.

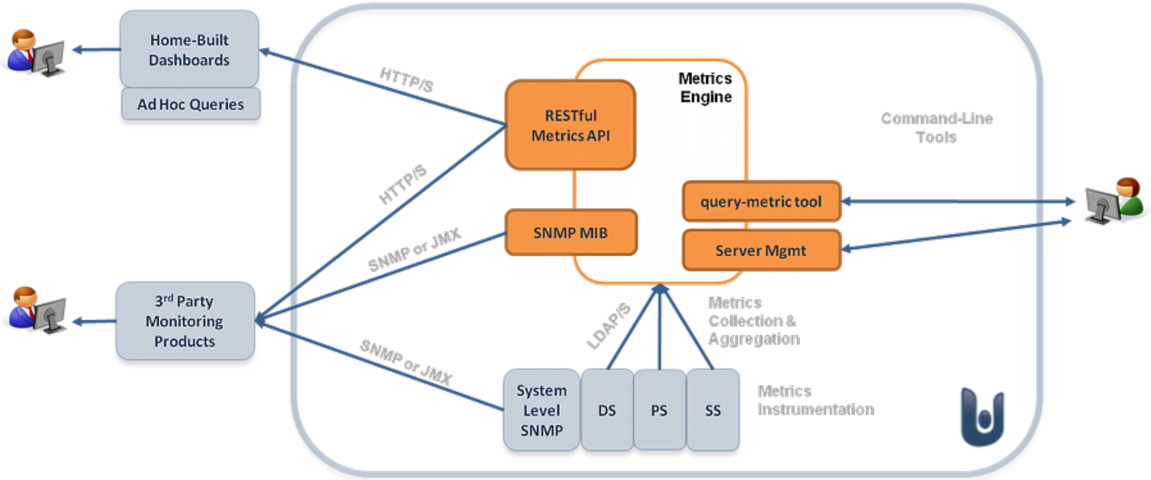


Figure 1: Key Components of the Metrics Engine

The diagram contains the following key components:

- **Metrics Engine.** The Metrics Engine itself is a stand-alone server, just like the other products of the UnboundID Core Server Suite. It includes versions of the same configuration, management, and logging tools as the other components of the suite. The Metrics Engine relies on a captive PostgreSQL data store for the collected metrics.
- **Metrics API.** A RESTful API, accessible over HTTP/S, gives easy access to the collected metrics and to information about the systems they represent. The API supports parameters for complete control over the data being returned, including filtering, minimum/maximum, average, server types, multiple data series (pivots), historical time periods, units, and histogram data.
- **query-metric tool.** This tool is the primary command-line tool for access to the metric data. In the interactive mode, use the tool to investigate the performance of the service. The query metrics tool also has a parameter-driven command-line mode for automating the extraction of data from the Metrics Engine, ideal for use with shell scripts. It includes an `explore` option that allows you to generate queries that drive the Metrics API, such as adding a specific chart or tabular result to a custom dashboard. It also generate HTML page output.
- **SNMP access.** Each individual UnboundID server makes its own system-level metrics available over SNMP.
- **Data set.** The Metrics Engine proprietary data set structure is space-optimized and designed for easy interoperability with charting libraries like Highcharts, FusionCharts, or JFreeChart.
- **SLA Viewer.** An example of a dynamic, rapidly-updated service level dashboard application. Released under the umbrella of UnboundID Labs as an unsupported add-on to the Metrics Engine, you may find the SLA Viewer a useful tool in its own right. You can get it at `https://www.unboundid.com/labs/`.

Our customers use a variety of commercial application performance monitoring tools. The Metrics Engine supports integration with third-party products through any of these data access mechanisms:

- Metrics REST API, accessed over HTTP
- SNMP
- Scripted use of `query-metric` tool
- Entries in `cn=monitor` available over LDAP

About Data Collection

The Metrics Engine provides collection and storage of performance data for a set of UnboundID Directory, Proxy, and Synchronization Servers. The current value of some of this data is accessible via LDAP at `cn=monitor` on the monitored servers. Each monitored server collects and locally stores a limited history of performance data. This history is organized into time-contiguous blocks available via LDAP at `cn=metrics`.

To collect the performance data, the Metrics Engine continuously polls all monitored product servers, fetching any new data and keeping it in a PostgreSQL DBMS. This polling incurs a small load on the product servers, so you should understand the value of collecting the data to make an informed cost/benefit decision. For an in-depth discussion of the data collected by the Metrics Engine and the metrics available, see “Data Collection and Metrics”.

The following figure illustrates the data collection pipeline.

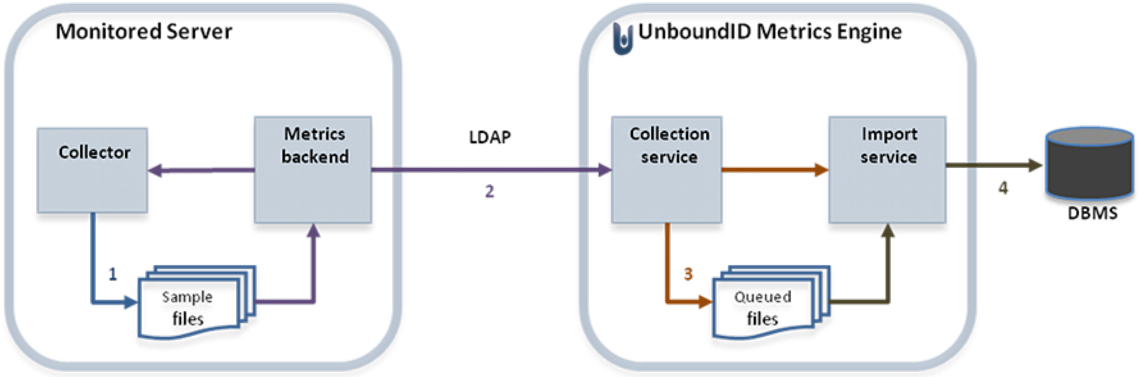


Figure 2: Data Collection Pipeline

Data collection flows in the diagram as follows. The expected delay between steps one and four is about 60 seconds.

1. Samples are taken and stored in time-contiguous blocks on the disk by the monitored server.
2. The Collection Service polls for new sample blocks.
3. New sample blocks are queued to disk on the Metrics Engine.
4. The Import Service loads new blocks into the DBMS.

Chapter 2 Installing the Server

This section describes how to install and configure the Metrics Engine.

Topics:

- *Before You Begin*
- *Installation Process Overview*
- *Directory Server Configuration*
- *Setting Up the Database*
- *Installing the Metrics Engine*
- *Configuring the Servers Monitored by the Metrics Engine*
- *Running the Metrics Engine*
- *Installing the Metrics Engine Management Console*
- *Backing Up the Metrics Engine DBMS*
- *Uninstalling the Metrics Engine*
- *Upgrading and Reverting the Metrics Engine*

Before You Begin

This section describes prerequisites for installing the Metrics Engine, including hardware and software requirements.

Supported Operating Platforms

Multi-Platform Support. The UnboundID® Metrics Engine is a pure Java application. It is intended to run within the Java Virtual Machine on any Java 6 Standard Edition (SE) or Enterprise Edition (EE) certified platform. For the list of supported platforms and Java versions, access your Customer Support Center portal or contact your authorized support provider.

The Metrics Engine runs both a Java Application Server and a PostgreSQL® RDBMS. Any additional RAM that can be used by PostgreSQL will improve the performance of the `query-metric` tool or the Metrics Engine RESTful API by allowing PostgreSQL to cache data in memory, reducing disk input and output.

Your topology should meet the following hardware requirements:

Table 1: Hardware Requirements

Topology Type	Metrics Engine RAM	DBMS RAM	Disk	CPU	Notes
Small (1 to 6 monitored servers)	8 GB	4 GB	300 GB	4 cores	
Medium (7 to 16 monitored servers)	20 GB	8 GB	400 GB	6 cores	A hardware RAID caching controller with a non-volatile write cache is desirable. Multiple disk spindles is helpful for DBMS data.
Large (17 to 50 monitored servers)	32 GB	16 GB	500 GB	8 cores	A hardware RAID caching controller with a non-volatile write cache is required. Multiple disk spindles is necessary for DBMS data.

The filesystem buffer cache can use any additional system RAM not allocated to the Metrics Engine JVM or PostgreSQL DBMS to cache recently read disk pages. Running other processes on this system will have a detrimental effect on query performance.

Software Requirements

Before you install the Metrics Engine, you need the following:

- An installer for PostgreSQL 9.1 or higher. Get the package installer that is native for your platform. The package installers create the DBMS user and sets the file permissions correctly. PostgreSQL must be installed on the same machine as the Metrics Engine.
- JVM version 1.6 or higher. For more about this requirement, see “Java Software Requirements”.

Installing Java

For optimized performance, the UnboundID® Metrics Engine requires Java for 32-bit and 64-bit architectures, respectively, depending on your system requirements. You can view the minimum required Java version on your Customer Support Center portal or contact your authorized support provider for the latest software versions supported.

Even if your system already has Java installed, you may want to create a separate Java installation for use by the UnboundID® Metrics Engine to ensure that updates to the system-wide Java installation do not inadvertently impact the Metrics Engine. This setup requires that the JDK, rather than the JRE, for both the 32-bit and 64-bit versions or both depending on your operating system, be downloaded.

On Solaris systems, if you want to use the 64-bit version of Java, you need to install both the 32-bit and 64-bit versions. The 64-bit version of Java on Solaris is not a full stand-alone installation, but instead relies on a number of files provided by the 32-bit installation. Therefore, the 32-bit version should be installed first, and then the 64-bit version installed in the same location with the necessary additional files.

On other platforms (for example, Linux and Microsoft Windows), the 32-bit and 64-bit versions of Java contain complete installations. If you only want to run the 64-bit version of Java, then it is not necessary to install the 32-bit JDK. If you want to have both versions installed, then they should be installed in separate directories, because the files cannot co-exist in the same directory as they can on Solaris systems.

To Install Java (Oracle/Sun)

1. Open a browser and navigate to the following Oracle download site:

   ```
   http://www.oracle.com/technetwork/java/javase/downloads/index.html
   ```

2. Download the latest version Java JDK. Click the JDK Download button corresponding to the latest Java update.

3. On the Java JDK page, click the Accept Licence Agreement button, then download the version based on your operating system.

To Install Java (IBM)

1. Open a browser and navigate to the following IBM download site:

   ```
   http://www.ibm.com/developerworks/java/jdk/
   ```

2. Select the Java version for your operating system.

About the PostgreSQL DBMS

The Metrics Engine uses a PostgreSQL DBMS to store the sample data. A traditional table-based DBMS serves the needs of the Metric Engine better than the attribute-based DBMS used by the UnboundID Directory Server because the sample data is tabular and the RDBMS system aggregates data efficiently.

To determine the storage requirements of the DBMS, you need to understand the expected data access patterns. The Metrics Engine interacts with the DBMS in four ways:

- Sample import
- Sample aggregation
- Sample age-out
- Sample query

Sample import places a very predictable and steady write load on the DBMS. This single-threaded interaction puts a table-level lock on the target table. Sample imports account for 80% of the writes to the DBMS, so performance benefits from a server-class RAID disk controller with a non-volatile write cache. A Metrics Engine that monitors 20 servers keeps a single 10K RPM disk 70% busy with this single interaction.

Sample aggregation places a very predictable but less frequent read/write load on the DBMS. This interaction is responsible for the aggregation of samples from one time resolution to the next, so it reads from one set of tables and writes 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. This DBMS interaction is negligible when sample imports are taking place.

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.

Sample queries are the least predictable, because they result from clients hitting the public API requesting metric samples. The API allows you to aggregate multiple dimensions and multiple servers in a single request, so 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 be as slow as 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 sample 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 directory 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. Predicting what samples a future query may want is impossible, and the results from previous queries are cached such that a subsequent request for the same data results in very little 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.

Preparing the Operating System (Solaris)

The UnboundID® Metrics Engine has been extensively tested on multiple operating systems. We have found that serveral operating system optimizations lead to improved performance. These optimizations include using the ZFS filesystem on Solaris systems, restricting ZFS memory consumption, limiting transaction group writes, using compression and disabling access time updates.

Using ZFS

UnboundID strongly recommends the use of ZFS™ as the underlying filesystem on Solaris 10 and OpenSolaris systems. ZFS is a 128-bit filesystem that can store billions of times more data than traditional 64-bit systems. Based on a storage pool model, ZFS aggregates devices (mirrors, RAID-Z with single or double parity, concatenated or striped storage) into a virtual data source from which filesystems can be constructed. ZFS provides excellent performance, end-to-end data integrity, simple administration management, and unmatched scalability. It also provides many useful features, such as automatic checksum, dynamic striping, variable block sizes, compression, and unlimited constant-time snapshots. ZFS is part of the Solaris 10 and OpenSolaris operating 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 (for example, one pool for the database and a second for log files). Single zpool configurations are the simplest and easiest to manage. From there, you can create multiple filesystems inside the pool and optionally reserve space for one or more of the filesystems.

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, so that as many underlying spindles as possible can be used to provide the configuration with the greatest number of I/O operations per second.

To Restrict ZFS Memory Consumption

Despite its excellent performance, ZFS does not release memory fast enough for some LDAP operations that might need it. This delay could cause some processes to fail to start while attempting to allocate a large amount of memory for a JVM heap.

To curb memory allocation problems, make sure that the system is configured to limit the amount of memory for caching (for example, up to two gigabytes). The Metrics Engine relies on database caching rather than filesystem caching for its performance. Thus, the underlying system should be configured, so that the memory used by ZFS will not interfere with the memory used by the Metrics Engine. In most environments, we recommend that systems be configured to allow ZFS to use no more than 2 GB of memory for caching.

1. Open the `/etc/system` file.

2. ZFS caches data from all active storage pools in the ARC cache. We can limit its memory consumption by setting the maximum size of the ARC caches using the `zfs_arc_max` property. For example, add the following line to the end of the `/etc/system` file.

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

This property sets the maximum size of the ARC cache to 2 GB (0x80000000 or 2147483648 bytes) for ZFS. Note that your system may require a different value.

3. If your system processes large write operations, see the section on Limiting ZFS Transaction Group Writes. Otherwise, reboot the machine for the change to take effect. Also note that this operation requires Solaris 10 update 4 (08/07) and Nevada (build 51) release or later.

To Limit ZFS Transaction Group Writes

UnboundID has found that the Metrics Engine can exhibit uneven throughput performance during continuous write loads for Oracle Berkeley DB Java Edition backends on ZFS systems. We have found that the ZFS Write Throttle feature stalls write operations when transaction groups are flushed to disk. During these periods, operation throughput can drop significantly with these large I/O bursts.

To smooth out write throughput and improve latency, we recommend setting the `zfs_write_limit_override` property in the `etc/system` file to the size of the available disk cache on the system.

1. Open the `/etc/system` file.

2. Add the following line to the end of the file. Set the value to the size of your onboard cache. For example, for a system that has a 32MB cache per disk, set the following parameter:

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

3. For the change to take effect, reboot the machine. Also note that this operation requires Solaris 10 update 4 (08/08) or later.

ZFS Access to Underlying Disks

Storage requirements vary depending on whether ZFS has access to the underlying disks. If possible, ZFS should be given direct access to the underlying disks that will be used to back the storage. Direct access to the underlying disks makes it possible to configure the system with the greatest degree of reliability and flexibility.

To configure the system, 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 reduces the amount of available space when compared with other configurations, like RAID-Z (ZFS data-parity scheme with full dynamic stripe width) or RAID-Z2 (ZFS dual parity RAID-Z), RAID 1+0 provides dramatically better performance and reliability.

If ZFS cannot get direct access to the underlying disks (for example, the system only has access to a logical unit number, LUN, on a storage area network, SAN), then the provided storage should already include some level of redundancy. Again, the RAID 1+0 configuration is recommended over other schemes like, RAID 5 or RAID 6. If the storage includes redundancy, then the zpool should be created with only that LUN and should not add any additional redundancy. In such a configuration, ZFS is not able to take advantage of its advanced selfhealing capabilities when it detects any corruption at the filesystem level. However, ZFS checksumming can still detect those types of problems.

Configuring ZFS Compression

The ZFS filesystem should have compression enabled to improve performance as it reduces the amount of data that needs to be written or read from the underlying disks. In most cases, the reduced costs of the disk I/O outweighs the CPU cost of compressing and decompressing the data.

The following procedure assumes that the ZFS filesystem is named ds. The changes take effect immediately with no need to reboot or perform any other action.

> **Caution:**
>
> Knowing the actual size of files is useful when you need to: 1) back up files to a non-ZFS filesystem, 2) run a binary copy initialization over the network, or 3) estimate the amount of memory dedicated to caching. On traditional UNIX filesystems, the `du` command reports the sum of all the specified file sizes. However, on ZFS, `du` reports the amount of disk space consumed, which might not equal the sum of the file sizes if features like compression or multiple copies are enabled. Administrators should be aware of this difference when determining the database size using `du`.
>
> Instead of using `du`, UnboundID® Metrics Engine provides a utility, `bin/sum-file-sizes`, that determines the size (in bytes, kilobytes, megabytes, or gigabytes) of the sum of a set of files even if ZFS compression or multiple copies are enabled.

To Configure ZFS Compression

- Turn on ZFS compression by running the `zfs` command.

```
# zfs set compression=on ds
```

Preparing the Operating System (Linux)

The UnboundID® Metrics Engine has been extensively tested on multiple operating systems. We have found that several operating system optimizations lead to improved performance. These optimizations include increasing the file descriptor limit on Linux systems, setting filesystem flushes, editing OS-level environment variables, downloading some useful monitoring tools for Redhat Linux systems, and configuring for Huge Page support.

To Set the File Descriptor Limit (Linux)

The Metrics Engine allows for an unlimited number of connections by default but is restricted by the file descriptor limit on the operating system. Many Linux distributions have a default file descriptor limit of 1024 per process, which may be too low for the server if it needs to handle a large number of concurrent connections.

1. Display the current hard limit of your 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 there is a line that sets the value of the `fs.file-max` property, make sure its value is set to at least 65535. If there is no line that sets a value for this property, 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 sets 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"). Also note that you should insert a tab, rather than spaces, between the columns.

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

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

   ```
   # ulimit -n
   ```

Setting the Filesystem Flushes

With the out-of-the-box settings on Linux systems running the ext3 filesystem, the data is only flushed to disk every five seconds. If the Metrics Engine is running on a Linux system using the `ext3` filesystem, consider editing the mount options for that filesystem to include the following:

```
commit=1
```

This variable changes the flush frequency from five seconds to one second.

About Editing OS-Level Environment Variables

Certain environment variables can impact the Metrics Engine in unexpected ways. This is particularly true for environment variables that are used by the underlying operating system to control how it uses non-default libraries.

For this reason, the Metrics Engine explicitly overrides the values of key environment variables like *PATH*, *LD_LIBRARY_PATH*, and *LD_PRELOAD* to ensure that something set in the environments that are used to start the server does not inadvertently impact its behavior.

If there is a legitimate need to edit any of these environment variables, the values of those variables should be set by manually editing the `set_environment_vars` function of the `lib/_script-util.sh` script. You will need to stop (stop-metrics-engine) and re-start (start-metrics-engine) the server for the change to take effect.

Install sysstat and pstack (Red Hat)

For Red Hat® Linux systems, you should install a couple of packages, `sysstat` and `pstack`, that are disabled by default, but are useful for troubleshooting purposes in the event that a problem occurs. The troubleshooting tool `collect-support-data` uses the `iostat`, `mpstat`, and `pstack` utilities to collect monitoring, performance statistics, and stack trace information on the server’s processes.

To Disable Filesystem Swapping

For all deployments, we recommend disabling 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
```

To Set noatime on ext3 Systems

If you are using an `ext3` filesystem, it is recommended that you set `noatime`, which turns off any *atime* updates during read accesses to improve performance.

- Run the following command:

```
# mount -t ext3 -o noatime /dev/fs1
```

Configuring Huge Page Support (Linux)

We recommend configuring Huge Page support to provide a performance gain for your system of about 5–10 percent. Typically, on Linux systems, memory is managed in page sizes of 4096 bytes per page. RedHat Enterprise Linux Server introduced the concept of *huge pages*, where memory is managed in page sizes of 2M or 1 GB per page. Huge Page support is especially useful for virtualized environments using VMWare. If you are configuring a VMWare-based system, you will need to calculate the memory configurations for your particular system. Follow the recommended VMWare Tuning Guidelines presented on their web site.

As a general guideline, on RedHat Enterprise Linux Server 5.x versions, you should ensure that the size of the huge page is set slightly higher than the maximum size of your JVM settings. For example, for a total system memory of 96 GB, you could set the JVM memory to 80GB, then configure your system to provide ~85GB of Huge Page support.

RedHat Enterprise Linux Server 6.0 or later introduces *Transparent Huge Pages*, which is an abstraction layer that simplifies the management of huge pages. By default, Transparent Huge Page support is enabled on RedHat Enterprise Linux Server 6.0 or later and CentOS 6.0 or later.

To Configure Huge Page Support on Releases Prior to Redhat Enterprise Linux Server 6.0

1. Log in as root. For this example, we assume you are using at least Redhat Enterprise Linux Server 5.5.

2. Verify that your kernel supports huge pages. If the contents of `/proc/meminfo` contains "HugePage_Total," "HugePages_Free" or "HugePagesize," then your kernel supports huge pages. Make note of the huge page size of your system, which is dependent on your system architecture.

   ```
   $ cat /proc/meminfo | grep Huge
   HugePages_Total: 0
   HugePages_Free:  0
   HugePages_Rsvd:  0
   HugePagesize: 2048 kB
   ```

3. Set the maximum amount of memory on the server. For 64-bit JVMs that support huge pages, you need to set the kernel for shared memory to be slightly higher than the maximum size of your JVM. This memory is "pinned" or reserved for the application once the JVM is started. For this example, set the maximum shared memory to 85 GB. In the `/etc/sysctl.conf` file, you need to add a line as follows:

   ```
   kernel.shmmax = <number of bytes>
   ```

   For example, for 85 GB, add the line:

   ```
   kernel.shmmax = 91268055040
   ```

4. Next, set a virtual memory kernel parameter to tell the OS how many huge pages you want to set aside. In the `/etc/sysctl.conf` file, you need to add the following line:

   ```
   vm.nr_hugepages = <number of pages>
   ```

   For example, we want to set the virtual memory to 85 GB (89128960 kB), which is 85 GB/ 2 MB (obtained in step 2 as the size of each huge page). Thus, 89128960 kB/2048 kB = 43520, which is the number of huge pages we want to reserve. This setting only takes effect at boot time.

   ```
   vm.nr_hugepages = 43520
   ```

   To set it without reboot, run one of the following commands:

   ```
   sysctl -w vm.nr_hugepages=43520
   ```

   If you want the setting to be present after reboot, then you have to modify the `/etc/sysctl.conf` file.

5. Reboot the machine. Repeat step 2 to ensure that huge page memory is configured.

   ```
   $ cat /proc/meminfo | grep Huge
   HugePages_Total: 44564
   HugePages_Free:  44564
   HugePages_Rsvd:  0
   HugePagesize: 2048 kB
   ```

6. Finally, go to the Metrics Engine root. Assuming you are using Sun JVM, set the *-XX:+UseLargePages* JVM option in the Metrics Engine’s `config/java.properties` file for the

`start-ds` tool and `import-ldif` tools. Note that each command should be on a single line. The command options are listed on separate lines for readability purposes.

```
# These JVM arguments can be used to run the Directory Server with an
# aggressive memory tuning:

start-ds.java-args=-d64 -server -Xmx80g -Xms80g -XX:+UseConcMarkSweepGC
-XX:+CMSConcurrentMTEnabled -XX:+CMSParallelRemarkEnabled
-XX:+CMSParallelSurvivorRemarkEnabled -XX:+CMSScavengeBeforeRemark
-XX:RefDiscoveryPolicy=1 -XX:ParallelCMSThreads=1
-XX:CMSMaxAbortablePrecleanTime=3600000 -XX:CMSInitiatingOccupancyFraction=80
-XX:+UseParNewGC -XX:+UseMembar -XX:+UseBiasedLocking -XX:+UseCompressedOops
-XX:PermSize=64M -XX:+HeapDumpOnOutOfMemoryError -XX:+UseLargePages

# These JVM arguments can be used to do an offline LDIF import with an aggressive
# memory tuning:
import-ldif.offline.java-args=-d64 -server -Xmx80g -Xms80g
-XX:+UseParallelGC -XX:+UseMembar -XX:NewRatio=8 -XX:+UseNUMA
-XX:+UseCompressedOops -XX:+UseNUMA -XX:+HeapDumpOnOutOfMemoryError
-XX:+UseLargePages
```

7. On the Metrics Engine, run the `dsjavaproperties` tool to save the JVM settings.

```
$ bin/dsjavaproperties
```

You have successfully set up Huge Page Support on your Linux system.

Running as a Non-Root User

On systems running Solaris 10 and OpenSolaris, you can use the User and Process Rights Management subsystem with the Role-Based Access Control (RBAC) mechanisms to grant users or roles only the privileges necessary to accomplish a specific task. Using RBAC avoids the assignment of full super-user (root) privileges to the user. For example, you can grant the `net_privaddr` privilege to a non-root user, or role, that gives him or her the ability to listen on privileged ports (for example, on ports 1024 or below). Similarly, granting the `sys_resource` privilege allows a user to bypass restrictions on resource limits, such as the number of file descriptors a process might use.

The Solaris User and Process Rights Management system can also be used to remove capabilities from users. For example, removing the `proc_info` privilege from a user prevents the user from seeing processes owned by other users. Removing the `file_link_any` privilege can prevent users from creating hard links to files owned by other users. Hard links are not needed by the Metrics Engine and can represent a security risk under certain conditions. The following table summarizes the Solaris privileges that you may want to assign to non-root users.

Privilege	Description
net_privaddr	Provides the ability to listen on privileged network ports.
sys_resource	Provides the ability to bypass restrictions on resource limits (including the number of available file descriptors).
proc_info	Provides the ability for users to see processes owned by other users on the system. This privilege is available to all users by default, but it can pose a security risk in some cases. UnboundID recommends that it be removed from the role used by the Metrics Engine.
file_link_any	Provides the ability to create hard links to files owned by other users on the system. This privilege is available to all users by default, but it can pose a security risk in some cases. UnboundID recommends that it be removed from the role used by the Metrics Engine.

Running as a Non-Root User (Linux)

Linux systems do not provide a direct analog to the Solaris User and Process Rights Management subsystems. As a result, there is no easy way to allow a non-root user to listen on a privileged port.

To run as a non-root user but still allow connections on a privileged port, two options are available:

- **Use a Load-Balancer or Proxy Server**. In many environments, the server can be run on a non-privileged port but can be hidden by a hardware load-balancer or LDAP proxy server.
- **Use `netfilter`**. The `netfilter` mechanism, exposed through the `iptables` command, can be used to automatically redirect any requests from a privileged port to the unprivileged port on which the server is listening.

Creating a Solaris Role

To give multiple administrators access to the Metrics Engine, UnboundID® Metrics Engine recommends that a Solaris role be created to run the server and that all necessary administrators be added to that role. The Solaris role provides an audit trail that can be used to identify which administrator performed a given action, while still allowing administrators to run the server, to view and edit files used by the server, and to execute commands as that same user. As with normal user accounts, roles can be assigned privileges. The role used for the Metrics Engine should include the `net_privaddr` and `sys_resource` privileges and should exclude the `proc_info` and `file_link_any` privileges for improved security (that is, to eliminate the need for root access).

To Create a Solaris Role for Multiple Administrators

To give multiple administrators access to the Metrics Engine, UnboundID® Metrics Engine recommends that a Solaris role be created to run the server and that all necessary administrators be added to that role. The Solaris role provides an audit trail that can be used to identify which administrator performed a given action, while still allowing administrators to run the server, to view and edit files used by the server, and to execute commands as that same user. As with normal user accounts, roles can be assigned privileges. The role used for the Metrics Engine should include the `net_privaddr` and `sys_resource` privileges and should exclude the `proc_info` and `file_link_any` privileges for improved security (that is, to eliminate the need for root access).

1. Create a Solaris role. Assume the role is named `ds` with all of the appropriate privileges needed to run the Metrics Engine. Make sure to enter the whole command on a single line.

```
# roleadd -d /export/home/ds -m -s /usr/bin/bash \
  -K defaultpriv=basic,net_privaddr,sys_resource,-prov_info,-file_link_any ds
```

2. Assign a password.

```
# passwd ds
```

3. For each administrator who is allowed to manage the Metrics Engine, assign the role with the `usermod` command. For example, to give someone with a user name of “john” the ability to assume the `ds` role, issue the following command:

   ```
   # usermod -R ds john
   ```

   If a user is already a member of one or more roles, then the entire list of existing roles, separated by commas, must also be provided or the user will be removed from those roles. For example, if the root account is also a role and the user “john” is also a member of that role, then the command would be:

   ```
   # usermod -R root,ds john
   ```

4. Log in using a normal user account and then use the `bin/su` command to assume the role created for the Metrics Engine. You cannot log directly into a system as a role. Only users that have been explicitly assigned to a role will be allowed to assume it.

Installation Process Overview

The process for setting up and installing a Metrics Engine involves the following steps:

- Configuring the UnboundID Directory, Proxy, and Synchronization Servers.
- Setting up the PostgreSQL database.
- Unzipping and installing the Metrics Engine using the setup tool.
- Configuring the Metrics Engine using the `monitored-servers` tool.

The remainder of this chapter describes each of these steps in detail.

Directory Server Configuration

Before you install the Metrics Engine, you need to configure your UnboundID Directory Server, UnboundID Directory Proxy Server, and UnboundID Synchronization Server to provide it information. The Metrics Engine requires all servers to be version 3.5.0 or later.

Once you have installed the directory server, you will use the `dsconfig` tool to make configuration changes for the Metrics Engine. When using the `dsconfig` tool interactively, set the complexity level to Advanced so that you will be able to make all the necessary configuration changes.

Configuring the Metrics Backend

The Metrics Backend manages the storage of metrics and provides access to the stored blocks of metrics via LDAP. The Metrics Backend is configured to keep a maximum amount of metric history based on log retention policies. The default retention policy uses the Default Size Limit Retention Policy, Free Disk Space Retention Policy, and the File Growth Limit Policy, limiting the total disk space used to 500 MB. This amount of disk typically contains more than 24 hours of metric history, which is ample. The Directory Server keeps a metric history so that the Metrics Engine can be down for a period and then catch up when it comes back online.

The following two commands create a Retention Policy that limits the number of files to 2000, and sets the Metrics Backend to flush data to a new file every 30 seconds.

```
$ bin/dsconfig create-log-retention-policy \
  --policy-name StatsCollectorRetentionPolicy \
  --type file-count --set number-of-files:2000

$ bin/dsconfig set-backend-prop \
  --backend-name metrics --set sample-flush-interval:30s \
  --set retention-policy:StatsCollectorRetentionPolicy
```

These commands configure the Metrics Backend to keep 16 hours of metric history, which consumes about 250 MB of disk, ensuring that captured metrics are available to the Metrics Engine within 30 seconds of when the metric was captured. The value of the `sample-flush-interval` attribute 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 seconds 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 the Dashboard Application. If you change the `sample-flush-interval` attribute to 60 seconds in the example above, then the Directory Server keeps 2000 minutes of history. Because the number of metrics produced per unit of time can vary depending on the configuration, no exact formula can be used to compute how much storage is required for each hour of history. However, 20 MB per hour is a good estimate.

Configuring the Processing Time Histogram Plugin

The Processing Time Histogram plugin is configured on each directory and proxy server as a set of histogram bucket ranges. When the bucket ranges for a histogram change, the Metrics Engine notices the change and marks samples differently. This process allows for histograms with the same set of bucket definitions to be properly aggregated and understood when returned in a query. If different servers have different bucket definitions, then a single metric query cannot return histogram data from the servers.

You should try to keep the Processing Time Histogram bucket definitions the same on all servers. Having different definitions restricts the ability of the Metrics Engine API to aggregate histogram data across servers and makes the results of a query asking "What percentage of the search requests took less than 12 milliseconds?" harder to understand.

For each server in your topology, you must set the `separate-monitor-entry-per-tracked-application` property of the processing time histogram plugin to true. This property must be set to expose per-application monitoring information under `cn=monitor`. When the `separate-monitor-entry-per-tracked-application` property is set to true, then the `per-application-ldap-stats` property must be set to `per-application-only` on the Stats Collector Plugin and vice versa.

For example, the following `dsconfig` command line 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 `dsconfig` command line sets the `per-application-ldap-stats` property of the Stats Collector plugin to `per-application-only`:

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

Setting the Connection Criteria to Collect SLA Statistics by Application

If you want to collect data about your SLAs, you need to configure connection criteria for each Service Level Agreement that you want to track. The connection criteria are used in many areas within the server. They are used by the client connection policies, but they can also be used when the server needs to perform matching based on connection-level properties, such as filtered logging. For assistance using connection criteria, contact your authorized support provider.

Note: UnboundID provides an unsupported sample application called the SLA Viewer, which provides a real-time view of the throughput and response times associated with each of your SLAs. This application is available through the UnboundID Labs web site: https://www.unboundid.com/labs/.

For example, imagine that we are interested in collecting statistics on data that is accessed by clients authenticating as the Directory Manager. We need to create connection criteria on the directory server that identifies any user authenticating as the Directory Manager. The connection criteria name corresponds to the `application-name` dimension value that clients will specify when accessing the data via the API. When you define the Connection Criteria, change the `included-user-base-dn` property to include the Directory Manager’s full LDIF entry.

The following `dsconfig` command line creates connection criteria for the Directory Manager:

```
$ bin/dsconfig create-connection-criteria \
  --criteria-name “Directory Manager” \
  --type simple \
  --set “included-user-base-dn:cn=Directory Manager,cn=Root DNs,cn=config”
```

Updating the Global Configuration

You also need to create Global Configuration-tracked applications for each app (connection criteria) you intend to track. The `tracked-application` property allows individual applications to be identified in the server by connection criteria. The name of the tracked application is the same as the name you defined for the connection criteria.

For example, the following `dsconfig` command line adds the connection criteria we created in the previous step to the list of tracked applications:

```
$ bin/dsconfig set-global-configuration-prop \
  --set "tracked-application:Directory Manager”
```

The value of the `tracked-application` field corresponds to the value of the `application-name` dimension value that clients will specify when accessing the data via the API.

Setting Up the Database

The Metrics Engine uses the PostgreSQL DBMS database to store and aggregate server data. Use PostgreSQL version 9.1 or later. Install the PostgreSQL database on the same host machine where the Metrics Engine will be hosted.

Get the PostgreSQL installer that is native for your platform. For Debian, you would use a `.deb` installer. For RedHat, you use an `.rpm` installer. The package installers create the DBMS user and set file permissions correctly. The `.zip` and `.bz2` distributions do not.

In the following procedures, you will use two different types of users, system users and DBMS users. A system user can log in to the operating system, typically the `root` user. The DBMS user can access the DBMS database and is the `postgres` user. These groups do not share password information.

About Setting Up the PostgreSQL DBMS Database

The PostgreSQL DBMS imposes its organizational structure on all users of the DBMS:

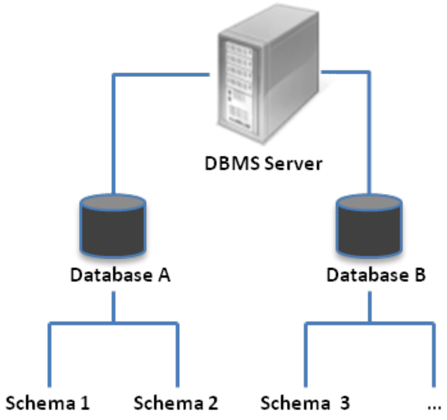


Figure 3: PostgreSQL DBMS Organizational Structure

A PostgreSQL DBMS server is a set of cooperating PostgreSQL processes listening on a single TCP port. The DBMS server is partitioned into one or more databases, and each database is partitioned into one or more schema. Tables are associated with exactly one schema, and indexes are associated with exactly one table. A user (or role, as they are referred to in PostgreSQL) is associated with a database and can be granted access to one or more schema within a database. When a user authenticates to the DBMS server, the resulting session is automatically assigned to a default schema, where each user can specify their own default schema.

The Metrics Engine treats the schema and the users as the same logical idea, creating a schema and a user with the same name and assigning the schema as the user's default schema. This is done for the sake of simplicity, not necessity. However, the Metrics Engine requires that the user connecting to the DBMS have the Metrics Engine Schema as the default schema.

Because the PostgreSQL DBMS server runs by default as the user `postgres`, and the Metrics Engine runs by default as a different user, the Metrics Engine process cannot directly read or modify the PostgreSQL database files, logs or configuration data. This can be an issue when diagnosing problems, because the `collect-support-data` tool cannot typically capture all of the data needed to diagnose some PostgreSQL problems.

By default, the PostgreSQL installer allows PostgreSQL to listen to requests on the loopback interface only, using port 5432. This configuration is secure because it restricts DBMS clients to processes running on the same host as the DBMS server. The Metrics Engine therefore requires that the DBMS server be on the same physical machine. If a DBMS client can connect to port 5432 on the loopback address, then it will need to provide user authentication sufficient to satisfy the configuration specified in the `pg.hba.conf` file. This file limits clients based on the following filters:

```
 socket-type     database-name     user-name    client-address  auth-method
```

The Metrics Engine uses the following filter:

```
host           all               all          127.0.0.1/32    md5
```

This filter allows the client to connect to any database with any username via the TCP loopback interface. It requires the user to provide the password that is stored by PostgreSQL in its credential table.

However, the DBMS may be vulnerable to attacks from other processes on the same server. Since the DBMS is only listening on the loopback address, any attack must originate from the same host. If the Metrics Engine and PostgreSQL DBMS are the only processes on this host other than OS processes, then the security risk is greatly diminished. And even if an attacker gets access to the DBMS directly, the Metrics Engine DBMS only contains performance metrics about the operations of the monitored servers and basic configuration information; none of the data stored in the userRoot backend is present in the DBMS An attacker could modify histogram sample data and alter performance history but could not affect directory data. Finally, even with LDAP hostname and port information for the monitored servers, an attacker would still need to overcome LDAP authentication, and no credential data is stored in the DBMS.

So that the `collect-support-data` tool can collect information related to the PostgreSQL RDBMS, you may want to allow the user that runs the Metrics Engine process access to PostgreSQL through `psql`. If so, a new PostgreSQL user with the same name as the Metrics Engine user needs to be created, and a `.pgpass` file containing the DBMS user's password needs to be created in the Metrics Engine user's home directory. This Metrics Engine DBMS user needs to be assigned the same default schema.

About the Metrics Engine DBMS Users

The Metrics Engine uses three different DBMS users:

- **`postgres` -** the default PostgreSQL DBA user, used during Metrics Engine setup only.
- **`metricsengine` -** the user created during Metric Engine setup. You can change the name of this user during setup. This name represents both the DBMS user name and the schema name.
- **`<engine-user>` -** the operating system username that owns the Metrics Engine files. This username also executes Metrics Engine configuration using the `dsconfig` command.

The `postgres` DBMS user is created by the PostgreSQL installer. The instructions below stipulate that you must set a password for this user in PostgreSQL. Without this password, the Metrics Engine installer cannot authenticate as this DBA user and cannot perform the necessary DBMS configuration.

The `metricsengine` DBMS user is created by the Metrics Engine at setup. The name is `metricsengine` by default, but can be changed during setup. This Metrics Engine uses this DBMS user to authenticate during normal operations. This user owns all of the Metrics Engine DBMS tables.

The Metrics Engine setup uses the `postgres` DBMS user to create the following plus all of the tables held within the schema:

- `metricsengine` user
- `metricsengine` database
- `metricsengine` schema

If setup detects that the `metricsengine` user or schema already exists, it bypasses the creation of these items, allowing you to run setup again and preserve an existing Metrics Engine DBMS. However, you must be sure that the existing Metrics Engine DBMS has a valid configuration. Setup only checks if the items exist. If the `metricsengine` schema already exist but have different tables, the Metrics Engine will fail to run. If setup detects an existing schema exists, it asks if you want to delete and recreate the schema.

The <*`engine-user`*> DBMS user is created manually later in this guide to enable the `collect-support-data` tool to execute `psql` scripts when it collects support information. The <*`engine-user`*> DBMS user is not required and `collect-support-data` will work even if it does not exist. In this case, all of the data that support needs might not be collected and you may need to run `psql` commands directly to capture the rest of the support data.

The Metrics Engine authenticates to the DBMS using the `metricsengine` user over the loopback TCP interface. All DBMS user authentication uses password data that is stored in the PostgreSQL DBMS tables themselves.

Note: The operating system password, if any, is not used by PostgreSQL. So, you must remember the `postgres` DBA DBMS user password. Having administrative privileges on the operating system will not help you if you forget a DBMS password and do not know the DBMS DBA password.

Installing and Configuring PostgreSQL on Centos/RHEL

1. As root, run rpm to install the package.

   You can download the rpm from http://yum.postgresql.org/repopackages.php.

   ```
   root# rpm -i pgdg-centos91-9.1-4.noarch.rpm
   ```

2. Use the Yellow Dog Update Manager (yum), a remote package management system, to install the two packages.

   The first package contains tools for administering the database, the second contains the server.

   ```
   root# yum install postgresql91-contrib.x86_64
   root# yum install postgresql91-server.x86_64
   ```

3. Run the `service` command as root to initialize the database.

   ```
   root# service postgresql-9.1 initdb
   ```

4. Start the server.

   ```
   root# /etc/init.d/postgresql-9.1 start
   ```

Configuring Administrative Credentials for a PostgreSQL database

The following procedure describes how to manually create the PostgreSQL database administrator account. You can specify that this account be created automatically when you install the Metrics Engine using the setup command.

1. Change from the root user to the PostgreSQL user.

   ```
   root# su - postgres
   ```

2. Run the `psql` tool, a PostgreSQL tool used for sending SQL commands, as `postgres` user.

   ```
   postgres$ psql
   ```

   If this step fails, you may need to start the PostgreSQL server first as follows:

   ```
   root# /etc/rc.d/init.d/postgresql-9.1 start
   ```

3. Assign a password to the `postgres` DBA DBMS user that you will use with the Metrics Engine `setup` tool.

```
\password
Enter new password:
Enter it again:
```

Note: Once you have set the DBA DBMS user password, do not forget it. Even the root user cannot change this password without knowing the old password.

Locating PostgreSQL Configuration Files

When the PostgreSQL database is running, you can locate the configuration files by running the following commands:

```
postgres$ echo "show all;" | psql | grep config_file

postgres$ echo "show all;" | psql | grep hba_file
```

These commands issue the `'show all'` query to the DBMS via the `psql` command and filter the results for the expected property. The results will resemble the following, where the path `/var/lib/pgsql/9.1/data/postgresql.conf` is the path to the configuration file.

```
postgres$ echo "show all;" | psql -d monserver | grep config_file
 config_file | /var/lib/pgsql/9.1/data/postgresql.conf | Sets the server's main
 configuration file.
```

Configuring PostgreSQL Connections

As the superuser, you need to edit the pg_hba.conf file to allow the new superuser to access the database. This file is located by default in the `/var/lib/pgsql/9.1/data` directory. The following lines will also enable the Metrics Engine user access to the database once setup has created this user. Using a text editor, edit the pg_hba.conf file.

1. Comment out the following lines by appending a hash mark (#).

```
# host    all        all     127.0.0.1/32      ident
# host    all        all     ::1/128           ident
```

2. Add the following lines.

```
host     all         all     127.0.0.1/32      md5
```

Enabling psql for the Metrics Engine User

If you experience problems with the Metrics Engine, you will need to run the `collect-support-data` tool. This tool captures a great deal of configuration information about the server as well as PostgreSQL. However, because the Metrics Engine users and the PostgreSQL user are different by design, the Metrics Engine user does not automatically have access to the DBS via the PSQL tool. As a result, some information is omitted from the `collect-support-data` output.

- To allow the Metrics Engine user to use `psql`, create a DBMS user for the <engine-user> operating system account, which runs the Metrics Engine.

  ```
  root# createuser --superuser --pwprompt <engine-user>
  ```

Configuring the Timezone and Routine Vacuuming

This section describes configuring the timezone and routing vacuuming. You make these changes to the `postgres.conf` file on Centos or the `postgresql.conf` file on Solaris.

With data partitioning in place, the PostgreSQL DBMS evaluates the TIMESTAMP fields according to an absolute timezone. The Metrics Engine expects the timezone to be set to GMT. You need to put the PostgreSQL DBMS in the GMT timezone by setting timezone to GMT in the postgres.conf file.

The PostgreSQL vacuum command recovers disk space occupied by updated or deleted rows. PostgreSQL contains a separate optional server process called the autovacuum daemon, which automates the execution of this command. When enabled, it runs periodically and checks for tables that have had a large number of inserted, updated or deleted entries. You need to ensure that this autovacuum daemon is enabled by setting autovacuum to on in the `postgres.conf` file. Without this change, the disk usage of the DBMS will grow without bound, filling up the file system.

1. Configure the timezone by adding the following line to the `postgres.conf` or `postgresql.conf` file:

   ```
   timezone = 'GMT'
   ```

2. Configure routine vacuuming by adding the following line to the `postgres.conf` or `postgresql.conf` file:

   ```
   autovacuum = on
   ```

3. Once you have finished making your changes, restart the server as the superuser:

   ```
   root# /etc/init.d/postgresql-9.1 restart
   ```

Tuning the PostgreSQL Configuration

You configure PostgreSQL using the `postgresql.conf` configuration file described earlier. Based on the size of the monitored installation and the RAM available to PostgreSQL, you may want to modify several configuration variables. The following table provides some guidelines to help you tune your PostgreSQL configuration:

Table 2: Hardware Requirements

Variable Name	< 4 GB	< 8 GB	< 16 GB	< 32 GM
`shared_buffers`	512 MB	1 GB	2 GB	2 GB
`maintenance_work_mem`	1 GB	2 GB	4 GB	4 GB

Installing the Metrics Engine

Use the `setup` tool for initial setup of the Metrics Engine.

Unpacking the Installation Packages

To begin the installation process, you can obtain the latest zip release bundle from UnboundID and unpack it in a folder of your choice. In this example, the release bundle unpacks in the `UnboundID-Metrics-Engine` directory.

Unpack the release bundle as follows:

```
$ unzip UnboundID-Metrics-Engine-3.6.0.0.zip
```

You can now install the Metrics Engine.

To Install the Metrics Engine

1. Login as the user you want the Metrics Engine process to run as. Go to the installation directory if you are not already there.

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

2. Use the `setup` command with the appropriate JAVA_HOME environment variable.

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

   Note: If your `JAVA_HOME` environment variable is set to an older version of Java, you must explicitly specify the path to the Java JDK installation during setup. You can either set the `JAVA_HOME` environment variable with the Java JDK path or execute the `setup` command in a modified Java environment using the `env` command.

3. Read the UnboundID End-User License Agreement. If you agree to its terms, type `yes` to continue.

4. The tool describes the installation process and the prerequisites. Type `yes` to continue.

5. Type the port number of your local PostgreSQL instance or press **Enter** to accept the default port, which is 5432.

6. Setup creates the Metrics Engine specific database, schema and role (user). If you answer `yes`, setup will attempt to create the database, schema and role, including dropping any existing schema that uses the same name.

If you answer `no`, the Metrics Engine specific database, schema, and role must already exist. Setup prompts for the DBA credentials and the desired Metrics Engine credentials in the next few steps.

7. Enter the name and password for the administrative account configured when installing PostgreSQL, `postgres` by default.

8. Next, enter the name and credentials for the user account the Metrics Engine uses to connect to the database. By default, the account name is `metricsengine`. If you want to change the password, select `yes`

9. Enter the name of the PostgreSQL database where the Metrics Engine will store its data. By default, the name is `metricsengine`.

10. Type the root user DN, or press **Enter** to accept the default (cn=Directory Manager), and then type and confirm the root user password.

11. If you want to enable support for HTTP clients, enter `1`. If you want to enable HTTPS support, enter `2`. To enable both, select `3`.

    Enter the port number or numbers depending upon the type of HTTP support you select, or press **Enter** to accept the defaults.

12. Type the LDAP port number of your Metrics Engine, or press **Enter** to accept the default port, 389.

Note: The Metrics Engine process needs special privileges to listen on a port less than 1024.

13. Type `yes` to enable LDAPS. Otherwise, press **Enter** to accept the default value of `no`.

    If you answered yes, you will be prompted for certificate options. If you use the Java or the PKCS#12 key store, you will be asked for the key store path, and the key store PIN. If you use the PKCS#11 token, you will be asked for only the key PIN.

14. Type `yes` to enable StartTLS. Otherwise, press **Enter** to accept the default value of `no`. As in the previous step, if you answered `yes`, you will be prompted for certificate options.

15. If you want to specify a particular address on which the server listens for client connections, enter `yes`. Otherwise, accept the default of `no`.

16. Enter whether you want to tune the JVM to maximize memory use. By default, the value is `no`.

17. Type `yes`, or press **Enter** to accept the default to start the Metrics Engine after the configuration has completed.

    If you plan to configure additional settings or import data, you can type `no` to keep the server in shutdown mode.

18. When you have finished entering your settings, press `1` to configure the Metrics Engine.

Configuring the Servers Monitored by the Metrics Engine

Once you have finished running the setup tool, you need to configure the servers monitored by the Metrics Engine. You can configure the set of monitored servers using the `monitored-servers` tool or configure them individually using `dsconfig`.

About the monitored-servers Tool

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

This tool may be run against the same external server repeatedly, meaning that the server can go through the preparation process again to update the user account or password.

When you run the tool with the `add-servers` subcommand, it 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.

About Adding Individual Servers Using dsconfig

Use the `dsconfig` tool to configure individual servers to be monitored by the Metrics Engine. Only the servers that you specify in the `monitored-server` property of the Monitoring Configuration configuration object will be actively monitored, though historical data may exist for disabled servers. If you want to temporarily disabl e monitoring and stop the Metrics Engine from collecting statistics, remove the external server from this property. Do not delete the external server object. Add the external server back to the `monitored-server` property when you are ready to re-enable monitoring of the server.

To Configure the Servers Monitored by the Metrics Engine

1. 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. The engine 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.

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

2. Use the `--dry-run` option so that the tool generates 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 --dry-run
```

To Add Individual Monitored Servers Using dsconfig

1. Run the `dsconfig` tool.

```
$ bin/dsconfig
```

2. Select `Monitoring Configuration` to edit the Metrics Engine configuration.

3. Edit the monitored-server property, then enter `2` to add a new server. Only servers specified in this property are monitored.

4. Create a new LDAP external server, and then select they type of server you want to create from the list.

5. Enter the name of the new server.

6. Enter the host name of the server you will monitor. For example, server.example.com.

7. Specify the DN used to bind to the target LDAP server. Enter `5` to specify the password used to bind to the server.

8. When you are satisfied with the properties of the external server, enter `f` to create the new external server.

9. When you have finished adding servers to be monitored, enter `1` to accept the new values

10. Enter `f` when you have finished making changes.

Running the Metrics Engine

To start the Metrics Engine, run the `bin/start-metrics-engine` command on UNIX® or Linux systems. Run the `bat/start-metrics-engine` command on Windows systems. The `start-metrics-engine` command starts the Metrics Engine as a background process when no options are specified. To run the Metrics Engine as a foreground process, use the `start-metrics-engine` command with the `--nodetach` option.

Starting the Metrics Engine

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.

To determine if your server is ready to respond to metric queries, you can check the Sample Import Backlog using the `status` tool. The following output shows that the server is available, because the Sample Import Backlog is zero:

```
-- Server Status ---
Server Run Status:    Started 22/Aug/2012:10:35:36.000 -0500
Operational Status:   Available
Open Connections:     2
Sample Import Backlog: 0
Sample Import Delay (ms): 7
Max Connections:      2
Total Connections:    1188

          --- Server Details ---
Host Name:            host.example.com
Administrative Users: cn=Directory Manager
Installation Path:    /UnboundID-Metrics-Engine
Server Version:       UnboundID Metrics Engine 3.6.0.0
Java Version:         1.6.0_31
```

Once the server’s Sample Import Backlog is relatively low compared to the number of servers being monitored (no more than five times the number of monitored server), it can answer metric queries for recent data on a particular server, and you can begin to analyze the data.

To Start the Metrics Engine as a Background Process

- Go to the installation directory, and then use `start-metrics-engine`.

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

To Start the Metrics Engine as a Foreground Process

1. Type `start-metrics-engine` to launch the Metrics Engine as a foreground process.

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

2. You can stop the Metrics Engine by pressing **Ctrl+C** in the terminal window where the server is running or by running the `stop-metrics-engine` utility from another window.

To Start the Metrics Engine at Boot Time

By default, the Metrics Engine does not start automatically when the system is booted. Instead, you must manually start it with the `bin/start-metrics-engine` command. To configure the monitoring server to start automatically when the system boots, use the `create-rc-script` tool to create a run control (RC) script as follows:

1. Create the startup script.

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

2. 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 ensure that the server is started) and `/etc/rc0.d` directory (starting with a "K" to ensure that the server is stopped).

```
# 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
```

3. Log out as root, and re-assume the `ds` role if you are on a Solaris system.

To Stop the Metrics Engine

Change to the installation directory and use `stop-metrics-engine`.

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

To Restart the Metrics Engine

You can restart the Metrics Engine using the `stop-metrics-engine` command with 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 and avoid any changes to the JVM options.

Go to the installation directory. Using a loopback interface, run the `stop-metrics-engine` command with the `-R` or `--restart` options.

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

Installing the Metrics Engine Management Console

The UnboundID® Metrics Engine provides a graphical web application tool, the UnboundID Metrics Engine Management Console. The Metrics Engine Management Console provides configuration and schema management functionality in addition to monitoring and server information. Like the `dsconfig` configuration tool, all changes made using the Metrics Engine Management Console are recorded in `logs/config-audit.log`. In addition, anytime a configuration is made to the system, the configuration backend is automatically updated and saved as gzip-compressed files. You can access the changes in the `config/archived-configs` folder.

The Metrics Engine Management Console is a web application that must be deployed in a servlet container that supports the servlet API 2.5 or later. An installation using Apache Tomcat is described below for illustration purposes only.

Note: The Metrics Engine Management Console supports JBoss 7.1.1 or later. Refer to the JBoss Compatibility section in the `WEB-INF/web.xml` file for specific configuration steps.

To Install the Metrics Engine Management Console Out of the Box

1. Download and install the servlet container. For example, download `apache-tomcat-<version>.zip` from http://tomcat.apache.org/, and then unzip this file in a location of your choice.

   Note:

2. Set the appropriate Apache Tomcat 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 Metrics Engine Management Console ZIP file, `metrics-web-console-<version>.zip` and unzip the file on your local host. You should see the following files:

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

4. Create a `metricsengconsole` directory in `apache-tomcat-<version>/webapps/metricsengconsole`. Then, copy the `dsconsole.war` file to `apache-tomcat-<version>/webapps/metricsengconsole`.

   ```
   $ mkdir apache-tomcat-<version>/webapps/metricsengconsole
   $ cp dsconsole.war apache-tomcat-<version>/webapps/metricsengconsole
   ```

5. Go 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 metricsengine.war.war
   ```

6. Optional. Edit the `WEB-INF/web.xml` file to point to the correct Metrics Engine instance. Change the host and port to match your server. The parameters in the `web.xml` file appear between <!-- and --> as comments. Uncomment the parameters you need to use. For example, you can specify the server or servers that the console uses to authenticate using the following parameters:

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

Note: If the `ldap-servers` parameter is left as-is (i.e., undefined by default), the web console displays a form field for the user to enter the server host and port.

7. Optional. With the default configuration, Tomcat will time out sessions after 30 minutes of inactivity, forcing the user to log back in again. This can be changed on a servlet container wide basis by editing `apache-tomcat-<version>/conf/web.xml`, and updating the value of this configuration parameter:

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

   The session expires after the specified number of minutes. Changing the value to 120, for example, will extend the expiration to two hours. Changes to this setting might not take effect until the servlet container is restarted, so consider changing the value before starting the server for the first time.

8. Start the Metrics Engine if it is not already running, and then start the Metrics Engine Management Console using the `apache-tomcat-<version>/bin/startup.sh` script. Use `shutdown.sh` to stop the servlet container. (On Microsoft Windows, use `startup.bat` and `shutdown.bat`.) Note that the *JAVA_HOME* environment variable must be set 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
   ```

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

Note: If you re-start the Metrics Engine, you must also log out of the current Metrics Engine Management Console session and then log back in to start a new console session.

Logging into the Metrics Engine Management Console

To log into the console, you can either use a DN (for example, cn=Directory Manager) or provide the name of an administrator, which is 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
```

To Log into the Metrics Engine Management Console

1. Go to the installation 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 Metrics Engine Management Console, click **Configuration**.

7. View the Configuration menu. By default, the console displays the Basic object type properties. You can change the complexity level of the object types using the **Object Types** drop-down list.

Fine-Tuning the Metrics Engine Management Console

The Metrics Engine Management Console uses a `web.xml` descriptor file for its configuration and deployment settings. Instead of specifying the host name and port on the Login page, you can configure one or more primary servers in the `web.xml` file as well as configure security and truststore settings for your Metrics Engine console. If you specify any servers using the `web.xml` file, the Login page will no longer display the LDAP Server field. It will automatically attempt to connect to the primary server(s) specified in the `web.xml` file in the order in which they are specified until one of the servers can authenticate the username and password. The console also uses this server to "discover" other servers in the topology, making them available for monitoring and management in the console.

To Configure One or More Primary Servers for the Console

1. Open the `metricsengconsole/WEB-INF/web.xml` file in a text editor to specify the server(s) that the console uses to authenticate. First, remove the comment tags (<!-- and -->) in the `ldap-servers` section.

2. Next, specify the servers as `host:port` (e.g., server1.example.com:389) or using the LDAPS protocol to specify security information (e.g., ldaps://server1.example.com:389). If you specify more than one server, you must separate them using a space. For example, if you

have two servers: one using standard LDAP communication, the other using SSL, you would see 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.

To Configure SSL for the Primary Console Server

You can configure the console so that it will communicate with all of its primary servers over SSL or StartTLS. See the previous section on how to specify one or more primary servers.

1. Open the `metricsengconsole/WEB-INF/web.xml` file in a text editor to specify the type of communication to authenticate. First, remove the comment tags (<!-- and -->) in the security section.

2. Specify `none`, `ssl`, or `starttls` for the type of security that you are using to communicate with the Metrics Engine.

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

3. Save the file.

To Configure a Truststore for the Console

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

1. Open the `metricsengconsole/WEB-INF/web.xml` file in a text editor to specify the truststore. First, remove the comment tags (<!-- and -->) in the truststore section.

2. Specify the path to your truststore.

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

3. Next, 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.

Upgrading the Metrics Engine Management Console

You can easily upgrade the Metrics Engine Management Console by first moving the `web.xml` file to another location, unpacking the latest Metrics Engine Management Console distribution, and then replacing the newly deployed `web.xml` file with the previous build.

To Upgrade the Metrics Engine Management Console

1. Shut down the console and servlet container.

2. In the current deployment of the Metrics Engine Management Console, move the `webapps/metricsengconsole/WEB-INF/web.xml` file to another location.

3. Download and deploy the latest version for the Metrics Engine Management Console. Follow steps 2–5 outlined in the section "To Install the Console Out of the Box".

4. Assuming you had not renamed the `.war` file when you originally deployed the Metrics Engine Management Console, run a diff between the previous and newer version of the web.xml file to determine any changes that should be applied to the new web.xml file. Make those changes to the new file, and then replace the newly deployed Metrics Engine Management Console’s `web.xml` to `webapps/metricsengconsole/WEB-INF/web.xml`.

5. Start the servlet container.

Backing Up the Metrics Engine DBMS

This section provides information about why you may need to backup the DBMS, and then how to plan and execute your backup strategy. Understanding what happens during the backup process is important because executing a DBMS backup requires taking the Metrics Engine offline for the duration of the backup.

About Backing Up DBMS Data

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 used. 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 allow you to:

- Diagnose performance problems that occurred in the past.
- Provide historical data for capacity planning and historical reporting.

- Provide the data needed for a revenue stream, such as when Metrics Engine data is used for billing and chargeback.

Evaluating the parts of the data that are important to you determines your backup strategy. For example, in the case of billing and chargeback, the data needed for these sorts of tasks is typically small compared to the total population of the DBMS, and you can use the API to extract the data on a regular basis and archive it in a set of CSV files. This may be all the data you need, and the planning and resources required to backup the DBMS will be minimal. However, in other circumstances, you may not be able to determine what data will be important to you in the future, in which case backing up all DBMS data is the safest approach.

About Historical Data Storage

The Metrics Engine DBMS stores all historical sample data starting from when the Metrics Engine first started collecting sample data. It can store time-aggregated data for up to twenty years, and 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 ample 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.

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.

By default, the Metrics Engine can be offline for about eight hours before any data is lost. If you target a backup that lasts less than two hours, you do not compromise the data.

Determining What Data to Backup

Sample data comprises about 95% of the DBMS by volume. It is broken evenly into four groups:

- One second
- One minute
- One hour
- One day

Each of these groups changes over time, with new samples flowing in and being aggregated and with old samples being deleted. The data in the one second group data changes much faster than the other groups. Using the default settings, 100% of the data in this group is replaced every 8 hours. So, even if you have a backup of this group, it will be completely out of date eight hours after you completed the backup. With such a short period of usefulness, we have omitted this group from the backup, saving both time and space.

The group that contains one minute data changes every seven days by default, though it can be as long as every five weeks. The data from this group can be useful if you back it up every week.

The one hour data changes every year by default, and the one day data changes every five years by default. Both of these groups are good candidates for backup.

From a planning perspective, we need to backup three of the four sample groups, if we backup at a frequency between daily and weekly. If we backup less often, then we may want to exclude the one minute data as well, and back up only the one hour and one day data groups.

Implications of Restoring Data

If you have a catastrophic storage failure, when you restore the data that was captured and aggregated between the most recent backup and now will be lost. For example, imagine we take weekly backups on Sunday night. If we have to restore from backup on the following Friday in a worst-case scenario, we will have a six day gap in our data. This gap represents six samples from the one day group, 144 samples from the one hour group, and 8640 samples for the one minute group. The collection points are still caching samples, so after the backup is restored, the Metrics Engine will immediately reclaim the most recent eight hours of data.

However, configuring the collection points to retain the data for an entire backup period, while providing 100% recovery of the data, comes at a high price. The Metrics Engine has a modest upper limit on how fast it can get data into the DBMS, approximately 500k samples per minute. If you presented a seven day backlog of data to the Metrics Engine, it will take several days for it to process the backlog, and the Metrics Engine would be unable to answer queries for current data until the backlog finished processing.

We recommend having a gap in the data rather than losing the ability to use the data at all for an extended period of time. Avoid any approach that results in more than two hours of catchup time.

Planning for Periodic Backups

When you plan for a periodic backup, you must choose a time window during which the Metrics engine can be offline and ensure that you have enough disk space to hold the new backup image. The exact size of a DBMS table and its corresponding backup is difficult to predict be because it depends on factors that change at each installation. These factors include 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. These values reflect backing up three of the four sample groups: one minute, one hour, and one day data.

Table 3: Data from Sample Deployments

Data	For 25 Monitored Servers	For 50 Monitored Servers
Number of tracked applications	20	20
1 second data resolution	8 hours	8 hours

Data	For 25 Monitored Servers	For 50 Monitored Servers
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 you choose to not make a backup and you lose your DBMS completely, you can always re-initialize the DBMS, restart the Metrics Engine, and start collecting data again. You will lose all collected metric data and all collected event data, but retain the configuration required to start collecting data again. If you have a fixed set of metrics that are historically important, simply taking snapshots of these metrics periodically (using the Metric engine RESTful API) and saving them as .csv files protects you if the DBMS is lost.

Before You Begin Your Backup

Before you attempt a backup or restore, you must shut down the Metrics Engine. If you backup or restore with the Metrics Engine running, you will end up with a corrupted backup or corrupted database.

You need the following information to complete a backup:

- **Database name**. This name was specified during Metrics Engine installation and is available through the `dsconfig` tool. The default value is `metricsengine`.
- **Schema name**. This name was specified during Metrics Engine installation and is available through the `dsconfig` tool. The default value is `metricsengine`.
- **PostgreSQL data base administrator login**. This information was specified at PostgreSQL installation time.
- **PostgreSQL data base administrator password**. This information was specified at PostgreSQL installation time.

The PostgreSQL login and password must have DBA superuser level privileges. Because you are running PostgreSQL tools from the command line, you need to know how to authenticate to the DBMS using these tools. Typically, you can use the following options to authenticate and prompt for the password:

```
-U dbms login -W
```

These options are omitted from the following examples to improve clarity.

How to Backup the Database

To backup the entrie DBMS, excluding the one second data, use the following command:

```
$ pg_dump -v -c -n schema --no-unlogged-table-data database > backup-file-name
```

If you installed the Metrics Engine with the default settings, the command appears as follows:

```
$ pg_dump -v -c -n metricsengine --no-unlogged-table-data metricsengine > backup-file-name
```

The backup command takes the following arguments and options:

- *schema* The schema name used. By default, the value is `unboundid`.
- *database* The database name used. By default, the value is unboundid.
- *backup-file-name* The name of the file where the backup is stored.
- `-n` This option specifies that only the schema we are using be backed up.
- `-v` This option specifies to use the verbose mode.

Note: You cannot backup and restore parts of the schema beyond the sample tables. The Metrics Engine application code creates implicit relationships between different tables that are not enforced by the DBMS. If you backup the DBMS with the Metrics Engine running, restoring that backup may corrupt the sample tables, aligning sample data with the wrong metric meta data. The backup must be on a DBMS that has been stopped, and it must capture all of the tables you intend to restore. Any tables not included in the backup must be truncated when you restore from the backup, otherwise these implicit relationships will be broken.

How to Restore the Database

Use the following command to restore the backup created above:

```
$ psql -d database < backup-file-name
```

Executing this command takes between 10 minutes and two hours, depending on the size of the backup. The command drops and recreates each backed up table and index, and then reloads all data for the table stored in the backup.

Excluding Data from Specific Aggregation Levels

The default backup command skips only the unlogged tables, which is the one second data we concluded should not be backed up because it would be stale before the backup could be used. However, you can skip other aggregation levels. To skip an aggregation level, the `-T` option of the `pg_dump` tool should be used. For example,the following command skips the one second and one minute data:

```
$ pg_dump -v -c -n  schema --no-unlogged-table-data \
  -T schema.histo_l1 -T schema.histo_d* \
  -T schema.scalar_l1 -T schema.scalar_d* \
  schema > backup-file-name
```

The following tables describe the names of the specific aggregation levels

Table 4: Aggregation Level Table Names

Aggregation Level	Table Names
1 second	histo_10, histo_h*, scalar_l0, scalar_h*
1 minute	histo_l1, histo_d*, scalar_l1, scalar_d*
1 hour	histo_l2, histo_m*, scalar_l2, scalar_m*
1 day	histo_l3, histo_y*, scalar_l3, scalar_y*

If you explicitly use the `-T` option, then you must prepare the DBMS before you can restore, otherwise the restore will report errors. Every table that is excluded by the `-T` option during backup must be dropped before the restore is attempted.

Performing a Full Backup

You may want a full DBMS backup and restore, rather than exclude the most recent data tables. A full DBMS backup can be used to load the data into a different DBMS server or into another database in the current server. Or, you may want to send the DBMS image to your support provider for analysis of a performance problem.

To create a full backup, use the following command:

```
$ pg_dump -v -c -n  schema database > backup-file-name
```

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

```
$ psql -d  new-database-name < backup-file-name
```

How to Export and Import the Database

You may want to export the full database, including the tables that should normally be excluded. For example, you may want to export the database to import the entire schema into another database, perhaps to do further analysis on it without the Metrics Engine continuing to import, aggregate, and trim the data. Below are the commands to export the full database, and then to import it into a new database for further processing or analysis:

```
$ pg_dump -n schema -Fc database > full-backup-file-name
$ createdb -T template0 new-database-name
$ pg_restore -d new-database-name full-backup-file-name
```

You may also want to export the full database if you need to send a Metrics Engine database image in for support to analyze.

Uninstalling the Metrics Engine

The Metrics Engine provides an uninstall command-line utility for quick and easy removal of the code base. You can uninstall the Metrics Engine using one of the following modes:

- **Interactive command-line mode**. This mode is a text-based interface. The utility prompts you for input if more data is required.
- **Non-interactive command-line mode**. This mode suppresses progress information from being provide in standard output during processing, except for fatal errors. This mode is convenient for scripting and is invoked with the `--no-prompt` option.

To Uninstall the Metrics Engine in Interactive Mode

Interactive mode uses a text-based, command-line interface to help you remove your Metrics Engine instance. If `uninstall` cannot remove all of the Metrics Engine files, the server generates a message with a list of the files and directories that must be manually deleted. The `uninstall` command must be run as either the root user or the same user (or role) that installed the Metrics Engine.

1. Go to the installation directory.

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

2. Use the uninstall command.

```
$ ./uninstall
```

3. Select the components to be removed. If you want to remove all components, press **Enter** to accept the default.

4. If the Metrics Engine is running, press **Enter** to shutdown the server before continuing the uninstall process.

5. Complete the uninstall, and view the logs for any remaining files. Manually remove any remaining files or directories, if required.

Uninstalling the Metrics Engine in Non-Interactive Mode

The uninstall utility provides a `--no-prompt` option that you can enter on the command line or use in a script. Use the `--forceOnError` option to continue the uninstall process even when an error is encountered. If an option is incorrectly entered or if a required option is omitted and the `--forceOnError` option is not used, the command will fail and abort.

To Uninstall the Metrics Engine in Non-Interactive Mode

1. Go to the installation directory.

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

2. Use the uninstall command.

```
$ ./uninstall
```

3. Use `uninstall` with the `--remove-all` option to remove all of the Metrics Engine's libraries. The `--quiet` option suppresses output information and is optional.

```
$ ./uninstall --remove-all --no-prompt --forceOnError
```

4. If any files or directories remain, manually remove them.

To Uninstall Selected Components in Non-Interactive Mode

1. Go to the installation directory.

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

2. Use `uninstall` with the `--backup-files` option to remove the Metrics Engine's backup files. Use the `uu--help` or `-H` option to view the other options available to remove specific components.

```
$ ./uninstall \
  --backup-files \
  --no-prompt \
  --quiet \
  --forceOnError
```

Uninstalling the Metrics Engine Management Console

You can easily remove the existing Metrics Engine Management Console by removing the `webapps/metricsengconsole` directory when no longer needed on your system.

To Uninstall the Metrics Engine Management Console

1. Close the Metrics Engine Management Console, and shut down the servlet container. (On Microsoft Windows, use `shutdown.bat`).

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

2. Remove the `webapps/metricsengconsole` directory.

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

3. Restart the servlet container instance if necessary. Alternatively, if no other applications are installed in the servlet instance, then the entire servlet installation can be removed by deleting the servlet container directory.

Cleaning Up the PostgreSQL DBMS After Uninstall

The Metrics Engine stores metric samples and bookkeeping information in the DBMS. After the Metrics Engine has been uninstalled this data may no longer be useful. If there is a chance that the Metrics Engine may be reinstalled and the existing metric sample data may be useful, then you do not need to cleanup the DBMS. However, if the Metrics Engine will not be reinstalled, or if you just want to get rid of all of the old data before you reinstall, use this procedure.

1. shutdown the Metrics Engine.

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

2. Login as the PostgreSQL DBA user, typically the `postgres` user.

   ```
   postgres$  echo "drop database if exists <database-name>" | psql -d template0
   ```

   The <database-name> is the name of the DBMS created when the Metrics Engine was first installed. By default, the value is `metrics-engine.` 'template0' is the name of the PostgreSQL built-in database. This name is required because a database cannot be dropped if any session is connected to it, so the psql command session must connect to a different database.

3. The command above deletes all of the Metrics Engine database specific files from the filesystem, so the disk space will be reclaimed shortly after the command completes. If you plan to re-install the metrics engine at a later time, you can leave the DBMS alone and allow the setup program (during re-install) to clean up all of the old DBMS data.

The command above deletes all of the Metrics Engine database specific files from the filesystem, so the disk space will be reclaimed shortly after the command completes.

If you plan to re-install the metrics engine at a later time, you can leave the DBMS alone and allow the setup program to clean up all of the old DBMS data during reinstall.

Upgrading and Reverting the Metrics Engine

UnboundID issues software release builds periodically that contain new features, enhancements, and fixes for improved server performance. Administrators can use the Metrics Engine update utility to upgrade the current server code version.

The section describes some upgrade implications and provides procedures to help you upgrade or revert your Metrics Engine deployment.

Metrics Engine DBMS Schema Considerations

The Metrics Engine PostgreSQL DBMS schema needs to be updated when you upgrade from version 3.5.1 to 3.6. The tools needed to update the schema are provided in the 3.6 update and takes very little time. However, you must update the schema manually. Also, no tool is provided to revert the schema from the 3.6 version to the 3.5.1 version. However, the Metrics Engine 3.5.1 can use the Metrics Engine 3.6 DBMS schema, so reversion is unnecessary even if you choose to revert the upgrade.

Overview of the Upgrade Process

The upgrade process involves downloading and unzipping a new version of the Metrics Engine on the machine hosting the Metrics Engine server. Next, you run the `update` utility provided in the 3.6 version of the server. This tool prompts for the installation directory of the current Metrics Engine. The `updates` utility stops the current Metrics Engine, if it is running, updates the binaries, checks the resulting configuration, and then attempts to restart the upgraded Metrics Engine. The restart fails if the Metrics Engine PostgreSQL DBMS schema are at version 1.0, because the 3.6 version of the Metrics Engine expects version 2.0.

To update the PostgreSQL DBMS schema you need to execute the command noted by the `update` utility. After the schema update is complete, you can restart the Metrics Engine.

To Upgrade the Metrics Engine

This section describes how to update a Metrics Engine installed in the `UnboundID-Metrics-Engine` directory.

1. Create a new empty directory that will contain the unzipped new Metrics Engine binaries.

   ```
   $ mkdir ~/metrics-3.6
   ```

2. Change to the new empty directory.

   ```
   $ cd ~/metrics-3.6
   ```

3. Unzip the new Metrics Engine binaries into the new empty directory.

   ```
   $ unzip ~/UnboundID-Metrics-Engine-3.6.0.0-no-je.zip
   ```

4. Change to the directory containing the new Metrics Engine binaries.

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

5. Initiate the upgrade of the current Metrics Engine. This upgrade will fail when trying to restart the newly upgraded Metrics Engine because the PostgreSQL schema are the wrong version.

   ```
   $ ./update --serverRoot UnboundID-Metrics-Engine
   ```

6. **Change to the directory containing the new upgraded Metrics Engine installation.**

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

7. Upgrade the PostgreSQL schema from version 1.0 to version 2.0.

   ```
   $ bin/metrics-engine-schema --from 1.0 --target 2.0
   ```

8. Restart the now fully upgraded Metrics Engine.

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

9. Confirm that the upgrade was successful by checking the version of the fully upgraded Metrics Engine.

   ```
   $ ./setup --version
   ```

To Revert the Metrics Engine

The Metrics Engine binaries can be reverted without the need revert the PostgreSQL DBMS schema because the new schema is compatible with old and new versions of the Metrics Engine code.

1. Change directory to the upgraded Metrics Engine installation directory.

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

2. Revert the Metrics Engine code from 3.6.0 to 3.5.1.x.

   ```
   $ ./revert-update
   ```

3. Check the version of the now reverted Metrics Engine.

   ```
   $ ./setup --version
   ```

Chapter 3 Data Collection and Metrics

This chapter describes how data is collected, how to tune the data collection, and how to access the data.

Topics:

- *Overview of Metrics Concepts*
- *Overview of Query Concepts*
- *About the Data Collection Process*
- *About the Collection of System Monitoring Data*
- *About Monitored Server Clock Skew*
- *Tuning Data Collection*
- *About Data Processing on the Metrics Engine*
- *Accessing Monitoring Data*

Overview of Metrics Concepts

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

- **Count metrics**. These metrics represent the number of times a specific event happens within the server. Examples of count metrics include number of LDAP operations performed, network packets received, or new connections established.
- **Discrete metrics**. These metrics correspond to measurements that have both a value and a weight. For exampe, the duration of an LDAP operation or the average duration of a checkpoint.
- **Continuous-valued metrics**. These metrics measure things that always hae 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.

Each metric collected by the Metrics Engine is of only one of these types, and the type is determined by what is measured and how it is measured. The statistics that can be applied when reading 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.

The metric type also plays a role in how samples are aggregated. Aggregation occurs when multiple metric samples taken over time are collapsed into a single sample.

About Analzying Aggregated Data

We think of metrics as having dimension because the dimension are a convenient way to organize the fact that a single metric may have several different values at the same time. Another way to think of the sample data is as independent data values, where each value is associated with exactly one set of dimensions. It can be more convenient to examine the data with certain dimensions aggregated, rather than pivoted or split. The aggregation helps us form a more simplified mental image of the data, which in turn helps us understand it more quickly.

Because the sample data stored in the DBMS is actually unaggregated, we need to understand the mathematics of aggregation. The following table shows data for a one-dimensional example:

Table 5: Example One-Dimensional Data

Dimension Value	T0	T1	T2	T3	T4	T5	T6	T7
a	5.0	5.0	5.1	5.2	5.1	5.2	5.1	NaN
b	5.0	5.0	NaN	5.2	5.1	5.2	5.1	NaN
c	5.0	5.0	5.1	5.2	5.1	5.2	5.1	NaN
d	6.0	6.0	6.1	6.2	5.1	6.2	6.1	NaN

If we aggregate dimension value (a:d) at time T0, we get different values depending on what statistic we request:

```
MINIMIM(aggregate((a:d)@T0) == 5.0
MAXIMUM(aggregate((a:d)@T0) == 6.0
AVERAGE(aggregate((a:d)@T0) == 21.0/4
COUNT(aggregate((a:d)@T0) == 21.0
```

If we aggregate dimension value (a:d) at time T2, we get different values depending on what statistic we requested:

```
MINIMIM(aggregate((a:d)@T2) == 5.1
MAXIMUM(aggregate((a:d)@T2) == 6.1
AVERAGE(aggregate((a:d)@T2) == 16.3/3
COUNT(aggregate((a:d)@T2) == 16.3
```

Note that the NaN values do not count as a zero, so missing data does not adversely affect aggregation. However, the COUNT statistic looks very different at T2 than it did at T0, even though the raw data only changed by 25%.

If we aggregate dimension value (a:d) at time T7, we get different values again. Here, we have no data to work with, so the aggregates are all NaN.

```
MINIMIM(aggregate((a:d)@T7) == NaN
MAXIMUM(aggregate((a:d)@T7) == NaN
AVERAGE(aggregate((a:d)@T7) == NaN
COUNT(aggregate((a:d)@T7) == NaN
```

The second axis of aggregation is across time. Normally, you would not think of time as an aggregation, but the DBMS only contains four different time resolutions for the samples. So, unless you want one of those four resolutions, you have to aggregate time. Using the same data as above, we aggregate time into two samples, such that T0-T3 are sample 1 and T4-T7 are sample 2. We will not aggregate by dimension. The results are given in the following table.

Table 6: Example Time Aggregation Data

Dimension Value	S1	S2
a	20.3/4	15.4/3
b	15.2/3	15.4/2
c	20.3/4	15.4/3
d	24.3/4	17.4/3

Note how again the NaN values do not affect the computations, and that the time aggregation is always the average of all actual values.

Next, we aggregate over time where the samples do not fit evenly into the new time intervals. In this example, we aggregate our eight time intervals into three as follows:

```
T0,T1,T2 -> S1
T3,T4,T5 -> S2
T6,T7 -> S3
```

The resulting aggregation follows.

Table 7: Aggregating Over Time with Uneven Samples

Dimension Value	S1	S2	S3
a	15.1/3	15.4/3	5.1

Dimension Value	S1	S2	S3
b	10/2	15.5/3	5.1
c	15.1/3	15.5/d	5.1
d	18.1/3	17.5/3	6.1

However, the results can be skewed if you have an outlier value in the wrong place. If the dimension value d at time T6 was 1.0, then the dim value d at sample time S3 would be 1.0 because it would not have any other data to average with.

Averaging that occurs during aggregation can result in misleading data when taken out of context. Generally, it is not wise to aggregate the raw data, isolate a single point in the aggregate results, and draw any conclusions. An outlier value in an aggregate should be examined along the different pivot dimensions before you can infer real meaning from the data, as missing data can influence the aggregates.

About the Types of Metrics

A count metric indicates the number of times a specific event happens within the server. The exact length of the measurement interval is not important. 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 the measurement interval. Using the COUNT statistic when querying for a count metric will return the sum of the counts. If the query has a time-based pivot, then it returns the sum of the counts split into time quanta. Count metrics can often be converted into a rate. Using the examples above, the per-minute rates would be packets per minute and Megabytes per minute, which ensures that the time quant is 1 minute in duration.

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 interval, a valid CPU percent busy measurement can be taken. A continuous metric differs from a count metric in that you cannot sum continuous metric samples across time in a meaningful way. For example, if was have a 1 second measurement of CPU percent busy and the CPU is 25% busy for 10 seconds, summing these samples would show the CPU is 250% busy, which is not meaningful. Instead, continuous metric samples use average, minimum, and maximum statistics. If you want to know how busy the CPU has been since midnight, you average, rather than sum, the samples since midnight.

A discrete metric is a measurement that has both a value and a weight. A value with a higher weight means more samples within the sample period for that value. A discrete metric is analogous to a weighted average and requires that multiple measurements be taken within a single sample interval. For example, response time is a discrete metric, where the actual response time of each LDAP operation is averaged and the number of LDAP operations is provided as the weight. Discrete metrics are different from continuous metrics because each measurement is weighted. If no LDAP operations occurs 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 rather than using a min()/max() function, so the aggregated values do not automatically push to the extremes but rather reflect the median of the minimum/maximum values.

Some discrete metrics may also convey histogram data. Histogram data represents an additional set of measurements that take individual measurements and place them into buckets, where each bucket is defined by a value range. The Metrics Engine supports histograms with up to 15 buckets. 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?". When multiple measurements are reduced to a single number (average), then the sample value distribution is lost. This loss occurs at the fine-grained measurement level, and during time-based aggregation. However, a histogram representation does not lose the distribution information. Histograms can be added over time so that we can always answer the question "How many samples are greater than X?" exactly (if the value of X is a histogram bucket boundary).

About 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 that the sample was created on. Beyond that, 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 on the UnboundID Synchronization Server. The `pipe-result` is one of the following set of 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 sync 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
```

First, we compare how busy pipe-one is compared to pipe-two by pivoting on `pipe-name`. We do not pivot on `pipe-result`, allowing all twelve results to aggregate to a single value. This results in the following:

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

However, failures would get double-counted, once in the failed result and again in the specific failure mode. So, we can filter specific dimension values into the result by considering only samples with `success` or `failed` pipe-result values in the aggregation. This results in the following:

```
pipe-one  132
  pipe-two  226
```

These values provide the real sync pipe operation count. Next, we can pivot by `pipe-result`, not `pipe-name`, 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 being encountered by the sync pipes.

The dimensions give us 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. The following diagram illustrates a response time histogram pivot.

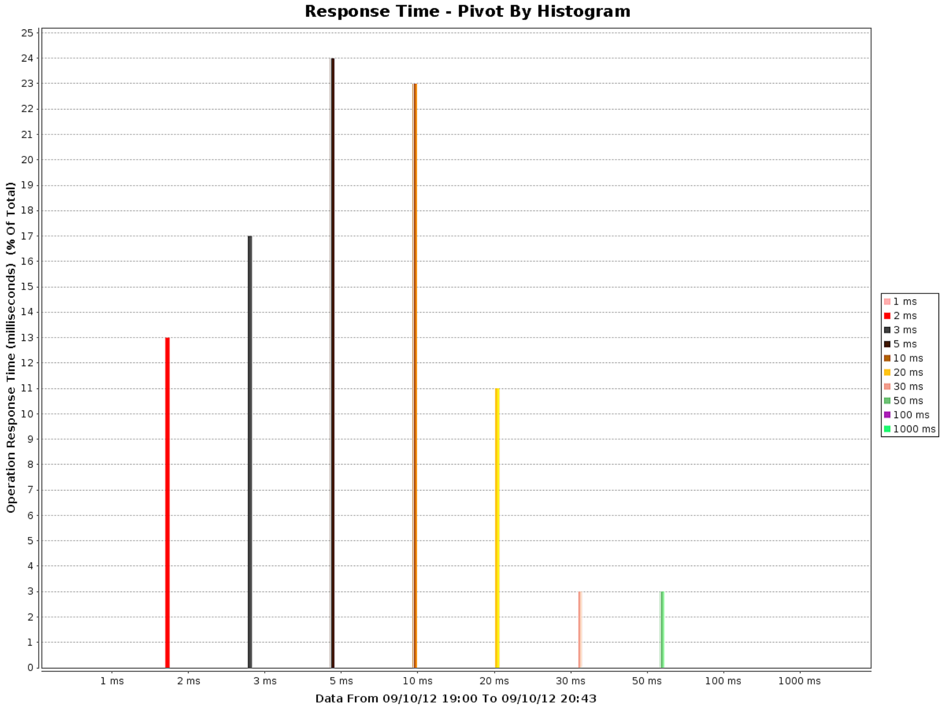


Figure 4: Pivot by Histogram

A time pivot allows the results to show time as a dimension, which breaks the samples that correspond to the requested time range into time quanta. Commonly, you specify N samples across range R, filling in a time-series chart that shows the metric value as a function of time. The following diagram illustrate a response time pivot by time.

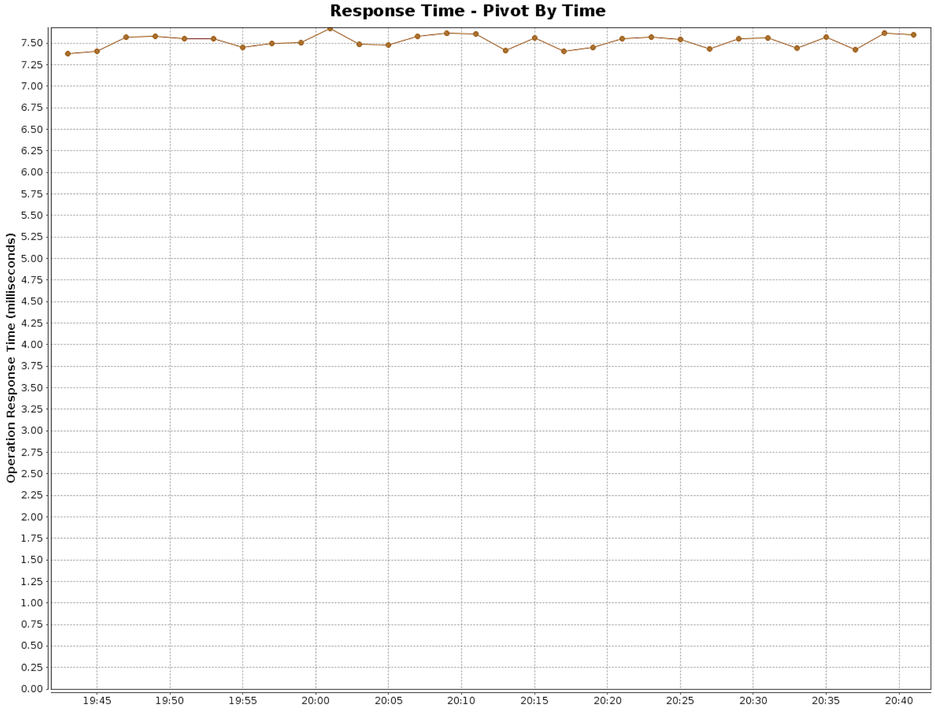


Figure 5: Pivot by Time

An instance pivot allows the results to be pivoted, or split, by server instance, allowing us to compare servers against each other. We can see how busy each server's CPU is by looking at the `host-system-cpu-used` with an instance pivot. The following diagram illustrates response time pivoted by time and instance.

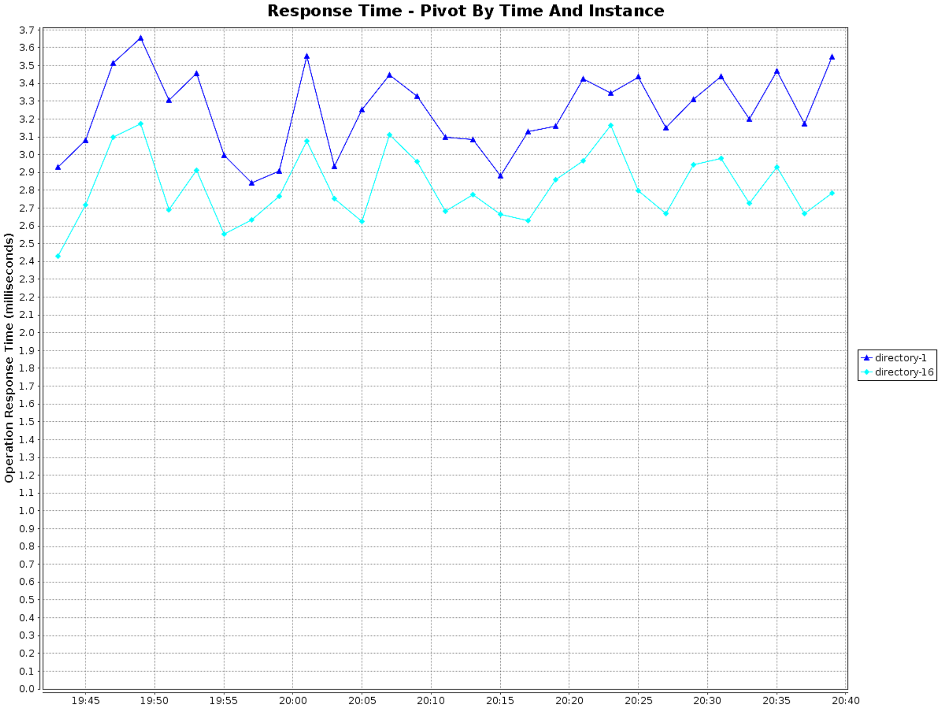


Figure 6: Pivot by Time and Instance

A histogram pivot requires a metric that supports the histogram statistic, and discriminates the results by histogram bucket boundaries. For example, LDAP response time with a histogram pivot shows how many LDAP operations fell into each of the histogram buckets. The following diagram illustrate throughput pivoted by time, instance, and application.

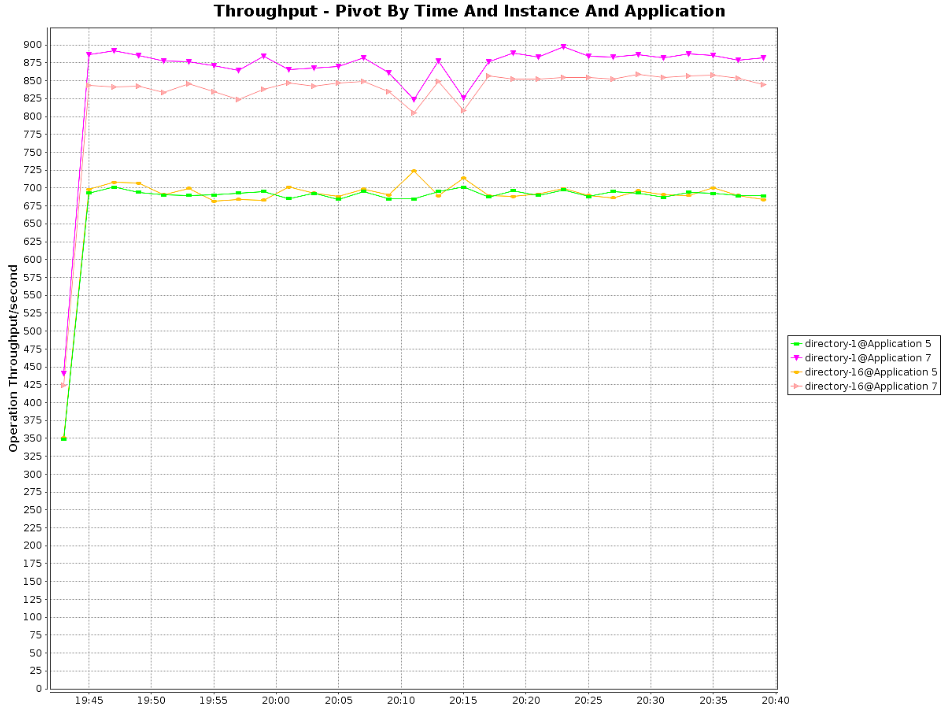


Figure 7: Throughput Pivot

Overview of Query Concepts

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

The remainder of this section describes each of these components in more detail.

Selecting 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, such as the average, and of a single metric, such as response time.

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, if you have selected 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 directory servers
- 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, you can query the count statistic 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.

Aggregating the Query Result

You may want a metric query to 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, as described earlier. A pivot directs the query processor to not aggregate one component of the query data. A pivot can be one of the following:

- Time
- Server instance
- A specific dimension
- Histogram buckets

Zero or more pivots can be specified in the query with the following results:

- If no pivot is specified, then 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.

- Using 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.

- Using 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 to make it more consumable by the client. 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.

Formatting the Query Result

The final step of query processing is to convert the results into the format requested by the client. Results can be returned in one of the following formats:

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

About the Data Collection Process

The Metrics Engine polls all the monitored servers over LDAP to fetch alert, status, and performance 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` entry 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.

The following sections describe the types of data collected in more detail.

About Performance Data

Performance data represents a majority of the data collected by the Metrics Engine. Depending on how you configure the servers in your topology, each may produce hundreds of kilobytes of performance data per minute, and the Metrics Engine stores performance data for 20 years. You must configure the volume of performance data collected by each monitored server so that the Metrics Engine can keep up with the flow. Ideally, the amount of data captured has little to no impact on the performance of the monitored system.

The performance data model is a dimensional data model, meaning that a measurement 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:

Name	Data Type	Description
Timestamp	Date	Time of measurement, using clock on the monitored server
Metric	String	Name of metric
Dimension	String[3]	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 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

About the Collection of System Monitoring Data

All UnboundID servers have the capability to monitor the health of the server and host system they run on for diagnostic review and troubleshooting. Initially, the servers do not collect any performance data until they are prepared for monitoring by an UnboundID Metrics Engine using the `monitored-servers add-servers` tool or an administrator enables system health data collection for real-time inspection and querying. At a high-level, all of the important server and machine metrics which can be monitored are available in the `cn=monitor` backend.

Note: Windows is not a supported monitoring platform. Host system monitor data is unavailable on Windows.

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 what kind of data aggregation occurs for LDAP application statistics. The Stats Collector plugin ensures that an UnboundID Metrics Engine is able to gather all of the detailed data required for a comprehensive diagnostic review.

The Stats Collector plugin relies exclusively on entries in the `cn=monitor` backend to sample data using LDAP queries. In order for real-time host system monitoring data to be present, the Host System Monitor Provider populates the monitor backend with specific real-time attributes about CPU and memory utilization. You can also configure the Host System Monitor Provider to collect real-time utilization data for specific disk subsystems and network interfaces on the host. Configuration of the Host System monitor provider enables the low-level generation of performance data through system-specific System Utilization monitor modules.

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

Utilization metrics are gathered via externally invoked OS commands, such has `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`, by the loopback interface (`lo`), and by disks (`sd1, sdab, sda2, scsi0`).

About the External Collector Daemon

For some UNIX-based operating systems, the System Utilization monitor contains an embedded collector daemon that runs on systems affected by a Java process fork memory issue, RFE 5049299. The embedded collector daemon is started automatically before the UnboundID server via the startup script named `_start-collector-helper.sh`. The `start-collector-helper` inspects the Host System Monitor provider configuration to conditionally determine whether the external daemon process is required.

Note: On Linux, no OS external commands are forked. Instead, the `/proc` filesystem is accessed directly using file input/output to read the latest CPU, memory, disk, and network I/O data.

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 logs directory for each command type so the Host System Monitor Provider can find the output of the most recently collected data.

Commands that are repeated on an interval are executed on a thread at a 2x interval sampling rate. The filename of the output contains the sample timestamp, such as iostats-[sample-timestamp]. Repeating commands use a subdirectory for each command type to keep results isolated from other command types and to help organize file cleanup. 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 will be missing. To handle clock-edge timing, the monitor sampler will also look for data in a filename of the previous second. Samples cannot be reused because timestamp files are deleted once their data have been collected.

The collector daemon runs with no inter-process communication. However the collector daemon monitors a process PID file provided by the startup script and it waits for the process PID file to be created and/or waits for an `empty-file-to-active` process PID transition. Once an active server process PID is acquired, the collector daemon monitors the contents of the process PID file, as well as the file existence, to determine whether the UnboundID server is active. If the process PID file is deleted, the process PID contained within the PID file changes, or the process PID becomes inactive, the UnboundID server has stopped and the monitor daemon exits within one second. The daemon also monitors the Host System Monitor provider configuration and the daemon exits within one second if the provider is disabled in the server.

Because of the cooperative nature of the external collector daemon files and the output file readers in the System Utilization monitor, the collector daemon can be safely killed by a system administrator. If the collector daemon is terminated, host system machine diagnostic metrics are not available for monitoring until the UnboundID server is restarted.

About Monitored Server Clock Skew

Correlating metric samples from multiple servers as a function of time requires that the timestamp associated with each sample (which is provided by the monitored server) is in sync. The monitored servers need to have their system clocks 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. If you have a five second spike in a metric on Server A and a similar six second spike on Server B, if the system clocks on Server A and Server B are not synchronized, you will not know which came first or if they were truly concurrent.

The Metrics Engine does not actively do anything to help synchronize the system clocks, but it does track that information and make it visible in the `cn=Monitored Server <`*`server-name`*`>,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 you have 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 will computed using the Metrics Engine system clock, if this clock is very different from the monitored servers' system clocks, then the start/end time of a metric query will not match the expected boundaries.

Tuning 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

You may not require all of the metrics produced by the Metrics Engine. If not, tune the sets of metrics collected by using the `dsconfig` command-line tool to update the Stats Collector Plugin's `entry-cache` property. For example, to omit all metrics related to the entry cache set the `entry-cache-info` group as follows:

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

☛ **Note:** The `dsconfig` commands in this section are to be run on each monitored server, not necessarily on the Metrics Engine server.

The server collects information for eight different info groups. In this example, we set the `entry-cache-info` group to none, meaning that none of the metrics from that info group are produced. Limit data collection to the devices of actual interest.

Reducing the Frequency of Data Collection

The monitored servers produce metric samples as quickly as every second, which is useful for short-duration changes. However, these samples are less useful hours later, after the per-second data is aggregated to per-minute data. The following example illustrates how to 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

You can also reduce the number of sample blocks processed by the Metrics Engine 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. For example, you can 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 Directory Performance

The Directory, Proxy, and Synchronization Servers all expose performance data through the `cn=monitor` DN. Performance penalties arise only when this 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 that contains human-readable column titles and several value columns per line. The output is suitable for human consumption, typically through a spreadsheet application.

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 at the `cn=metrics` DN. The Stats Collector CSV files are suitable for use by the Metrics Engine, and contain one metric value per line.

If you do not want to monitor performance, you can disable both the Periodic Stats Logger (disabled by default) and the Stats Collector (disabled by default) plugins using dsconfig. Each of these plugins adds an approximate 3% CPU utilization penalty, plus a negligible amount of disk I/O and JVM heap usage.

For example, to enable the Stats Collector plugin, use `dsconfig` as follows:

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

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

About Data Processing on the Metrics Engine

When blocks of samples arrive at the Metrics Engine, they are queued on disk and loaded into the database on a FIFO basis. 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 allows different monitored servers to become unavailable to the Metrics Engine, for example, by going off line, 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.

Note: Samples collected from the Metrics Engine itself are processed ahead of all other servers.

Data Importing

The Metrics Engine polls all of the monitored servers at a regular interval, looking for blocks of samples that have not already been collected. When new samples are available, the Metrics Engine fetches them via LDAP, queues them to disk, and adds an import record to the FIFO import queue. The Metrics Engine has one dedicated thread draining the import queue, taking each block of samples 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.

For example, if a monitored server becomes unavailable for an extended period of time, perhaps for several hours, 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 even 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.

When the Metrics Engine first starts, it will queue (for import) all sample blocks still on disk. All sample blocks on disk at server startup are first checked for maximum sample age. Blocks that are older than two hours are discarded.

Data Aggregation

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.

The monitored servers generate three types of metrics, and each type is aggregated differently:

- **Counts**. Samples are aggregated by a sum of the values.
- **Continuous**. Samples are aggregated by an average of the values.
- **Discrete**. Samples are aggregated by a weighted average of the values.

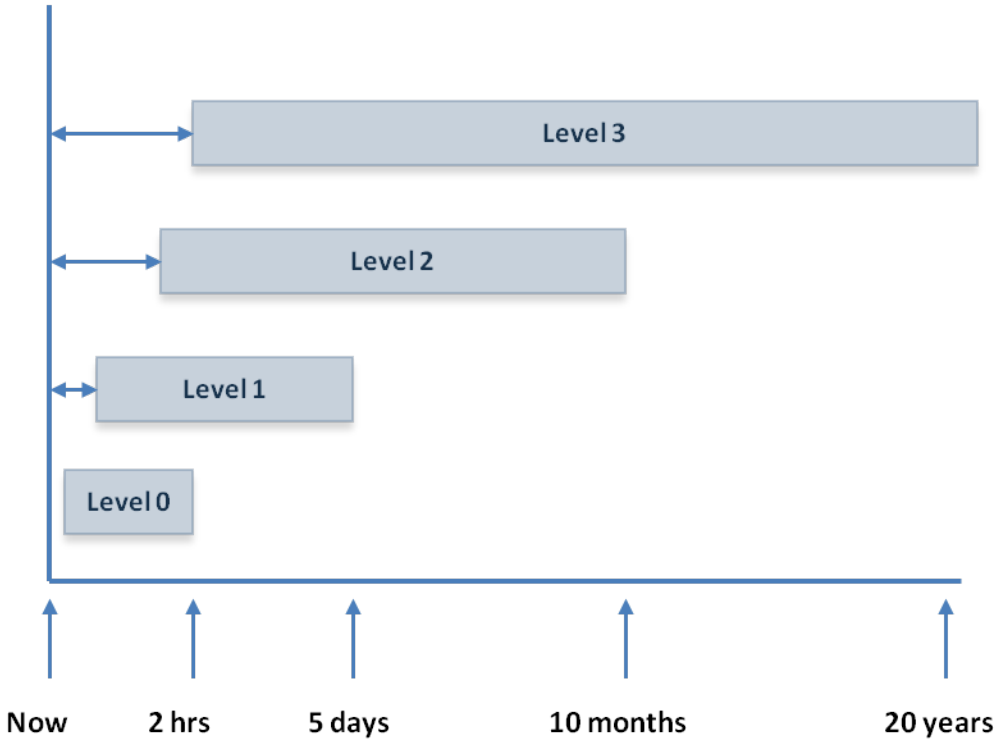
To keep the data tables for each aggregation level at a constrained size, each aggregation level has a maximum age for the samples. When the samples are older than this age, they are deleted from the level. While aggregation occurs soon after the samples arrive in the level, pruning occurs only after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

Level	Resolution	Default Maximum Age	Max 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

The raw data from the Metrics Engine is initially put in level 0. After a period of time, the newest data in level 0 is aggregated and put into level 1. This aggregation process carries on up to level 3. To keep the DBMS at a fixed upper bound, as the data ages out of each level it is deleted. Consider the following diagram. Note that the x-axis time scale is non-linear:



Level 0 data holds the most recent 2 hours of data. When data in level 0 is older than 2 hours, it is deleted. Because the data in the lower levels have greater time resolution, transient data issues are more visible in lower levels than in upper levels. The aggregation process results in a type of averaging or data smoothing.

A second feature of this type of aggregation is that a gap exists between now and the newest available data for each level. This gap results from aggregation occurring on the boundary of the time resolution of the data. Level 3 data has one day resolution, so the newest data point it could have would need to aggregate an entire day’s worth of data. If the aggregation occurs at midnight, then at 12:01 AM, the level 3 data would include yesterday. However, it will not include today until after 12:01 AM tomorrow.

The pruning of data from each level can fall behind at times, so that a given level has more data than it should. However, pruning occurs often enough to ensure that the storage for the data does not grow without bound.

Accessing Monitoring Data

The Metrics Engine stores the data it collects in a DBMS, accessed via JDBC. The default configuration uses a DBMS server located on the same host as the Metrics Engine, with JDBC access limited to loopback connections. Within the DBMS server itself, the Metrics Engine uses a distinct database instance and distinct schema within the database. To access this schema, the Metrics Engine uses a DMBMS user with rights to the specified schema. A user can access the metric data over an HTTP port using the `query-metric` tool. This tool uses the Metrics Engine REST API, which is also available to custom applications

The data collected by the Metrics Engine does not contain any of the data in the LDAP entries of the monitored servers, so there is no risk of customer data being inadvertently exposed. The Metrics Engine does collect monitored server configuration data, most commonly exposed in the dimension values.

Chapter 4 Accessing the Metrics Engine Data

The data collected by the UnboundID® Metrics Engine is available through two main interfaces, the Metrics Engine RESTful API and the query-metric command-line tool. This chapter contains information about how to use these tools to access your monitored data, including API reference materials.

This chapter includes the following topics:

Topics:

- *About the query-metric tool*
- *Using the Query Metric Tool*
- *About the Metrics Engine API*
- *Metrics Engine API Reference*

About the query-metric tool

The query-metric tool can be used to view the data in a more interactive way than the API itself permits. The tool is a client application of the Metrics Engine API. It features subcommands that can help you understand how to form an API query, as well as tell you what values are permissible.

The query-metric tool features a non-interactive mode as well as an interactive mode that prompts you for information like a wizard. In addition to the subcommands for listing metrics, server instances, and dimension values, you can form queries using the following subcommands:

- **`explore`** subcommand. This command 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 helps you understand the breadth of available metrics and look for patterns or anomalies across multiple metrics. In interactive mode, the tool prompts you for the servers and the metrics using a menu.
- **`query`** subcommand. This command help you refine a query for specific data of interest. In interactive mode, the tool prompts you for the server, metrics, dimensions, statistics, and pivot values using menus. The tool can be used to request a server generated chart image file or data formatted in XML, JSON, or CSV.

Using the Query Metric Tool

The `query-metric` tool gives you access to all the metrics being gathered by the server. This tool allows you to explore the full breadth of the collected data, examining any metric and dimension.

This tool runs in both interactive and non-interactive modes. Interactive mode presents options in a wizard-like way, allowing you to choose values from menus of options. To start the tool in interactive mode, simply invoke the tool with no parameters:

```
$ query-metric
```

You will be shown a menu allowing you to choose the subcommand you want to invoke. Once the subcommand is finished, you will be given the opportunity to choose another subcommand or quit. You may invoke a subcommand in interactive mode by specifying it on the command line. For example, the following command starts the `explore` 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 can be used to obtain a JSON formatted data table that shows average throughput for all Proxy Server instances in the topology 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 subcommand and global tool options, invoke the tool's help as follows:

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

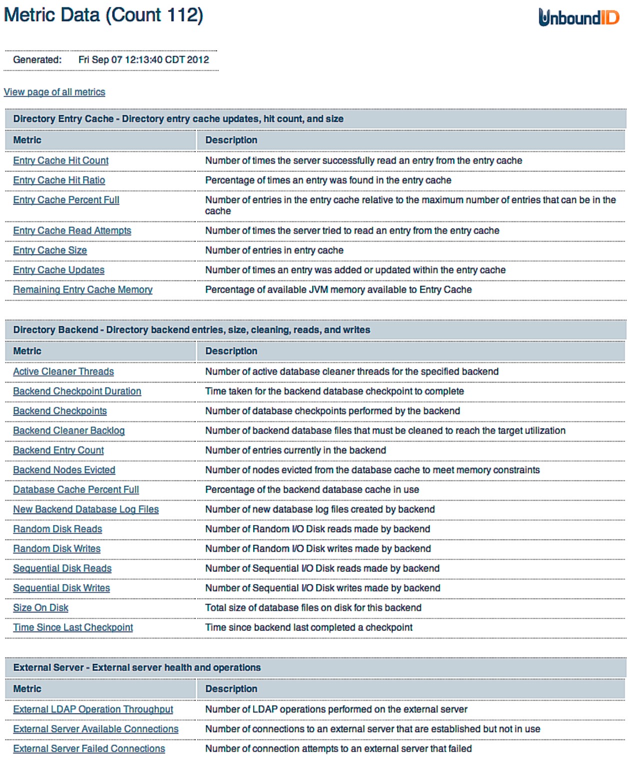
To get detailed information about a particular subcommand, invoke the subcommand's help as follows:

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

About the Query Metric Explore Command HTML Pages

The `query-metric` tool's `explore` subcommand generates queries that drive the Metrics API, such as adding a specific chart or tabular result to a custom dashboard. It also generate HTML page output. This section provides some examples of the HTML page output you can expect from the `explore` subcommand.

For example, the `explore` subcommand can generate an index page that shows tables of all the metrics that were collected:



Metric Data (Count 112)

Generated: Fri Sep 07 12:13:40 CDT 2012

View page of all metrics

Directory Entry Cache - Directory entry cache updates, hit count, and size	
Metric	**Description**
Entry Cache Hit Count	Number of times the server successfully read an entry from the entry cache
Entry Cache Hit Ratio	Percentage of times an entry was found in the entry cache
Entry Cache Percent Full	Number of entries in the entry cache relative to the maximum number of entries that can be in the cache
Entry Cache Read Attempts	Number of times the server tried to read an entry from the entry cache
Entry Cache Size	Number of entries in entry cache
Entry Cache Updates	Number of times an entry was added or updated within the entry cache
Remaining Entry Cache Memory	Percentage of available JVM memory available to Entry Cache

Directory Backend - Directory backend entries, size, cleaning, reads, and writes	
Metric	**Description**
Active Cleaner Threads	Number of active database cleaner threads for the specified backend
Backend Checkpoint Duration	Time taken for the backend database checkpoint to complete
Backend Checkpoints	Number of database checkpoints performed by the backend
Backend Cleaner Backlog	Number of backend database files that must be cleaned to reach the target utilization
Backend Entry Count	Number of entries currently in the backend
Backend Nodes Evicted	Number of nodes evicted from the database cache to meet memory constraints
Database Cache Percent Full	Percentage of the backend database cache in use
New Backend Database Log Files	Number of new database log files created by backend
Random Disk Reads	Number of Random I/O Disk reads made by backend
Random Disk Writes	Number of Random I/O Disk writes made by backend
Sequential Disk Reads	Number of Sequential I/O Disk reads made by backend
Sequential Disk Writes	Number of Sequential I/O Disk writes made by backend
Size On Disk	Total size of database files on disk for this backend
Time Since Last Checkpoint	Time since backend last completed a checkpoint

External Server - External server health and operations	
Metric	**Description**
External LDAP Operation Throughput	Number of LDAP operations performed on the external server
External Server Available Connections	Number of connections to an external server that are established but not in use
External Server Failed Connections	Number of connection attempts to an external server that failed

Figure 8: Index Page

Clicking the "View page of all metrics" link on the index page display a page that contains all the generated charts. This view allows you to easily scroll through all of the charts collected:

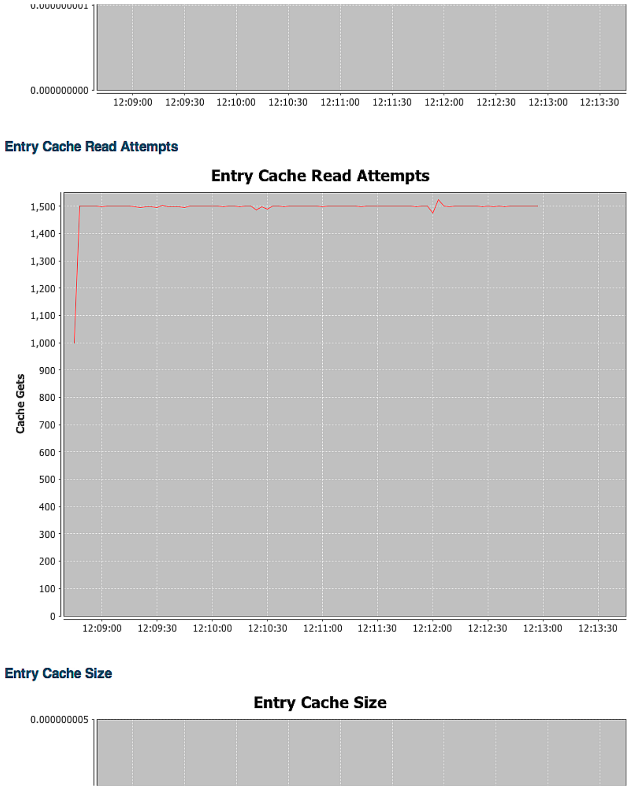
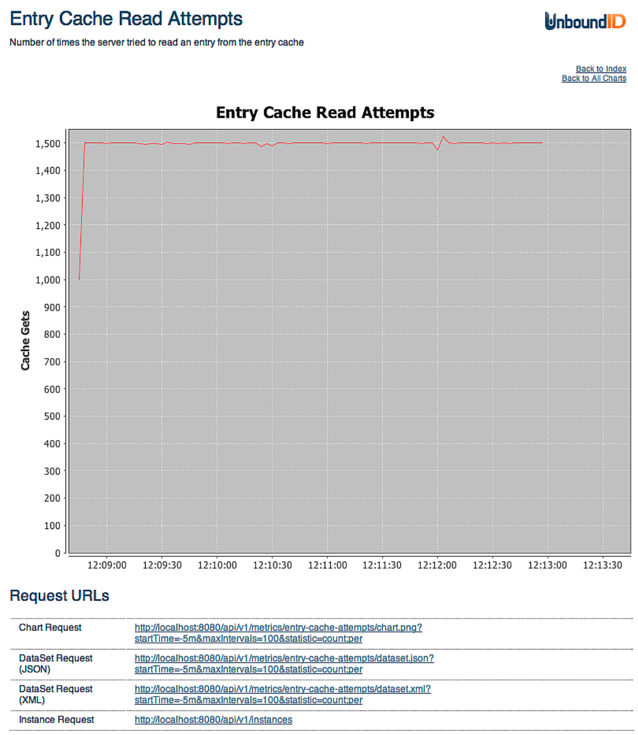


Figure 9: Metrics Overview Page

Clicking one of the links in the Metric column of the index page or clicking on a chart itself displays a new page. This page contains the chart, links for making requests from the Metrics Engine API for the same data in chart and data formats, as well as other information about the collected data:



Entry Cache Read Attempts

Number of times the server tried to read an entry from the entry cache

Back to Index
Back to All Charts

Request URLs

Chart Request	http://localhost:8080/api/v1/metrics/entry-cache-attempts/chart.png?startTime=-5m&maxIntervals=100&statistic=count;per
DataSet Request (JSON)	http://localhost:8080/api/v1/metrics/entry-cache-attempts/dataset.json?startTime=-5m&maxIntervals=100&statistic=count;per
DataSet Request (XML)	http://localhost:8080/api/v1/metrics/entry-cache-attempts/dataset.xml?startTime=-5m&maxIntervals=100&statistic=count;per
Instance Request	http://localhost:8080/api/v1/instances

Figure 10: Metric Details Page

About the Metrics Engine API

The Metrics Engine API can be used to build custom dashboards and other applications for exploring the data. It features a RESTful interface that can be accessed using standard, off-the-shelf tools and charting packages, such as the Google Chart Tools. The Metrics Engine API can also be easily accessed from a Web browser.

Metrics Engine API Reference

This section provides reference information for using the RESTful API of the UnboundID® Metrics Engine.

Connection Security and Authentication

As discussed in the section "Setting Up the Database," no sensitive user data is collected by the Metrics Engine and stored in the DBMS. However, if you wish to secure access to the Metrics Engine REST API, you may enable secure HTTPS connections and require authentication. A secure HTTPS Connection Handler may be created during setup, and authentication can be enabled using `dsconfig`.

Metrics Engine REST API authentication is disabled by default. When enabled, the REST API service requires HTTP basic authentication to be used with each request. Requests will be authenticated against entries in the `api-users` LDIF backend or entries in "cn=Root DNs,cn=config". Because Root DN users possess many privileges by default, we strongly recommend that you authenticate with users in the `api-users` backend instead, to prevent the unnecessary use of more privileged account credentials.

To Enable REST API Authentication

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
```

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
```

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

2. Load the entry using `ldapmodify`.

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

3. You can now authenticate using either the `cn` or the `uid` of the users added, in this case `api-user1`.

Tuning the RESTful API Service

By default, the Metrics Engine can open up to 20 simultaneous connections to its PostgreSQL RDBMS. The number of connections allowed is set by the `max-db-connections` property of the Monitoring Database configuration object. The HTTP Connection Handler, which runs the REST Servlet, has a default `num-request-handlers` value of 15. This value must be less than the maximum number of connections. If the RESTful API service is handling its maximum number of concurrent requests, this leaves at least five database connections available for other components, such as the import service.

> **Note:** If the Metrics Engine services requests through multiple HTTP Connection Handlers, such as to support both HTTP and HTTPS, then you must ensure that the total number of request handlers for both HTTP Connection Handlers does not exceed the maximum number of DB connections.

Listing Monitored Instances

Get a list of all monitored instances along with their current status. 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. Results may be filtered using the various `instance` query params.

URL	/api/v1/instances
Method	GET
Formats	JSON, XML
Query Parameters	• **`instanceType` (multi-valued)** - Types of server(s) to get data from. Possible values are: ➢ directory ➢ proxy ➢ sync ➢ metrics-engine • **`instanceLocation` (multi-valued)** - Location(s) of the servers to get data from. Multiple values are evaluated as logical ORs. • **`instanceHostname` (multi-valued)** - Hostname(s) of the servers to get data from. Multiple values are evaluated as logical ORs. • **`instanceVersion` (multi-valued)** - 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 OK

Response Body	`{`
	`  "found" : 2,`

```
  "offset" : 0,#
  "instances" : [ {
    "type" : "directory",
    "id" : "unboundid3600",
    "hostname": "unboundid3600.example.com",
    "displayName" : "unboundid3600",
    "version": "UnboundID Directory Server 3.6.0.0",
    "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"
      ]
    }
}] }
```

Retrieving a Monitored Instance

Get a specific monitored instance along with its status. 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/instances/{instance}{.format}
Method	GET
Formats	JSON, XML
Query Parameters	N/A

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

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

<table>
<tr><td><b>Response Code</b></td><td>200 0K</td></tr>
<tr><td><b>Response Body</b></td><td><pre>{
  "displayName": "metrics-engine",
  "hostname": "metrics-engine.example.com",
  "id" : "metrics-engine",
  "operatingSystem": "Solaris",
  "status" : {
    "state" : "ONLINE"
  },
  "type" : "metrics-engine",
  "version": "UnboundID Metrics Engine 3.6.0.0"
}</pre></td></tr>
</table>

Listing Available Metrics

Get a list of metric definitions along with their the units, dimensions, names, and so on. 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{.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` **(multi-valued)** - Limits the results to the metrics of the specified type. Possible values are: ➢ `discreteValued` ➢ `continuousValued` ➢ `count` • `group` **(multi-valued)** - Limits the results to the metrics with 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` **(multi-valued)** - Limits the result to metrics that uses the specified instance types as sources. Possible values are: ➢ `directory` ➢ `proxy` ➢ `sync` ➢ `metrics-engine` • `statistic` **(multi-valued)** - 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 in JSON format:

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

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

<table>
<tr><td><b>Response Code</b></td><td>200 OK</td></tr>
<tr><td><b>Response Body</b></td><td><pre>
{
    "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"
        },
        {
            "countUnit": {
                "abbreviatedName": "Evicted",
                "pluralName": "Evicted",
                "singularName": "Evicted"
            },
            "description": "Number of nodes evicted from the
                database cache to meet memory constraints",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-nodes-evicted",
            "instanceTypes": [
                "directory"
            ],
            "name": "Backend Nodes Evicted",
            "shortName": "Nodes Evicted",
            "statistics": [
                "count"
            ],
            "type": "count"
        },
        {
            "countUnit": {
                "abbreviatedName": "JE File",
</pre></td></tr>
</table>

```
                "pluralName": "JE Files/Logs",
                "singularName": "JE File/Log"
            },
            "description": "Number of new database log files
                created by backend",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-new-db-logs",
            "instanceTypes": [
                "directory"
            ],
            "name": "New Backend Database Log Files",
            "shortName": "New Log Files",
            "statistics": [
                "count"
            ],
            "type": "count"
        },
        {
            "countUnit": {
                "abbreviatedName": "RandRead",
                "pluralName": "Random Reads",
                "singularName": "Random Read"
            },
            "description": "Number of Random I/O Disk reads
                made by backend",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-random-reads",
            "instanceTypes": [
                "directory"
            ],
            "name": "Random Disk Reads",
            "shortName": "Random Reads",
            "statistics": [
                "count"
            ],
            "type": "count"
        },
        {
            "countUnit": {
                "abbreviatedName": "Rand Wr",
                "pluralName": "Random Writes",
                "singularName": "Random Write"
            },
            "description": "Number of Random I/O Disk writes
                 made by backend",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-random-writes",
            "instanceTypes": [
                "directory"
            ],
            "name": "Random Disk Writes",
            "shortName": "Random Writes",
            "statistics": [
                "count"
            ],
            "type": "count"
        },
        {
            "countUnit": {
                "abbreviatedName": "Seq Rd",
                "pluralName": "Sequential Reads",
                "singularName": "Sequential Read"
            },
            "description": "Number of Sequential I/O Disk reads
                 made by backend",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-sequential-reads",
            "instanceTypes": [
                "directory"
            ],
            "name": "Sequential Disk Reads",
            "shortName": "Sequential Reads",
            "statistics": [
                "count"
            ],
            "type": "count"
        },
        {
            "countUnit": {
                "abbreviatedName": "Seq Wr",
                "pluralName": "Sequential Writes",
                "singularName": "Sequential Write"
            },
            "description": "Number of Sequential I/O Disk writes
                 made by backend",
            "dimensions": [
                {
                    "id": "backend",
                    "values": [
                        "userroot"
                    ]
                }
            ],
            "group": "Directory Backend",
            "id": "backend-sequential-writes",
            "instanceTypes": [
                "directory"
            ],
            "name": "Sequential Disk Writes",
            "shortName": "Sequential Writes",
            "statistics": [
                "count"
            ],
            "type": "count"
        }
    ],
    "offset": 0
}
```

Retrieving 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
```

<table>
<tr><td><b>Response Code</b></td><td>200 0K</td></tr>
<tr><td><b>Response Body</b></td><td>

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>
```

</td></tr>
<tr><td><b>Response Body (JSON format)</b></td><td>

Discrete valued metric:

```
{
    "countUnit": {
        "abbreviatedName": "Cluster Operation",
        "pluralName": "Cluster Operations",
        "singularName": "Cluster Operation"
    },
    "description": "Time spent performing a cluster operation on
 a DBMS partition",
    "dimensions": [
        {
            "id": "aggregation-level",
            "values": [
                "level0",
                "level1",
                "level2"
            ]
        }
    ],
    "group": "Monitoring DBMS",
    "id": "monitor-cluster-time",
    "instanceTypes": [
        "metrics-engine"
    ],
    "name": "DBMS Cluster",
    "shortName": "DBMS Cluster",
    "statistics": [
        "average",
        "count"
    ],
    "type": "discreteValued",
    "valueUnit": {
        "abbreviatedName": "Msec",
        "pluralName": "Milliseconds",
        "singularName": "Millisecond"
    }
}
```

</td></tr>
<tr><td><b>Response Body (JSON format)</b></td><td>

Continuous valued metric.

```
{
```

</td></tr>
</table>

```
    "description": "Number of active database cleaner threads for
 the specified backend",
    "dimensions": [
        {
            "id": "backend",
            "values": [
                "userroot"
            ]
        }
    ],
    "group": "Directory Backend",
    "id": "backend-active-cleaner-threads",
    "instanceTypes": [
        "directory"
    ],
    "name": "Active Cleaner Threads",
    "shortName": "Active Cleaner Threads",
    "statistics": [
        "average"
    ],
    "type": "continuousValued",
    "valueUnit": {
        "abbreviatedName": "Cleaner",
        "pluralName": "Cleaner Threads",
        "singularName": "Cleaner Thread"
    }
}
```

Performing a Metric Query

A metric query will return the collected sample data from the various monitored instances. The data returned by the query may be presented several different ways depending on client requirements.

Common Query Parameters	• `instanceType` **(multi-valued)** - Type(s) of instances to get data from. Possible values are: ➢ `directory` ➢ `proxy` ➢ `sync` ➢ `metrics-engine` • `instanceLocation` **(multi-valued)** - Location(s) of the instances from which data is collected. • `instanceHostname` **(multi-valued)** - Names of the machines hosting the instances. • `instanceVersion` **(multi-valued)** - Version(s) of the instances providing the data. • `instance` **(multi-valued)** - ID(s) of the instances from which data is collected. Note that the instance ID is the `cn` of the external server. It is the same name as the name displayed 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. Note that 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 value may be thought of as 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`** **(multi-valued)** - 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`** **(multi-valued)** - 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`** **(timezone)** - Specifies the timezone to be used when displaying dates. By default, GMT. The timezone is specified in Java TimeZone format, so "US/Central" specificities 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	By default, the histogram statistic includes all buckets and keeps the raw value for each bucket. However, you can configure graphs that show the percentage of all operations above a given threshold, such as 50 ms. These graphs are useful for focusing on the small percentage of operations in a given category. We recommend that this value be a histogram bucket boundary. If the value falls between boundaries, then 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. Note that if both min and max are specified, the returned value is the sum of all buckets between min and max (including the max).

Data Set Structure

The data set structure is a proprietary data structure that is space-optimized and designed for easy interoperability with charting libraries like Highcharts, FusionCharts, or JFreeChart. This format is ideal for clients capable of performing some simple manipulation of the returned data to fit the target use case. 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}/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 7/7/2012 for all directory and proxy servers 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&startTime=-1d
  &endTime=2012-07-07&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 only 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 sub-series. 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" ]
}
```

Response Body

For queries that request more than one time interval, a timeInterval data set will be returned. Each data point corresponds to the consecutive time interval. Pivoted dimensional values and histogram buckets are included as a series and sub-series.

```
 {
  "type" : "timeInterval",
  "firstSampleTime" : 1344089954000,
  "lastSampleTime" : 1344090254000,
  "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" : [ "count", "average", "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" : "unboundid3",
    "data" : [ "0", "0", "0", "0", "0", "0", "0", "0", "0", "0" ]
  }, {
    "label" : "unboundid35",
    "data" : [ "0", "0", "0", "0", "0", "0", "0", "0", "0", "0" ]
  } ],
  "resolutionInSeconds" : 30
}
```

Chart Image

This API is the simplest way to retrieve and visualize 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`** - Whether to include the chart legend. Default value is true. ➢ **`title`** - A custom title of the chart. Default value is the metric name.

Note: All of the Common Query parameters apply to this resource.

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

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

This results in the following chart.

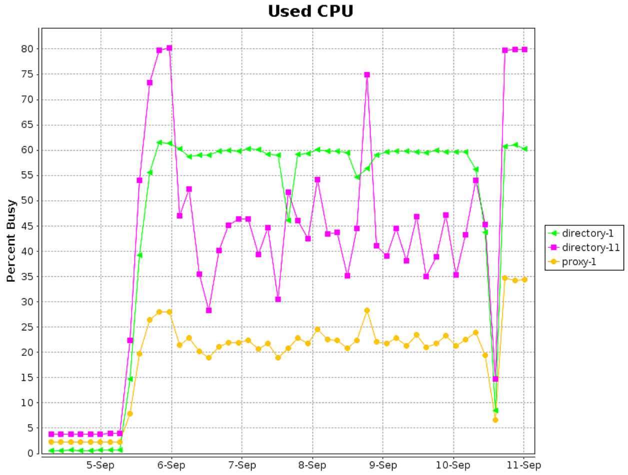


Figure 11: CPU Percent Busy

Google Chart Tools Datasource Protocol

Metrics data may also be requested and presented in tabular format that is fully compatible with Google's Chart Tools Datasource protocol (https://developers.google.com/chart/interactive/docs/dev/implementing_data_source). However, the Google Visualization API query language (the tq request parameter) is not supported. The standard metric query parameters as outlined above will still be used instead. 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` (timezone)** - Specifies the timezone to be used when displaying dates. The Google Visualization API has no notion of time zones and always assumes that the times returned are in local time. The Metrics Engine stores all time stamps in GMT and this is the time that is returned by default. This parameter allows you to configure how the Metrics Engine presents the times in the specified timezone. Usually, the client will pass the user's local timezone in IANA Time Zone Database format, so "US/Central" specificities CST in the United States.

Note: All of the Common Query parameters apply to this resource.

Get 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 OK

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. The CSV format will be shown for readability. 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 code illustrates using Google chart tools:

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

      // Load the Visualization API and the line chart package.
      google.load('visualization', '1.0', {'packages':['corechart']});

      // Set a callback to run when the Google Visualization API is loaded.
      google.setOnLoadCallback(drawChart);

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

      var data = response.getDataTable();

      var visualization = new
 google.visualization.LineChart(document.getElementById('chart_div'));
        visualization.draw(data, null);
      }
    </script>
  </head>
  <body>
    <!--Div that will hold the chart-->
    <div id="chart_div"></div>
  </body>
</html>
```

Pagination

Pagination is supported for both the metrics and instances listing URLs.

Query Parameters	➢ **`limit`** - Specifies the maximum number of results to return. 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 params. ➢ **`offset`** - The index into the total result set where the current response begins.

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>`

Chapter 5 Managing the Metrics Engine

This chapter provides information about managing the UnboundID® Metrics Engine. It includes information about working with logs, notifications, and alerts, as well as information about the command-line tools included with the Metrics Engine.

Topics:

- *Working With Logs*
- *Monitoring the Metrics Engine*
- *Monitoring with JMX*
- *Managing Notifications and Alerts*
- *Command-Line Tools*

Working With Logs

UnboundID® Metrics Engine provides error loggers that provide information about warnings, errors, or significant events that occur within the server. The remainder of this section describes how to create new log publishers, how to configure log rotation and retention, how to manage the file-based error log publisher and how to manage the syslog-based error log publisher.

Creating New Log Publishers

The UnboundID® Metrics Engine provides customization options to help you create your own log publishers with the `dsconfig` command.

When you create a new log publisher, you must also configure the log retention and rotation policies for each new publisher. For more information, see Configuring Log Rotation and Configuring Log Retention.

To Create a New Log Publisher

1. Use the `dsconfig` command in non-interactive mode to create and configure the new log publisher. This example shows how to create a logger that only 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
```

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

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

Compression cannot be disabled or turned off once configured for the logger. Therefore, careful planning is required to determine your logging requirements including log rotation and retention with regards to compressed logs.

2. View the Log Publishers.

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

```
Log Publisher           : Type           : enabled
---------------------------:-------------------:--------
Disconnect Logger          : file-based-access : true
File-Based Access Logger   : file-based-access : true
File-Based Audit Logger    : file-based-access : false
File-Based Debug Logger    : file-based-debug  : false
File-Based Error Logger    : file-based-error  : true
```

```
Replication Repair Logger  : file-based-error  : true
```

To Create a Log Publisher Using dsconfig Interactive Command-Line Mode

1. On the command line, type `bin/dsconfig`.

2. Authenticate to the server by following the prompts.

3. On the UnboundID® Metrics Engine Configuration console main menu, select the option to configure the log publisher.

4. On the **Log Publisher Management** menu, select the option to create a new log publisher.

5. Select the Log Publisher type. In this case, select **File-Based Access Log Publisher**.

6. Type a name for the log publisher.

7. Enable it.

8. Type the path to the log file, relative to the Metrics Engine root. For example, `logs/disconnect.log`.

9. Select the rotation policy you want to use for your log publisher.

10. Select the retention policy you want to use for your log publisher.

11. On the Log Publisher Properties menu, select the option for `log-connects:false`, `log-disconnects:true`, `log-requests:false`, and `log-results:false`.

12. Type `f` to apply the changes.

Configuring Log Rotation

The Metrics Engine allows you to configure the log rotation policy for the server. When any rotation limit is reached, the Metrics Engine rotates the current log and starts a new log. If you create a new log publisher, you must configure at least one log rotation policy.

You can select the following properties:

- **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 100 MB.

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

To Configure the Log Rotation Policy

- Use `dsconfig` to modify the log rotation policy for the access logger.

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --remove "rotation-policy:24 Hours Time Limit Rotation Policy" \
  --add "rotation-policy:7 Days Time Limit Rotation Policy"
```

Configuring Log Retention

The Metrics Engine allows you to configure the log retention policy for each log on the server. When any retention limit is reached, the Metrics Engine removes the oldest archived log prior to creating a new log. Log retention is only effective if you have a log rotation policy in place. If you create a new log publisher, you must configure at least one log retention policy.

- **File Count Retention Policy**. Sets the number of log files you want 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 500 MBytes.
- **Size Limit Retention Policy**. Sets the maximum size of the combined archived logs. The default size limit is 500 MBytes.
- **Custom Retention Policy**. Create a new retention policy that meets your Metrics Engine’s requirements. This will require developing custom code to implement the desired log retention policy.
- **Never Delete Retention Policy**. Used in a rare event that does not require log deletion.

To Configure the Log Retention Policy

- Use `dsconfig` to modify the log retention policy for the access logger.

```
$ bin/dsconfig set-log-publisher-prop \
  --publisher-name "File-Based Access Logger" \
  --set "retention-policy:Free Disk Space Retention Policy"
```

Managing the File-Based 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. 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`, `severe-warning`.
- **Message ID**. Specifies the numeric identifier of the message.
- **Message**. Stores the error, warning, or informational message.

Error Log Example

The following example displays the 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 3.6.0.0
build 20121021003738Z, R12799) has started successfully"
```

To Modify the File-Based Error Logs

- Use `dsconfig` to modify the default File-Based Error Log.

```
$ 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
```

Monitoring the Metrics Engine

The Metrics Engine exposes its monitoring information under the `cn=monitor` entry. Administrators can use various means to monitor the servers, including the UnboundID Metrics Engine, through SNMP, the Metrics Engine Management Console, JConsole, LDAP command-line tools, and the Periodic Stats Logger.

The following monitoring components are accessible:

Table 8: Metrics Engine Monitoring Components

Component	Description
Active Operations	Provides information about the operations currently being processed by the Metrics Engine. Shows 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, including the backend ID, base DN(s), entry counts, entry count for the `cn=admin data`, writability mode, and whether it is a private backend. The following backend monitors are provided: ➢ adminRoot ➢ ads-truststore ➢ alerts ➢ api-users ➢ backup ➢ config ➢ monitor ➢ schema ➢ tasks ➢ userRoot
Berkeley DB JE Environment	Provides information about the state of the Oracle Berkeley DB Java Edition database used by the Metrics Engine backend. Most of the statistics are obtained directly from the Berkeley DB JE.
Client Connections	Provides information about all client connections to the Metrics Engine. The client connection information contains a name followed by an equal sign and a quoted value (e.g., connID="15", connectTime="20100308223038Z", etc.)
Connection Handlers	Provides information about the available connection handlers on the Metrics Engine, which includes the LDAP and LDIF connection handlers. These handlers are used to accept client connections and to read requests and send responses to those clients.
DBMS Activity	Provides the number of DBMS operations and average time per operation on a per-table basis.
DBMS Table	Provides the size, number of table and index scans, number of live records, and number of dead records of each DBMS table.
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, server version, etc.
Index	Provides on each index. The monitor captures the number of keys preloaded, and counters for read/write/remove/open-cursor/read-for-search. 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, operations initiated, completed, and abandoned, etc.

Component	Description
Monitored Server	Provides the internal metrics of the Metrics Engine, including sample import and aggregation times.
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.
Sample Cache	Provides the size, utilization, and eviction of the memory cache that holds metric samples from the DBMS.
Sample Query	Provides the count and average time per metric query operation, including query execution time, percentage of query already in the Sample Cache, and the number of records returned from DBMS for query.
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.

Monitoring Disk Space Usage

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 additional action if the amount of usable space drops below the defined thresholds.

You can configure three thresholds for this monitor:

- **Low space warning threshold**. This threshold is defined as either a percentage or absolute amount of usable space. If the amount of usable space drops below this threshold, then the Metrics Engine will generate an administrative alert but will remain fully functional. It will generate alerts at regular intervals that you configure (such as once a day) unless action is taken to increase the amount of usable space. The Metrics Engine will also generate additional alerts as the amount of usable space is further reduced (e.g., each time the amount of usable space drops below a value 10% closer to the low space error threshold). If an administrator frees up disk space or adds additional capacity, then the server should automatically recognize this and stop generating alerts.

- **Low space error threshold**. This threshold is also defined as either a percentage or absolute size. Once the amount of usable space drops below this threshold, then the server will generate an alert notification and will begin rejecting all operations requested by non-root users with "UNAVAILABLE" results. The server should continue to generate alerts during this time. Once the server enters this mode, then an administrator will have to take some kind of action (e.g., running a command to invoke a task or removing a signal file) before the server will resume normal operation. 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 threshold**. This threshold may also be defined as a percentage or absolute size. Once the amount of usable space drops below this threshold, then the UnboundID® 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.

The threshold values may be specified either as absolute sizes or as percentages of the total available disk space. All values must be specified as absolute values or as percentages. A mix of absolute values and percentages cannot be used. The low space warning threshold must be greater than or equal to the low space error threshold, the low space error threshold must be greater than or equal to the out of space error threshold, and the out of space error threshold must be greater than or equal to zero.

If the out of space error threshold is set to zero, then the server will not attempt to automatically shut itself down if it detects that usable disk space has become critically low. If the amount of usable space reaches zero, then the database will preserve its integrity but may enter a state in which it rejects all operations with an error and requires the server (or at least the affected backends) to be restarted. If the low space error threshold is also set to zero, then the server will generate periodic warnings about low available disk space but will remain fully functional for as long as possible. If all three threshold values are set to zero, then the server will not attempt to warn about or otherwise react to a lack of usable disk space.

Monitoring with JMX

The UnboundID® Metrics Engine supports monitoring the JVM™ through a Java Management Extensions (JMX™) management agent, which can be accessed using JConsole or any other kind of JMX client. The JMX interface provides JVM performance and resource utilization information for applications running Java. You can monitor generic metrics exposed by the JVM itself, including memory pools, threads, loaded classes, and MBeans, as well as all the monitor information that the Metrics Engine provides. You can also subscribe to receive JMX notifications for any administrative alerts that are generated within the server.

Running JConsole

Before you can access JConsole, you must configure and enable the JMX Connection Handler for the Metrics Engine using the `dsconfig` tool. See Configuring the JMX Connection Handler and Alert Handler.

To invoke the JConsole executable, type `jconsole` on the command line. If *JDK_HOME* is not set in your path, you can access JConsole in the `bin` directory of the `JDK_HOME` path.

Monitoring the Metrics Engine Using JConsole

You can set up JConsole to monitor the Metrics Engine using a remote process. Make sure to enable the JMX Connection Handler and to assign at least the `jmx-read` privilege to a regular user account (the `jmx-notify` privilege is required to subscibe to receive JMX notifications). Do not use a root user account, as this would pose a security risk.

Managing Notifications and Alerts

The UnboundID® Metrics Engine provides delivery mechanisms for account status notifications and administrative alerts using SMTP, JMX, or SNMP in addition to standard error logging. Alerts and events reflect state changes within the server that may be of interest to a user or monitoring service. Notifications are typically the delivery of an alert or event to a user or monitoring service. Account status notifications are only delivered to the account owner notifying a change in state in the account.

This chapter presents the following topics:

Working with Administrative Alert Handlers

The UnboundID® Metrics Engine provides mechanisms to send alert notifications to administrators when significant problems or events occur during processing, such as problems during server startup or shutdown. 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 will be 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 Simple Mail Transfer Protocol (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 via SNMP 2c.
- **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.

Administrative Alert Types

If enabled, the Metrics Engine can generate administrative alerts when the events occur. The Alert types are presented in the table below.

Table 9: Administrative Alert Types

Alert Type	Severity	Description
access-control-change	Info	Indicates that access control configuration has been changed.
access-control-disabled	Warning	Indicates that access control evaluation has been disabled.
access-control-enabled	Info	Indicates that access control evaluation has been enabled.
access-control-parse-failure	Error	Indicates that an error occurred while attempting to parse an access control rule.
access-log-criteria-matched	Info	Indicates that an access log message matched the criteria for the admin alert access log publisher.
backend-end-initialization-failed	Error	Indicates that an attempt to initialize the backend failed.
cannot-acquire-shared-backend-lock	Error	Indicates that an error occurred while attempting to acquire a shared backend lock.
cannot-copy-schema-files	Error	Indicates that an error occurred while attempting to copy schema files during a schema update.
cannot-decode-entry	Error	Indicates that an error occurred while attempting to decode an entry stored in a backend.
cannot-find-recurring-task	Error	Indicates that the definition for a recurring task could not be found.
cannot-register-backend	Error	Indicates that an error occurred while trying to register a backend.
cannot-register-shared-backend-lock	Error	Indicates that an error occurred while trying to release a shared backend lock.
cannot-rename-current-task-file	Error	Indicates that an error occurred while trying to rename the current task backing file.
cannot-rename-new-task-file	Error	Indicates that an error occurred while trying to rename the new task backing file.
cannot-restore-backup	Error	Indicates that an error occurred while trying to restore a backup.
cannot-schedule-recurring-task-iteration	Error	Indicates that an error occurred while trying to schedule a recurring task iteration.
cannot-write-configuration	Error	Indicates that an error occurred while trying to write the updated server configuration.
cannot-write-new-schema-files	Error	Indicates that an error occurred while trying to update schema files.
cannot-write-server-state-file	Error	Indicates that an error occurred while attempting to write the server status file.
cannot-write-task-backing-file	Error	Indicates that an error occurred while trying to write the task backing file.
config-change	Info	Indicates the a configuration change has been made in the server.

Alert Type	Severity	Description
deadlock-detected	Error	Indicates that a deadlock has been detected in the JVMTM in which the Metrics Engine is running.
duplicate-alerts-suppressed	Error	Indicates that duplicate alert notifications have been suppressed.
entering-lockdown-mode	Warning	Indicates that the server is entering lockdown mode, in which it will only allow operations from root users.
external-config-file-edit-handled	Warning	Indicates that the server has detected an external edit to the configuration file with the server online, but that it was able to copy the modifications into a separate file without applying them.
external-config-file-edit-handled	Warning	Indicates that the server has detected an external edit to the configuration file with the server online, but that it was able to copy the modifications into a separate file without applying them.
external-config-file-edit-lost	Error	Indicates that the server has detected an external edit to the configuration file with the server online and was unable to copy the modifications into a separate file.
external-server-initialization-failed	Error	Indicates that an attempt to initialize an external server failed.
force-gc-complete	Info	Indicates that the server has completed a forced garbage collection.
force-gc-starting	Info	Indicates that the server is about to force a synchronous garbage collection.
index-degraded	Warning	Indicates that a backend is operating with a degraded index that needs to be rebuilt before that index may be used.
index-rebuild-completed	Info	Indicates that a backend is in the progress of rebuilding one or more indexes.
index-rebuild-in-progress	Info	Indicates that a backend is in the progress of rebuilding one or more indexes.
invalid-privilege	Warning	Indicates that a user has been configured with an invalid privilege.
je-recovery-required	Fatal	Indicates that a backend using the Oracle Berkeley DB Java Edition (JE) has encountered a server error and requires recovery.
large-attribute-update-error	Error	Indicates that an error occurred while updating large attribute information in the Metrics Engine, and the large attribute information may be out of sync with the entry contents.
ldap-connection-handler-cannot-listen	Fatal	Indicates that an error occurred when the LDAP connection handler tried to start listening for client connections and therefore the connection handler will be disabled.
ldap-connection-handler-consecutive-failures	Fatal	Indicates that the LDAP connection handler has experienced consecutive failures and will be disabled.
ldap-connection-handler-uncaught-error	Fatal	Indicates that the LDAP connection handler has encountered an uncaught error and will be disabled.

Alert Type	Severity	Description
ldif-backend-cannot-write	Error	Indicates that an error occurred while trying to write the backing file for the LDIF backend.
ldif-connection-handler-io-error	Error	Indicates that the LDIF connection handler encountered an I/O error that prevented it from processing.
ldif-connection-handler-parse-error	Error	Indicates that the LDIF connection handler encountered an I/O error that has prevented it from processing.
leaving-lockdown-mode	Info	Indicates that the server is leaving lockdown mode and resuming normal operation.
logging-error	Error	Indicates that an error occurred while attempting to log a message.
low-disk-space-error	Error	Indicates that the amount of usable disk space has dropped below the low space error threshold.
low-disk-space-warning	Warning	Indicates that the amount of usable disk space has dropped below the configured low space warning threshold.
low-disk-space-warning	Warning	Indicates that the amount of usable disk space has dropped below the configured low space warning threshold.
out-of-disk-space-error	Fatal	Indicates that the amount of usable disk space has dropped below the configured out of space error threshold.
replication-backlogged	Warning	Indicates that the replication backlog has exceeded the replication backlog count alert threshold for longer than the replication backlog duration alert threshold.
replication-backlog	Warning	Indicates that a replication server has known changes that have not been applied yet.
replication-monitor-data-unresolved	Warning	Indicates that replication monitor data is unavailable from cn=monitor.
replication-unresolved-conflict	Error	Indicates that the multi-master replication cannot automatically resolve a conflict.
replication-plugin-message-serialization-failure	Warning	Indicates that the replication plug-in failed to serialize or de-serialize a replication message.
replication-replay-failed	Error	Indicates that the server has failed to replay a replication operation.
replication-server-changelog-failure	Error	Indicates that the replication server encountered an error while accessing the replication changelog database.
replication-server-listen-failure	Error	Indicates that the replication server encountered an error while trying to listen on the configured replication port. There may be another application listening on the same port or the replication server host name may not be resolvable. Check the replication server configuration.
replication-unresolved-conflict	Error	Indicates that the server has detected a replication conflict that could not be resolved.
server-jvm-paused	Warning	Indicates that the server's JVM paused for some reason possibly due to misconfiguration.
server-shutting-down	Info	Indicates that the server has begun the shutdown process.
server-started	Info	Indicates that the server has completed that startup process.

Alert Type	Severity	Description
server-starting	Info	Indicates that the server is starting.
system-nanotime-stopped	Error	Indicates that System.nanoTime() has stopped advancing.
thread-exit-holding-lock	Error	Indicates that a thread has exited while still holding one or more locks.
uncaught-exception	Error	Indicates that the server has detected an uncaught exception that may have caused a thread to terminate.
unique-attribute-sync-conflict	Error	Indicates that the server has detected a unique attribute conflict that was introduced from synchronization.
unique-attribute-sync-error	Error	Indicates that the server has encountered an error while attempting to detect unique attribute conflicts via synchronization.
unrecognized-alert-type	Error	Indicates that an unrecognized alert type was encountered. This should never be used for any alert that is generated, but only for cases in which the server needs to create an alert type from a string but the string does not match any recognized type.
user-defined-error	Error	Indicates that an error alert notification has been generated by third-party code.
user-defined-fatal	Fatal	Indicates that a fatal error alert notification was generated by third-party code.
user-defined-info	Info	Indicates that an informational alert notification was generated by third-party code.
user-defined-warning	Warning	Indicates that a warning alert notification was generated by third-party code.
worker-thread-caught-error	Error	Indicates that a worker thread caught an unexpected error.
work-queue-backlogged	Error	Indicates that the work queue has become significantly backlogged and operations have been required to wait a significant length of time to be processed.
work-queue-full	Error	Indicates that the work queue is full and has rejected a client request.
work-queue-no-threads-remaining	Fatal	Indicates that all worker threads have been terminated due to errors and the server must shut down.
worker-thread-caught-error	Error	Indicates that a worker thread has caught an unexpected error that has caused it to be terminated.

Configuring the JMX Connection Handler and Alert Handler

You can configure the JMX connection handler and alert handler respectively using the `dsconfig` tool. Any user allowed to receive JMX notifications must have the `jmx-read` and `jmx-notify` privileges. By default, these privileges are not granted to any users (including root users or global administrators). For security reasons, we recommend that you create a separate user account that does not have any other privileges but these. Although not shown in this section, you can configure the JMX connection handler and alert handler using `dsconfig` in interactive command-line mode, which is visible on the "Standard" object menu.

To Configure the JMX Connection Handler

1. Use `dsconfig` to enable the JMX Connection Handler.

```
$ bin/dsconfig set-connection-handler-prop \
  --handler-name "JMX Connection Handler" \
  --set enabled:true \
  --set listen-port:1689
```

2. Add a new non-root user account with the `jmx-read` and `jmx-notify` privileges. This account can be added using the `ldapmodify` tool using an LDIF representation like:

```
dn: cn=JMX User,cn=Root DNs,cn=config
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: ds-cfg-root-dn-user
givenName: JMX
sn: User
cn: JMX User
userPassword: password
ds-cfg-inherit-default-root-privileges: false
ds-cfg-alternate-bind-dn: cn=JMX User
ds-privilege-name: jmx-read
ds-privilege-name: jmx-notify
```

To Configure the JMX Alert Handler

- Use `dsconfig` to configure the JMX Alert Handler.

```
$ bin/dsconfig set-alert-handler-prop --handler-name "JMX Alert Handler" \
  --set enabled:true
```

Configuring the SMTP Alert Handler

By default, there is no configuration entry for an SMTP alert handler. To create a new instance of an SMTP alert handler, use the `dsconfig` tool.

Configuring the SMTP Alert Handler

- Use the `dsconfig` tool to configure the SMTP Alert Handler.

```
$ 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\%\%"
```

Configuring the SNMP Subagent Alert Handler

You can configure the SNMP Subagent alert handler using the `dsconfig` tool, which is visible at the "Standard" object menu. Before you begin, you need an SNMP Subagent capable of communicating via SNMP2c. For more information on SNMP, see Monitoring Using SNMP.

To Configure the SNMP Subagent Alert Handler

- Use `dsconfig` to configure the SNMP subagent alert handler. The `server-host-name` is the address of the system running the SNMP subagent. The `server-port` is the port number on which the subagent is running. The `community-name` is the name of the SNMP community that is used for the traps.

  The Metrics Engine also supports a SNMP Alert Handler, which is used in deployments that do not enable an SNMP subagent.

```
$ bin/dsconfig set-alert-handler-prop \
  --handler-name "SNMP Subagent Alert Handler" \
  --set enabled:true \
  --set server-host-name:host2 \
  --set server-port:162 \
  --set community-name:public
```

Working with the Alerts Backend

The Metrics Engine stores recently generated admin alerts in an Alerts Backend 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 in the `cn=alerts` backend.

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 Metrics Engine Management Console.

To View Information in the Alerts Backend

- Use `ldapsearch` to view the admin 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
```

""""

To Modify the Alert Retention Time

1. Use `dsconfig` to change the maximum time information about generated admin alerts is retained in the Alerts backend. After this time, the information gets 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"
   ```

2. 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
   ```

To Configure Duplicate Alert Suppression

- Use `dsconfig` to configure the maximum number of times an alert is generated within a particular timeframe 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. The `duplicate-alert-limit` property specifies the maximum number of duplicate alert messages should be sent over the administrative alert framework in the time limit specified in the `duplicate-alert-time-limit` property.

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

Command-Line Tools

The UnboundID® Metrics Engine provides a full suite of command-line tools necessary 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.

This chapter presents the following topics:

Using the Help Option

The following table provides a brief description of each command-line tool. You can view detailed argument options and examples by typing `--help` with the command.

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

For those utilities that support additional subcommands (`dsconfig`, `dsframework`, `dsreplication`, `manage-account`), you can get a list of the subcommands by typing `--help-subcommands`.

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

You can also get more detailed subcommand information by typing `--help` with the specific subcommand.

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

Note: For detailed information and examples of the command-line tools, see the *UnboundID Directory Server Command-Line Tool Reference*.

Available Command-Line Utilities

The Metrics Engine provides the following command-line utilities, which can be run directly in interactive or non-interactive modes or can be included in scripts.

Table 10: Command-Line Utilities

Command-Line Tools	Description
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 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*.
dump-dns	Obtain a listing of all of the DNs for all entries below a specified base DN in the Metrics Engine.
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.

Command-Line Tools	Description
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.
list-backends	List the backends and base DNs configured in the Metrics Engine.
revert-update	Returns a server to the version before the last update was performed. Unlike the `update` tool, this tool operates on the local instance from which it is invoked. This tool relies on files from the history directory that are created during an update to restore the server to a prior state. It should be noted that this tool does not revert database files to prior states. Therefore, any changes made to the directory data between the time of the update and time of reversion will be lost.
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 a directory server 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 you wish to update. Then, use the `update` tool from the new server package to update the older version of the server. Before upgrading a server, you should ensure that it is capable of starting without severe or fatal errors. During the update process, the server is stopped if running, then the update performed, and a check is made to determine if the newly updated server starts without major errors. If it cannot start cleanly, the update will be backed out and the server returned to its prior state. See the `revert-update` tool for information on reverting an update.

Managing the tools.properties File

The UnboundID® Metrics Engine supports the use of a tools properties file that simplifies command-line invocations by reading in a set of arguments for each tool from a text file. Each property is in the form of name/value pairs that define predetermined values for a tool’s arguments. Properties files are convenient when quickly testing the Metrics Engine in multiple environments.

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. You can override all of the Metrics Engine's command-line utilities with a properties file using the `config/tools.properties` file.

Creating a Tools Properties File

You can create a properties file with a text editor by specifying each argument, or option, using standard Java properties file format (name=value). For example, you can create a simple properties file that define a set of LDAP connection parameters as follows:

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

Next, you can specify the location of the file using the `--propertiesFilePath /path/to/File` option with the command-line tool. For example, if you save the previous properties file as `bin/mytool.properties`, you can 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 of any kind around values. Any spaces or special characters should be escaped. For example,

```
bindDN=cn=QA\ Managers,ou=groups,dc=example,dc=com
```

The following is not allowed as it contains quotation marks:

```
bindDN=cn="QA Managers,ou=groups,dc=example,dc=com"
```

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 uses `port=1389`.

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

Specifying Default Properties Files

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

If you place a custom properties file that has a different filename as `tools.properties` in this default location, you need to specify the path using the `--propertiesFilePath` option. If you make changes to the `tools.properties` file, you do not need the `--propertiesFilePath` option. See the examples in the next section.

Evaluation Order Summary

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 takes 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 `<directory-instance>/config/`.
- 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`).

Evaluation Order Example

Given the following properties file that is saved as `<server-root>/bin/tools.properties`:

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

The Metrics Engine locates a command-line options in a specific priority order.

1. All options presented with the tool on the command line take precedence over any options in any properties file. In the following example, the client request is run with the options specified on the command line (port and baseDN). The command uses the `bindDN` and `bindPassword` arguments specified in the properties file.

   ```
   $ bin/ldapsearch --port 2389 --baseDN ou=People,dc=example,dc=com \
     --propertiesFilePath bin/tools.properties “(objectclass=*)”
   ```

2. Next, if you specify the properties file using the `--propertiesFilePath` option and no other command-line options, the Metrics Engine uses the specified properties file as follows:

   ```
   $ bin/ldapsearch --propertiesFilePath bin/tools.properties \
     “(objectclass=*)”
   ```

3. If no options are presented with the tool on the command line and the `--noPropertiesFile` option is not present, the Metrics Engine attempts to locate any default `tools.properties` file in the following location:

   ```
   <server-root>/config/tools.properties
   ```

   Assume that you move your tools.properties file from `<server-root>/bin` to the `<server-root>/config` directory. You can then run your tools as follows:

   ```
   $ bin/ldapsearch "(objectclass=*)"
   ```

The Metrics Engine can be configured so that it does not search for any properties file by using the `--noPropertiesFile` option. This options tells the Metrics Engine to use only those options specified on the command line. The `--propertiesFilePath` and `--noPropertiesFile` options are mutually exclusive and cannot be used together.

4. If no default `tools.properties` file is found and no options are specified with the command-line tool, then the tool generates an error for any missing arguments.

Chapter 6 Troubleshooting the Metrics Engine

This chapter provides the common problems and potential solutions that might occur when running UnboundID® Metrics Engine.

This chapter presents the following information:

Topics:

- *Debugging the Metrics Engine*
- *Working with the Troubleshooting Tools*
- *Metrics Engine Troubleshooting Tools*
- *Troubleshooting Resources for Java Applications*
- *Troubleshooting Resources in the Operating System*
- *Troubleshooting Performance Problems*
- *Troubleshooting the Metrics Engine API*
- *Common Problems and Potential Solutions*

Debugging the Metrics Engine

You can enable the JVM debugging options to track garbage collection data for your system. The options can impact JVM performance, but they provide valuable data to tune your server when troubleshooting garbage collection issues. 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 the JVM to use when running the server to provide additional detail.

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

To run the Metrics Engine with these options, edit the `config/java.properties` file and add them to the end of the line that begins with "`start-metrics-engine.java-args`". After the file has been saved, invoke the following command to make those new arguments take effect the next time the server is started:

```
$ bin/dsjavaproperties
```

Working with the Troubleshooting Tools

The UnboundID® Metrics Engine provides a highly-reliable service that satisfies your company's objectives. However, if problems do arise (whether from issues in the Metrics Engine itself or a supporting component, like the JVM, operating system, or hardware), then it is essential to be able to diagnose the problem quickly to determine the underlying cause and the best course of action to take towards resolving it.

Working with the Collect Support Data Tool

The Metrics Engine provides a significant amount of information about its current state including any problems that it has encountered during processing. If a problem occurs, the first step is to run the `collect-support-data` tool in the `bin` directory. The tool aggregates all relevant support files into a zip file that administrators can send to your authorized support provider for analysis. The tool also runs data collector utilities, such as `jps`, `jstack`, and `jstat` plus other diagnostic tools for Solaris and Linux machines, and bundles the results in the zip file.

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 varies between systems. For example, on Solaris Zone, configuration information is gathered using commands like `zonename` and `zoneadm`. However, the tool always tries to get the same information across all systems for the target Metrics Engine. 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.

To Run the Collect Support Data Tool

1. Go to the server root directory.

2. Use the `collect-support-data` tool. Make sure to 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 your Authorized Support Provider.

Metrics Engine Troubleshooting Tools

The UnboundID® Metrics Engine provides a set of tools that can also be used to obtain information for diagnosing and solving problems.

Server Version Information

If it becomes necessary to contact your authorized support provider, then it will be important to provide precise information about the version of the Metrics Engine software that is in use. If the server is running, then this information can be obtained from the "`cn=Version,cn=monitor`" entry. It can also be obtained using the command:

```
$ bin/status --fullVersion
```

This command outputs a number of important pieces of information, including:

- Major, minor, point and patch version numbers for the server.
- Source revision number from which the server was built.
- Build information including build ID with time stamp, OS, user, Java and JVM version for the build.
- Auxiliary software versions: Jetty, JZlib, SNMP4j (SNMP4J, Agent, Agentx), Groovy, UnboundID LDAP SDK for Java, and UnboundID Server SDK.

Embedded Profiler

If the Metrics Engine appears to be running slowly, then it is helpful to know what operations are being processed in the server. The JVM Stack Trace monitor entry can be used to obtain a point-in-time snapshot of what the server is doing, but in many cases, it might be useful to have information collected over a period of time.

The embedded profiler is configured so that it is always available but is not active by default so that it has no impact on the performance of the running server. Even when it is running, it has a relatively small impact on performance, but it is recommended that it remain inactive when it is not needed. It can be controlled using the `dsconfig` tool or the web administration console by managing the "Profiler" configuration object in the "Plugin" object type, available at the standard object level. The `profile-action` property for this configuration object can have one of the following values:

- **start** – Indicates that the embedded profiler should start capturing data in the background.
- **stop** – Indicates that the embedded profiler should stop capturing data and write the information that it has collected to a `logs/profile{timestamp}` file.
- **cancel** – Indicates that the embedded profiler should stop capturing data and discard any information that it has collected.

Any profiling data that has been captured can be examined using the `profiler-viewer` tool. This tool can operate in either a text-based mode, in which case it dumps a formatted text representation of the profile data to standard output, or it can be used in a graphical mode that allows the information to be more easily understood.

To Invoke the Profile Viewer in Text-based Mode

- Run the `profile-viewer` command and specify the captured log file using the `--fileName` option.

```
$ bin/profile-viewer --fileName logs/profile.20110101000000Z
```

To Invoke the Profile Viewer in GUI Mode

- Run the `profile-viewer` command and specify the captured log file using the `--fileName` option. To invoke GUI mode, add the option `--useGUI`.

```
$ bin/profile-viewer --fileName logs/profile.20110101000000Z --useGUI
```

Troubleshooting Resources for Java Applications

Because the UnboundID® Metrics Engine is written entirely in Java, it is possible to use standard Java debugging and instrumentation tools when troubleshooting problems with the Metrics Engine. In many cases, obtaining the full benefit of these tools requires access to the Metrics Engine source code. These Java tools should be used under the advisement of your authorized support provider.

Java Troubleshooting Documentation (Oracle/Sun JDK)

There are a number of documents providing general information about troubleshooting Java-based applications. Some of these documents include:

- http://www.oracle.com/technetwork/java/javase/index-138283.html – Troubleshooting Java SE

- http://www.oracle.com/technetwork/java/javase/index-137495.html – Troubleshooting Guide for Java SE 6 with HotSpot VM
- http://www.sun.com/bigadmin/hubs/java/troubleshoot/ – BigAdmin Page on Java SE Troubleshooting
- http://www.oracle.com/technetwork/java/javase/tools6-unix-139447.html – Tools for troubleshooting Java on Solaris and Linux

Java Troubleshooting Tools (Oracle/Sun JDK)

The Java Development Kit provides a number of very useful tools to obtain information about Java applications and diagnosing problems. These tools are not included with the Java Runtime Environment (JRE), so the full Java Development Environment (JDK) should always be installed and used to run the UnboundID® Metrics Engine.

jps

The `jps` tool is a Java-specific version of the UNIX `ps` tool. It can be used to obtain a list of all Java processes currently running and their respective process identifiers. When invoked by a non-root user, it will list only Java processes running as that user. When invoked by a root user, then it lists all Java processes on the system.

This tool can be used to see if the Metrics Engine is running and if a process ID has been assigned to it. This process ID can be used in conjunction with other tools to perform further analysis.

This tool can be run without any arguments, but some of the more useful arguments that include:

- **-v** – Includes the arguments passed to the JVM for the processes that are listed.
- **-m** – Includes the arguments passed to the main method for the processes that are listed.
- **-l** – (lowercase L). Include the fully qualified name for the main class rather than only the base class name.

Additional documentation for the `jps` tool is available at:

- http://java.sun.com/javase/6/docs/techs/tools/share/jps.html

jstack

The `jstack` tool is used to obtain a stack trace of a running Java process, or optionally from a core file generated if the JVM happens to crash. A stack trace can be extremely valuable when trying to debug a problem, because it provides information about all threads running and exactly what each is doing at the point in time that the stack trace was obtained.

Stack traces are helpful when diagnosing problems in which the server appears to be hung or behaving slowly. Java stack traces are generally more helpful than native stack traces, because Java threads can have user-friendly names (as do the threads used by the UnboundID® Metrics

Engine), and the frame of the stack trace may include the line number of the source file to which it corresponds. This is useful when diagnosing problems and often allows them to be identified and resolved quickly.

To obtain a stack trace from a running JVM, use the command:

```
jstack {processID}
```

where {processID} is the process ID of the target JVM as returned by the `jps` command. To obtain a stack trace from a core file from a Java process, use the command:

```
jstack {pathToJava} {pathToCore}
```

where `{pathToJava}` is the path to the java command from which the core file was created, and `{pathToCore}` is the path to the core file to examine. In either case, the stack trace is written to standard output and includes the names and call stacks for each of the threads that were active in the JVM.

In many cases, no additional options are necessary. The "`-l`" option can be added to obtain a long listing, which includes additional information about locks owned by the threads. The “`-m`” option can be used to include native frames in the stack trace.

Additional documentation for the `jstack` tool is available at http://java.sun.com/javase/6/ docs/ techs/tools/share/jstack.html.

jmap

The `jmap` tool is used to obtain information about the memory consumed by the JVM. It is very similar to the native `pmap` tool provided by many operating systems. As with the `jstack` tool, `jmap` can be invoked against a running Java process by providing the process ID, or against a core file, like:

```
jmap {processID}
jmap {pathToJava} {pathToCore}
```

Some of the additional arguments include:

- **-dump:live,format=b,file=filename** – Dump the live heap data to a file that can be examined by the `jhat` tool
- **-heap** – Provides a summary of the memory used in the Java heap, along with information about the garbage collection algorithm in use.
- **-histo:live** – Provides a count of the number of objects of each type contained in the heap. If the “`:live`” portion is included, then only live objects are included; otherwise, the count include objects that are no longer in use and are garbage collected.

Additional information about the `jmap` tool can be found at http://java.sun.com/javase/6/ docs/ techs/tools/share/jmap.html.

jhat

The `jhat` (Java Heap Analysis Tool) utility provides the ability to analyze the contents of the Java heap. It can be used to analyze a heap dump file, which is generated if the Metrics Engine encounters an out of memory error (as a result of the "`-XX:+HeapDumpOnOutOfMemoryError`" JVM option) or from the use of the `jmap` command with the "`-dump`" option.

The `jhat` tool acts as a web server that can be accessed by a browser in order to query the contents of the heap. Several predefined queries are available to help determine the types of objects consuming significant amounts of heap space, and it also provides a custom query language (OQL, the Object Query Language) for performing more advanced types of analysis.

The `jhat` tool can be launched with the path to the heap dump file, like:

```
jhat /path/to/heap.dump
```

This command causes the `jhat` web server to begin listening on port 7000. It can be accessed in a browser at `http://localhost:7000` (or `http://address:7000` from a remote system). An alternate port number can be specified using the "`-port`" option, like:

```
jhat -port 1234 /path/to/heap.dump
```

To issue custom OQL searches, access the web interface using the URL `http://localhost:7000/oql/` (the trailing slash must be provided). Additional information about the OQL syntax may be obtained in the web interface at `http://localhost:7000/oqlhelp/`. Additional information for the `jhat` tool may be found at `http://java.sun.com/javase/6/docs/techs/tools/share/jhat.html`.

jstat

The `jstat` tool is used to obtain a variety of statistical information from the JVM, much like the `vmstat` utility that can be used to obtain CPU utilization information from the operating system. The general manner to invoke it is as follows:

```
jstat {type} {processID} {interval}
```

The `{interval}` option specifies the length of time in milliseconds between lines of output. The `{processID}` option specifies the process ID of the JVM used to run the Metrics Engine, which can be obtained by running `jps` as mentioned previously. The `{type}` option specifies the type of output that should be provided. Some of the most useful types include:

- **-class** – Provides information about class loading and unloading.
- **-compile** – Provides information about the activity of the JIT complex.
- **-printcompilation** – Provides information about JIT method compilation.
- **-gc** – Provides information about the activity of the garbage collector.
- **-gccapacity** – Provides information about memory region capacities.

Java Diagnostic Information

In addition to the tools listed in the previous section, the JVM can provide additional diagnostic information in response to certain events. UnboundID® Metrics Engine supports the Sun/Oracle JDK 1.6.0_31 and the IBM JRE 1.6.0 IBM J9 2.4 for 32-bit and 64-bit architectures.

JVM Crash Diagnostic Information

If the JVM itself should happen to crash for some reason, then it generates a fatal error log with information about the state of the JVM at the time of the crash. By default, this file is named `hs_err_pid{processID}.log` and is written into the base directory of the Metrics Engine installation. This file includes information on the underlying cause of the JVM crash, information about the threads running and Java heap at the time of the crash, the options provided to the JVM, environment variables that were set, and information about the underlying system. More information about the content that may be written to this log file may be found at `http://java.sun.com/javase/6/webs/trouble/TSG-VM/html/felog.html`.

Java Troubleshooting Tools (IBM JDK)

The UnboundID® Metrics Engine can be run on machines using the IBM JDK. IBM provides Java monitoring and diagnostic tools that can assess JVM performance and troubleshoot any Java application failures. The following tools are available for the IBM JDK. For more detailed information, see the IBM Developers web-site for a description of each tool:

- **Health Center Version 1.3**. Monitors Java applications running on the JDK. The tool provides profiling information for performance, memory usage, system environment, object allocations and other areas.
- **Memory Analyzer Version 1.1**. Analyzes Java heap memory using a system or heap dump snapshot of a Java process.
- **Garbage Collection and Memory Visualizer Version 2.6**. Fine-tunes Java performance by optimizing garbage collection performance, provides Java heap recommendations based on peak and average memory usage, and detects memory leaks and heap exhaustion.
- **Dump Analyzer Version 2.2**. Helps troubleshoot the cause of any application failure using an operating system dump. The tool detects any potential problems based on state, thread, stack information and error messages that were generated when the application failed.
- **Diagnostics Collector Version 1.0**. Collects diagnostic and context information during Java runtime processes that failed. The tool verifies your Java diagnostic configuration to ensure that disabled diagnostic analyzers are enabled to troubleshoot a problem.
- **IBM Diagnostic Tool Framework for Java Version 1.5**. Runs on dump data extracted by the `jextract` tool. The tool checks memory locations, Java threads, Java objects and other important diagnostic areas when the system dump was produced.

Troubleshooting Resources in the Operating System

The underlying operating system also provides a significant amount of information that can help diagnose issues that impact the performance and the stability of the Metrics Engine. In some cases, problems with the underlying system can be directly responsible for the issues seen with the Metrics Engine, and in others system, tools can help narrow down the cause of the problem.

Identifying Problems with the Underlying System

If the underlying system itself is experiencing problems, it can adversely impact the function of applications running on it. Places to look for problems in the underlying system include:

- The system log file (`/var/adm/messages` on Solaris and `/var/log/messages` on Linux). Information about faulted or degraded devices or other unusual system conditions are written there.
- On Solaris systems, if the fault management system has detected a problem with a system component, information about that problem is obtain by running the `fmdump` command.
- If the ZFS filesystem is in use, then the `zpool` status command provides information about read errors, write errors, or data checksum errors.

Examining CPU Utilization

Observing CPU utilization for the Metrics Engine process and the system as a whole provides clues as to the nature of the problem.

System-Wide CPU Utilization

To investigate CPU consumption of the system as a whole, use the `vmstat` command with a time interval in seconds, like:

```
vmstat 5
```

The specific output of this command varies between different operating systems, but it includes the percentage of the time the CPU was spent executing user-space code (user time), the percentage of time spent executing kernel-space code (system time), and the percentage of time not executing any code (idle time).

If the CPUs are spending most of their time executing user-space code, the available processors are being well-utilized. If performance is poor or the server is unresponsive, it can indicate that the Metrics Engine is not optimally tuned. If there is a high system time, it can indicate that the system is performing excessive disk and/or network I/O, or in some cases, there can be some other system-wide problem like an interrupt storm. If the system is mostly idle but the Metrics Engine is performing poorly or is unresponsive, there can be a resource constraint elsewhere (for example, waiting on disk or memory access, or excessive lock contention), or the JVM can be performing other tasks like stop-the-world garbage collection that cannot be run heavily in parallel.

Per-CPU Utilization

To investigate CPU consumption on a per-CPU basis, use the `mpstat` command with a time interval in seconds, like:

```
mpstat 5
```

On Linux systems, it might be necessary to add "`-P ALL`" to the command, like:

```
mpstat -P ALL 5
```

Among other things, this shows the percentage of time each CPU has spent in user time, system time, and idle time. If the overall CPU utilization is relatively low but `mpstat` reports that one CPU has a much higher utilization than the others, there might be a significant bottleneck within the server or the JVM might be performing certain types of garbage collection which cannot be run in parallel. On the other hand, if CPU utilization is relatively even across all CPUs, there is likely no such bottleneck and the issue might be elsewhere.

Per-Process Utilization

To investigate CPU consumption on a per-process basis, use the `prstat` tool on Solaris or the `top` utility on Linux. If a process other than the Java process used to run the Metrics Engine is consuming a significant amount of available CPU, it might be interfering with the ability of the Metrics Engine to run effectively.

If the `mpstat` command showed that one CPU was much more heavily utilized than the others, it might be useful to identify the thread with the highest CPU utilization as it is likely the one that is a bottleneck preventing other threads from processing. On Solaris, this can be achieved by using the `prstat` command with the "`-L`" option, like:

```
prstat -L -p {processID}
```

This command will cause each thread to be displayed on a separate line, with the LWPID (lightweight process identifier) displayed as the last item on each line, separated from the process name by a slash. The thread that is currently consuming the largest amount of CPU will be displayed at the top of the list, and the `pstack` command can be used to identify which thread is responsible.

Examining Disk Utilization

If the underlying system has a very high disk utilization, it can adversely impact Metrics Engine performance. It could delay the ability to read or write database files or write log files. It could also raise concerns for server stability if excessive disk I/O inhibits the ability of the cleaner threads to keep the database size under control.

The `iostat` tool may be used to obtain information about the disk activity on the system. On Solaris systems, this should be invoked using the "`-x`" and "`-n`" arguments, like:

```
iostat -x -n 5
```

On Linux systems, `iostat` should be invoked with the "`-x`" argument, like:

```
iostat -x 5
```

A number of different types of information will be displayed, but to obtain an initial feel for how busy the underlying disks are, look at the "%b" column on Solaris and the "%util" column on Linux. Both of these fields show the percentage of the time that the underlying disks are actively servicing I/O requests. A system with a high disk utilization likely exhibits poor Metrics Engine performance.

If the high disk utilization is on one or more disks that are used to provide swap space for the system, the system might not have enough free memory to process requests. As a result, it might have started swapping blocks of memory that have not been used recently to disk. This can cause very poor server performance. It is important to ensure that the server is configured appropriately to avoid this condition. If this problem occurs on a regular basis, then the server is likely configured to use too much memory. If swapping is not normally a problem but it does arise, then check to see if there are any other processes running, which are consuming a significant amount of memory, and check for other potential causes of significant memory consumption (for example, large files in a `tmpfs` filesystem).

On Solaris systems using ZFS, you can use the `zpool iostat {interval}` command to obtain information about I/O activity on a per-pool basis. While this command provides a useful display of the number of read and write operations and the amount of data being read from and written to the disks, it does not actually show how busy the underlying disks. As a result, the `zpool iostat` command is generally not as useful as the traditional `iostat` command for identifying potential I/O bottlenecks.

Examining Process Details

There are a number of tools provided by the operating system that can help examine a process in detail.

ps

The standard `ps` tool can be used to provide a range of information about a particular process. For example, the command can be used to display the state of the process, the name of the user running the process, its process ID and parent process ID, the priority and nice value, resident and virtual memory sizes, the start time, the execution time, and the process name with arguments:

```
ps -fly -p {processID}
```

Note that for a process with a large number of arguments, the standard `ps` command displays only a limited set of the arguments based on available space in the terminal window. In that case, the BSD version of the `ps` command (available on Solaris as `/usr/ucb/ps`) can be used to obtain the full command with all arguments, like:

```
/usr/ucb/ps auxwww {processID}
```

pstack

The `pstack` command can be used to obtain a native stack trace of all threads in a process. While a native stack trace might not be as user-friendly as a Java stack trace obtained using `jstack`, it includes threads that are not available in a Java stack trace. For example, the command displays those threads used to perform garbage collection and other housekeeping tasks. The general usage for the `pstack` command is:

```
pstack {processID}
```

dbx / gdb

A process debugger provides the ability to examine a process in detail. Like `pstack`, a debugger can obtain a stack trace for all threads in the process, but it also provides the ability to examine a process (or core file) in much greater detail, including observing the contents of memory at a specified address and the values of CPU registers in different frames of execution. The GNU debugger `gdb` is widely-used on Linux systems and is available on Solaris, but the Sun Studio debugger `dbx` is generally preferred over `gdb` on Solaris.

Note that using a debugger against a live process interrupts that process and suspends its execution until it detaches from the process. In addition, when running against a live process, a debugger has the ability to actually alter the contents of the memory associated with that process, which can have adverse effects. As a result, it is recommended that the use of a process debugger be restricted to core files and only used to examine live processes under the direction of your authorized support provider.

pfiles / lsof

To examine the set of files that a process is using (including special types of files, like sockets) on Solaris, you can use the `pfiles` command, like:

```
pfiles {processID}
```

On Linux systems, the `lsof` tool can be used, like:

```
lsof -p {processID}
```

Tracing Process Execution

If a process is unresponsive but is consuming a nontrivial amount of CPU time, or if a process is consuming significantly more CPU time than is expected, it might be useful to examine the activity of that process in more detail than can be obtained using a point-in-time snapshot like you can get with `pstack` or a debugger. For example, if a process is performing a significant amount of disk reads and/or writes, it can be useful to see which files are being accessed. Similarly, if a process is consistently exiting abnormally, then beginning tracing for that process just before it exits can help provide additional information that cannot be captured in a core file (and if the process is exiting rather than being terminated for an illegal operation, then no core file may be available).

On Solaris systems, the dtrace tool provides an unmatched mechanism for tracing the execution of a process in extremely powerful and flexible ways, but it is also relatively complex and describing its use is beyond the scope of this document. In many cases, however, observing the system calls made by a process can reveal a great deal about what it is doing. This can be accomplished using the truss utility on Solaris or the strace tool on Linux.

The truss utility is very powerful and has a lot of options, but two of the most useful forms in which it may be invoked are:

- **truss -f -p {processID}** – Provides a basic overview of all system calls being made by the specified process (and any subprocesses that it creates) and their associated return values.
- **truss -fear all -p {processID}** – Provides an extremely verbose trace of all system call activity, including details about data being read from or written to files and sockets.

In both cases, the output may be written to a file instead of the terminal window by adding the -o {path} option. Further, rather than observing an already-running process, it is possible to have truss launch the process and trace execution over its entire life span by replacing -p {processID} with name and arguments for the command to invoke.

On Linux systems, the basic equivalent of the first truss variant above is:

```
strace -f -p {processID}
```

Consult the strace manual page for additional information about using it to trace process execution on Linux.

Examining Network Communication

Because the UnboundID® Metrics Engine is a network-based application, it can be valuable to observe the network communication that it has with clients. The Metrics Engine itself can provide details about its interaction with clients by enabling debugging for the protocol or data debug categories, but there may be a number of cases in which it is useful to view information at a much lower level. A network sniffer, like the snoop tool on Solaris or the *tcpdump* tool on Linux, can be used to accomplish this.

There are many options that can be used with these tools, and their corresponding manual pages will provide a more thorough explanation of their use. However, to perform basic tracing to show the full details of the packets received for communication on port 389 with remote host 1.2.3.4, the following commands can be used on Solaris and Linux, respectively:

```
snoop -d {interface} -r -x 0 host 1.2.3.4 port 389
tcpdump -i {interface} -n -XX -s 0 host 1.2.3.4 and port 389
```

On Solaris systems, the snoop command provides enhanced support for parsing LDAP communication (but only when the Metrics Engine is listening on the default port of 389). By adding the "-v" argument to the snoop command line, a verbose breakdown of each packet will be displayed, including protocol-level information. It does not appear that the tcpdump tool provides support for LDAP parsing. However, in either case it is possible to write capture data to a file rather than displaying information on the terminal (using "-o {path}" with snoop, or "-w {path}" with tcpdump), so that information can be later analyzed with a graphical tool like Wireshark, which provides the ability to interpret LDAP communication on any port.

Note that enabling network tracing generally requires privileges that are not available to normal users and therefore may require root access. On Solaris systems, granting the `net_rawaccess` privilege to a user should be sufficient to allow that user to run the `snoop` utility.

Troubleshooting Performance Problems

This section addresses some possible performance issues that the Metrics Engine may experience. The Metrics Engine monitors itself at the same time that 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.

Example of Interpreting Performance Data to Troubleshoot Problems

This section describes troubleshooting system performance problems. It uses a contrived example and answers the question of why an application, which was performing great 30 minutes ago, now exhibits terrible performance.

The help desk receives a phone call at 11:50 AM from a user indicating that application XYZ is performing poorly. The help desk personnel check and all applicable servers appear to be up and running, CPU utilization is within tolerance, and there are no observable network issues. After 30 minutes pass, the help desk staff make no progress with the issue and it gets escalated -- to you.

The application in question is hosted on a pair of proxy servers with both servers sharing the same pair of directory servers in a round-robin configuration.

First, you get a plot of the average proxy server response time that covers the time frame of the initial complaint. This chart 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.

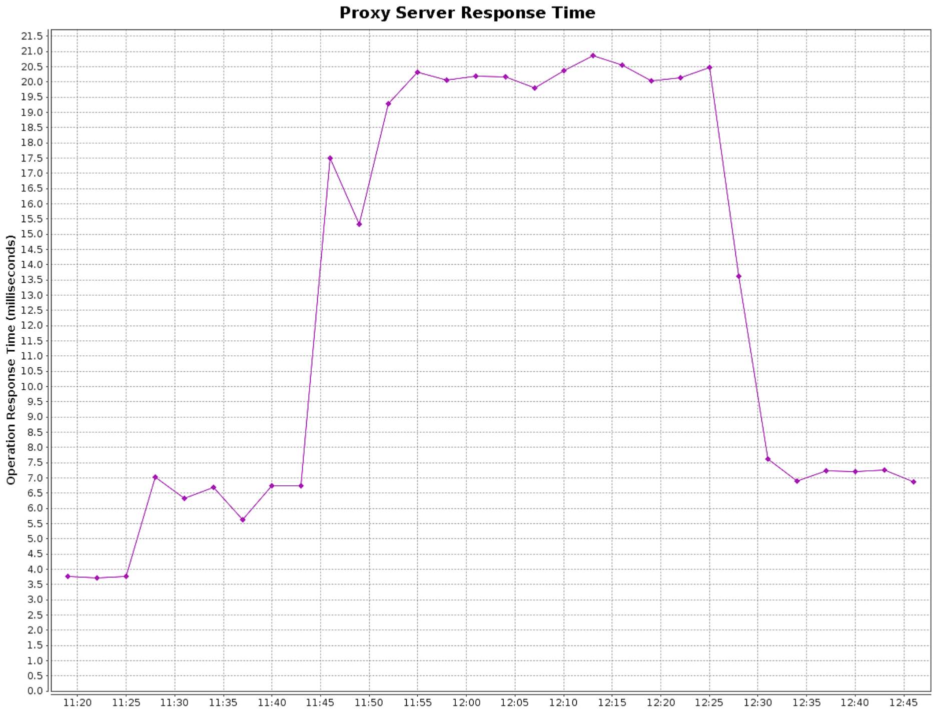


Figure 12: Proxy Server Response Time

This chart shows the problem that the user observed. Application response time tripled right around the time they called in. The average shifted up, meaning that either a few request to a really long time, or maybe everything slowed down. To get more information, you 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.

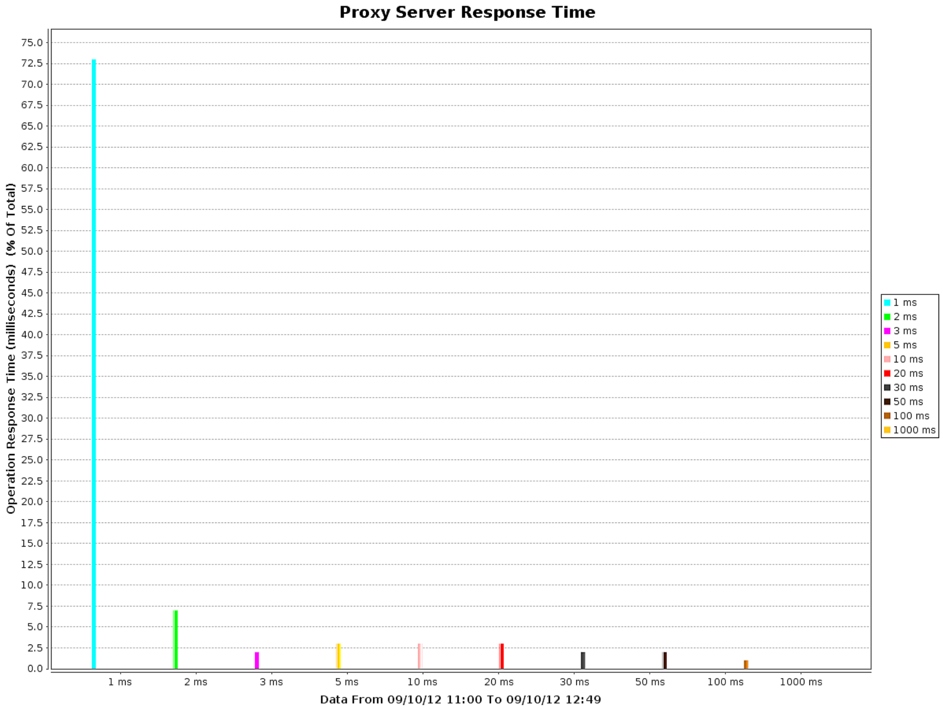


Figure 13: Application Response Time Histogram

The graph shows that no requests during this period took a really long time. So it appears that all operations were slow, so we look at the external server health.

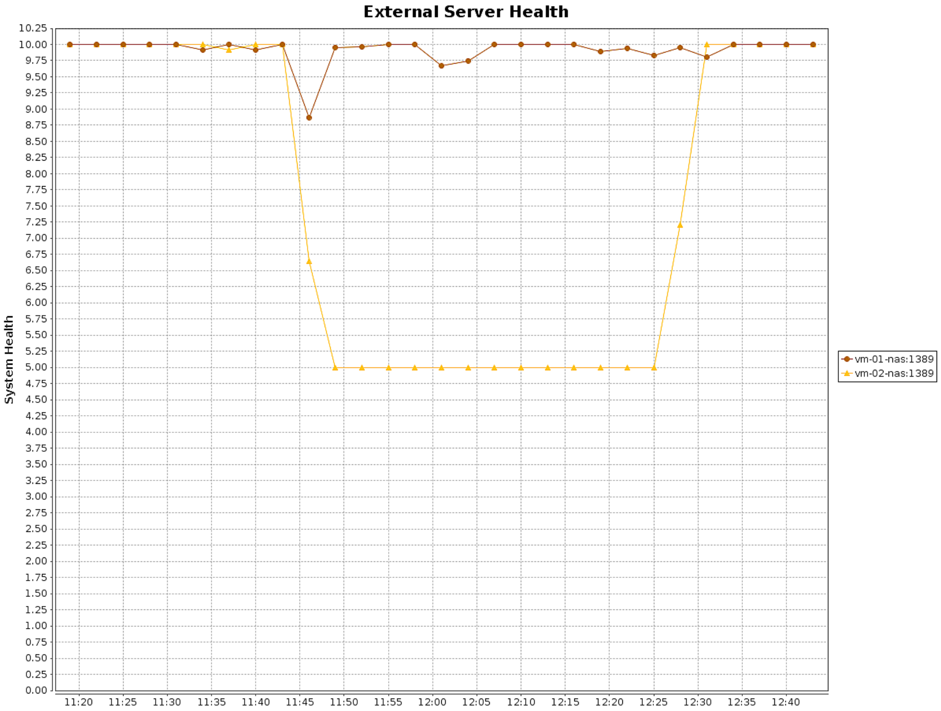


Figure 14: External Server Health

Here, we see an increase in response time that matches the decrease in external server health on vm-02-nas:1389. So, the problem appears to be on that specific Directory Server. Next, we look at what each Directory Server was doing.

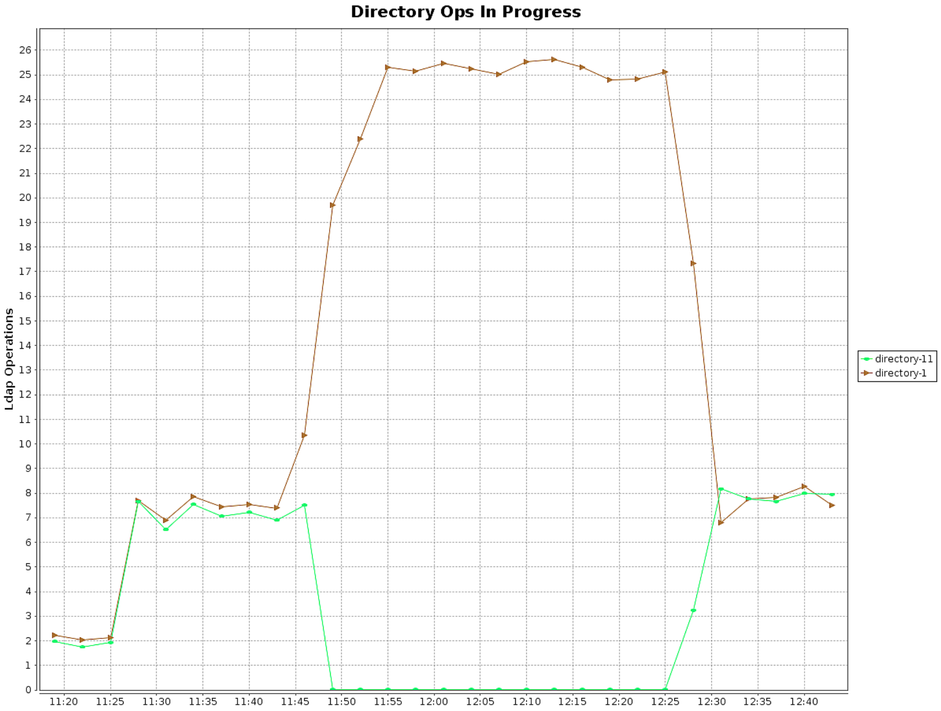


Figure 15: Directory Operations in Progress

During the "bad" period from the first plot, we see that directory-11 (vm-02-nas:1389) stopped doing anything. Something happened on directory-11 and then cleared up about 30 minutes later. Finally, we consult the most recent status on directory-11 using the `status` command. We see the following:

```
--- 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/deploy/ds2/logs' (totalBytes: 18624344064,
usableBytes: 1851559936, usablePercent: 10)
```

Looking at the charts and server status above, you conclude that available disk space on directory-11 went below the warning threshold for a period, resulting in the traffic shifting from two directory servers to only one for about 30 minutes. At the end of that time, both directory servers resumed normal operations and the response time returned to normal. What caused the disk space to suddenly decrease is not known.

All of the charts above were captured using the `query-metric` tool of the Metrics Engine. You can write a script that will capture historical information and use it to quickly analyze performance problems that occurred hours, days, or even weeks ago with a high degree of confidence.

Long Time Before Samples Appear in Queries

The delay between when a metric sample is capture and when it is available in the Metrics Engine is a combination of queuing and polling delays. The default configuration allows the monitored server queue samples in memory for up to 30 seconds before writing them to disk. Samples are not available for the Metrics Engine to capture until after they are written to disk on the monitored servers, so there is a delay of up to 30 seconds in queuing o the monitored server.

The Metrics Engine polls each monitored server every 30 seconds by default, looking for new data. 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.

Sometimes there is a backlog of blocks of sample data to be imported into the Metrics Engine. In this case, a sample block may be delayed by minutes or even hours before becoming available to a query, in part because the import of sample blocks is a sequential operation. Fortunately, you can easily observe this condition and predict when the backlog will be cleared and normal latency can once again be expected.

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. You can 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, and when the Metrics Engine started back up it fetched them and queued them for import. You can see from the chart that about 1500 sample blocks were queued and it took the Metrics Engine about three minutes to catch back up.

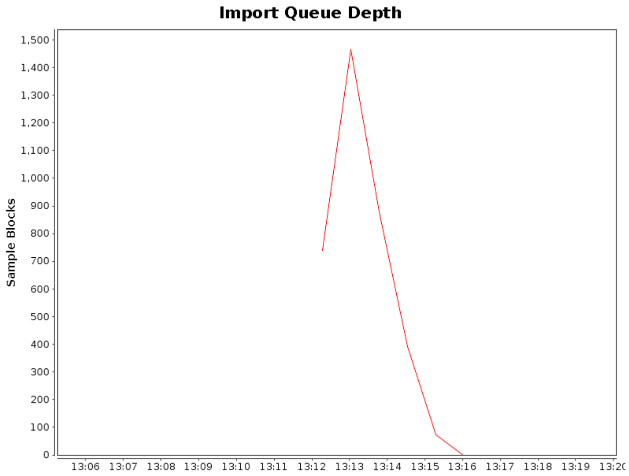


Figure 16: Import Queue Depth

If you choose to monitor over LDAP, the following LDAP entry contains the equivalent information.

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

The Metrics Engine captures and stores all of the data above, so you can easily go back and look at the data's history to judge how well things are working.

Slow Queries for a Particular Metric

The Metrics Engine DBMS is designed for minimal space usage and the data in the samples tables is eventually optimized for query performance. However, there are cases where some queries may be slow. The expected query performance for a metric query should be less than 500 milliseconds per query. If the Metrics Engine host system has adequate CPU and disk performance, as well as enough RAM for both the Metrics Engine server process and PostgreSQL DBMS processes, then 500 milliseconds or less per query is expected for most queries. There are a few exceptions. Understanding the performance considerations can help you improve query response time.

Some metric queries must read millions of records from the DBMS. If the query has not been executed in the recent past, then chances are very good that all of the data will need to come from disk. If the tables holding the data have been layout optimized, this process can take several seconds. If the tables have not been layout optimized, this process can take more than a minute. The Metrics Engine caches recent query results, so making the same query a second time increases the likelihood that it can be much of the data from the sample cache, bypassing the DBMS and reducing the overall query time.

If you have problems with a particular query and want to understand it better, enable the `slow-query-threshold` property using `dsconfig`. Queries for the specified metric that take longer than the threshold will print query statistics to the server's error log. The statistics include the percentage of the query that was already in cache (as a function of time), the number of records read from the DBMS, and how long the DBMS query took.

The following command causes any query for the `throughput` metric that take more than 500 milliseconds to have its query statistics printed to the server error log.

```
dsconfig set-monitoring-configuration-prop \
  --set slow-query-threshold-ms:500:throughput
```

For metrics that you know you want to query, but you cannot predict the frequency of the queries, you can configure metric queries to run in the background. Running them in the background keeps the metrics in the cache and avoids the slow first query. The `slow-query threshold` setting above prints the command you should use to setup a prefetch query when a query exceeds the specified `slow-query-threshold`. See the Prefetched Metric Query configuration object with dsconfig for the details on how to configure this feature.

All Metric Queries are Slow

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. If this attribute is much great than zero, the cache is undersized for the query load placed on the server. You can 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 ten minutes, but this can be increased up to one hour. If you observer the `expired-count` monitor attribute increasing between queries, you may consider increasing the idle timeout as follows:

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

Strange Query Results for Time Ranges Ending Now

The query API attempts by default to aggregate samples across servers and dimension values. Sometimes, the samples for different servers, or even different dimension values, are imported into the Metrics Engine at different times. The only guarantee about the importing f metric samples is that they will be imported in time-order for each server. You can not set the ordering across servers and samples for a specific time may arrive in stages. So, a metric query that aggregates across servers or dimensions may get partial data when the query time range ends close to now. This problem is compounded when the monitored servers have significant clock skew relative to each other, because samples are timestamped with the monitored server clock, not the Metrics Engine clock. Since the query looks at a single time range, the more clock skew the monitored servers have, the higher the probability of the results close to now looking strange.

To illustrate this problem, consider the following example. We want to look at the `throughput` metric for four proxy server. and we want to know how many LDAP operations have occurred every minute of the last hour. The delay problem is most like to appear in the last minute, because the DBMS has throughput data for some of the servers but not all. So, for 59 of the 60 minutes, the throughput data shows all four proxy servers. However, for the last minute, it only has data for two of the four servers. The last sample has only half the value of all the others. While it appears that throughput has dropped dramatically, in fact the last minute sample only contains part of the data, the rest has not year arrived. This problem can occur for metrics that are average-based (like response-time) as well, though it usually is not as dramatic.

If you see this behavior and want to understand it better, the API makes it very easy to investigate. With the PI, you can pivot (split) the data by server and dimension and look at the last few minutes. Since the API lets you format the results as an HTML table, you can see that the data has not arrived and get a better idea of how to formulate your query to avoid this data influx area.

The following sequence of API URLs return the last three minutes of data in 10-second increments:

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

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

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

The first URL aggregate 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 you add dimension pivots (splits), you can see more clearly how the results are aggregations of partial data, a fact that is particularly pronounced in the most recent 60 seconds. The Metrics Engine is especially good at processing large flows of data, but less well suited to low latency reporting.

Note that this behavior is not limited to the most recent 60 seconds. If a server stops reporting metric samples for any reason, this behavior will occur. A server can remain active with LDAP activity but be inaccessible to the Metrics Engine (for example, if the WAN link is disrupted) for a period of time, and the same pattern will be visible in the data. However, this occurs almost all the time within the most recent 30 to 60 seconds.

Optimizing the Layout of the Sample Data Table

The Metrics Engine loads data into the sample data tables in close to chronological order, so that the records in the table are essentially time continguous. Sample data tables are partitioned by time, so that all samples for a given time range are in a single partition. While the insert order does not guarantee the record layout on disk, the records will not be optimized for the supported queries. A sample block may contain samples for 50 different metrics, such that the metric of interest for a query only occurs every 50 records. As a result, the DBMS storage may contain only one record in each DBMS disk page, which is pathological for query performance. The Metrics Engine compensates for this by optimizing the partition when it believe no new records will be added (at the end of the time range the partition supports). This optimization takes up to a minute while it rewrites the entire partition in an order that matches the index order used for metric queries. Query performance for an unoptimized partition is about 80% slower than performance for an optimized partition. This simple background task is critical for good query performance. If a query arrives during partition optimization and needs data from the partition being optimized, the query will be blocked until optimization completes.

Troubleshooting the Metrics Engine API

When making requests of the Metrics Engine API, you may get an HTTP response indicating an internal server error (HTTP code 500). These errors may indicate a problem processing the request that resulted in an exception. If you encounter this error, enable the debug logger for the API resources, as the UnboundID support staff will need the debug log to help diagnose the problem. Enable the debug logger as follows:

```
dsconfig create-debug-target --publisher-name "File-Based Debug Logger" \
   --target-name com.unboundid.directory.mon.api.v1.resources --set debug-level:info \
   --set include-throwable-cause:true
```

Common Problems and Potential Solutions

This section describes a number of different types of problems that can occur and common potential causes for them.

The Server Will Not Run Setup

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

A Suitable Java Environment Is Not Available

The UnboundID® Metrics Engine requires that Java 1.6 (minimum version: Sun/Oracle JDK 1.6.0_31 or JRE 1.6.0 IBM J9 2.4) be installed on the system and made available to the server, and it must be installed prior to running `setup`. If the `setup` tool does not detect that a suitable Java environment is available, it will refuse to run.

To ensure that this does not happen, the `setup` tool should be invoked with an explicitly-defined value for the *JAVA_HOME* environment variable that specifies the path to the Java installation that should be used. For example:

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

If this still does not work for some reason, then it can be that the value specified in the provided *JAVA_HOME* environment variable can be overridden by another environment variable. If that occurs, try the following command, which should override any other environment variables that can be set:

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

Unexpected Arguments Provided to the JVM

If the `setup` script attempts to launch the java command with an invalid set of Java arguments, it might 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, then invoke the following command before trying to re-run `setup`:

```
unset JAVA_ARGS UNBOUNDID_JAVA_ARGS
```

The Server Has Already Been Configured or Used

The `setup` tool is only intended to provide the initial configuration for the Metrics Engine. It refuses to run if it detects that the `setup` tool has already been run, or if an attempt has been made to start the Metrics Engine prior to running the `setup` tool. This protects an existing Metrics Engine installation from being inadvertently updated in a manner that could harm an existing configuration or data set.

If the Metrics Engine has been previously used and if you want to perform a fresh installation, it is recommended that you first remove the existing installation, create a new one and run `setup` in that new installation. However, if you are confident that there is nothing of value in the existing installation (for example, if a previous attempt to run `setup` failed to complete successfully for some reason but it will refuse to run again), the following steps can be used to allow the `setup` program to run:

- 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 below 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. If an instance is already running, then subsequent attempts to start the server will fail. Similarly, some other administrative operations can also 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, and the server still believes that it is running, then it is possible that a previously-held lock was not properly released. In that case, you can try removing all of the files in the `locks` directory before attempting to start the server.

If you wish to have multiple instances running at the same time on the same system, then you should create a completely separate installation in another location on the filesystem.

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 that indicates that the server could not be started with the specified set of arguments. Note that it is possible that an invalid option was provided to the JVM (as described below), but if that same set of JVM arguments has already been used successfully to run the server, then it is more likely that the system does not have enough memory available.

There are a number of potential causes for this:

- If the amount of memory in the underlying system has changed (for example, system memory has been removed, or if the Metrics Engine is running in a zone or other type of virtualized container and a change has been made to the amount of memory that container will be allowed to use), then the Metrics Engine might need to be re-configured to use a smaller amount of memory than had been previously configured.
- Another process running on the system is consuming a significant amount of memory so that there is not enough free memory available to start the server. If this is the case, then

either terminate the other process to make more memory available for the Metrics Engine, or reconfigure the Metrics Engine to reduce the amount of memory that it attempts to use.

- The Metrics Engine was just shut down and an attempt was made to immediately restart it. In some cases, if the server is configured to use a significant amount of memory, then it can take a few seconds for all of the memory that had been in use by the server, when it was previously running, to be released back to the operating system. In that case, run the `vmstat` command and wait until the amount of free memory stops growing before attempting to restart the server.
- For Solaris-based systems only, if the system has one or more ZFS filesystems (even if the Metrics Engine itself is not installed on a ZFS filesystem), but it has not been configured to limit the amount of memory that ZFS can use for caching, then it is possible that ZFS caching is holding onto a significant amount of memory and cannot release it quickly enough when it is needed by the Metrics Engine. In that case, the system should be re-configured to limit the amount of memory that ZFS is allowed to use as described in the Using the Collect Support Data Tool.
- If the system is configured with one or more memory-backed filesystems, for example, `tmpfs` used for `/tmp` for Solaris), then look to see if there are any large files that can be consuming a significant amount of memory in any of those locations. If so, then remove them or relocate them to a disk-based filesystem.
- For Linux systems only, if there is a mismatch between the huge pages setting for the JVM and the huge pages reserved in the operating system. For more information, see Configure Huge Page Support (Linux).

If nothing else works and there is still not enough free memory to allow the JVM to start, then as a last resort, try rebooting the system.

An Invalid Java Environment or JVM Option Was Used

If an attempt to start the Metrics Engine fails with an error message indicating that no valid Java environment could be found, or indicates that the Java environment could not be started with the configured set of options, then you should first ensure that enough memory is available on the system as described above. If there is a sufficient amount of memory available, then other causes for this error can include the following:

- The Java installation that was previously used to run the server no longer exists (for example, an updated Java environment was installed and the old installation was removed). In that case, update the `config/java.properties` file to reference to path to the new Java installation and run the `bin/dsjavaproperties` command to apply that change.
- The Java installation used to run the server has been updated and the server is trying to use the correct Java installation but one or more of the options that had worked with the previous Java version no longer work with the new version. In that case, it is recommended that the server be re-configured to use the previous Java version, so that it can be run while investigating which options should be used with the new installation.
- If an *UNBOUNDID_JAVA_HOME* or *UNBOUNDID_JAVA_BIN* environment variable is set, then its value may override the path to the Java installation used to run the server as 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, then explicitly unset the *UNBOUNDID_JAVA_HOME*, *UNBOUNDID_JAVA_BIN*, and *UNBOUNDID_JAVA_ARGS* environment variables before trying to start the server.

Note that 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, then it can be necessary to 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, like:

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

An Invalid Command-Line Option Was Provided

There are a small number of arguments that are provided when running the `bin/start-ds` command, but in most cases, none are required. If one or more command-line arguments were provided for the `bin/start-ds` command and any of them is not recognized, then the server provides an error message indicating that an argument was not recognized and displays version information. In that case, 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 an officially-supported tool like `dsconfig` or the directory management console, the server should validate that configuration change before applying it. However, it is possible that a configuration change can appear to be valid at the time that it is applied, but does not work as expected when the server is restarted. Alternately, a change in the underlying system can cause a previously-valid configuration to become invalid.

In most cases involving an invalid configuration, the Metrics Engine displays (and writes to the error log) a message that explains the problem, and this can be sufficient to identify the problem and understand what action needs to be taken to correct it. If for some reason the startup failure does not provide enough information to identify the problem with the configuration, then look in the `logs/config-audit.log` file to see what recent configuration changes have been made with the server online, or in the `config/archived-configs` directory to see if there might have been a recent configuration change resulting from a direct change to the configuration file itself that was not made through a supported configuration interface.

If the server does not start as a result of a recent invalid configuration change, then it can be possible to start the server using the configuration that was in place the last time that the server started successfully (for example, the "last known good" configuration). This can be achieved using the `--useLastKnownGoodConfig` option:

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

Note that if it has been a long time since the last time the server was started and a number of configuration changes have been made since that time, then the last known good configuration can be significantly out of date. In such cases, it can be preferable to manually repair the configuration.

If there is no last known good configuration, if the server no longer starts with the last known good configuration, or if the last known good configuration is significantly out of date, then manually update the configuration by editing the `config/config.ldif` file. In that case, you should make sure that the server is offline and that you have made a copy of the existing configuration before beginning. You might wish to discuss the change with your authorized support representative before applying it to ensure that you understand the correct change that needs to be made.

Note: In addition to manually-editing the config file, you can look at previous achived configurations to see if the most recent one works. You can also use the `ldif-diff` tool to compare the configurations in the archive to the current configuration to see what is different.

You Do Not Have Sufficient Permissions

The Metrics Engine should only be started by the user or role used to initially install the server. In most cases, if an attempt is made to start the server as a user or role other than the one used to create the initial configuration, then the server will fail to start, because the user will not have sufficient permissions to access files owned by the other user, such as database and log files. However, if the server was initially installed as a non-root user and then the server is started by the root account, then it can no longer be possible to start the server as a non-root user because new files that are created would be owned by root and could not be written by other users.

If the server was inadvertently started by root when it is intended to be run by a non-root user, or if you wish to change the user account that should be used to run the server, then it should be sufficient to simply change ownership on all files in the Metrics Engine installation, so that they are owned by the user or role under which the server should run. For example, if the Metrics Engine should be run as the "ds" user in the "other" group, then the following command can be used to accomplish this (invoked by the root user):

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

The Server Has Crashed or Shut Itself Down

You can first 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:

- The Metrics Engine was shut down by an administrator. Unless the server was forcefully terminated (for example, using “kill -9”), then messages are written to the `error` and `server.out` logs explaining the reason for the shutdown.
- The Metrics Engine was shut down when the underlying system crashed or was rebooted. If this is the case, then running the `uptime` command on the underlying system shows that it was recently booted.

- The Metrics Engine process was terminated by the underlying operating system for some reason (for example, the out of memory killer on Linux). If this happens, then a message will be written to the system error log.
- The Metrics Engine decided to shut itself down in response to a serious problem that had arisen. At present, this should only occur if the server has detected that the amount of usable disk space has become critically low, or if significant errors have been encountered during processing that left the server without any remaining worker threads to process operations. If this happens, then messages are written to the `error` and `server.out` logs (if disk space is available) to provide the reason for the shutdown.
- The JVM in which the Metrics Engine was running crashed. If this happens, then the JVM should dump a fatal error log (a `hs_err_pid{processID}.log` file) and potentially a core file.

In the event that the operating system itself crashed or terminated the process, then you should work with your operating system vendor to diagnose the underlying problem. If the JVM crashed or the server shut itself down for a reason that is not clear, then contact your authorized support provider for further assistance.

The Server Will Not Accept Client Connections

You can first 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, then potential reasons 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, then connection attempts should time out rather than be rejected.
- If the Metrics Engine is configured to allow secure communication via SSL or StartTLS, then a problem with the key manager and/or trust manager configuration can cause connections to be rejected. If that is the case, then messages should be written to the server access log for each failed connection attempt.
- If the Metrics Engine has been configured with a maximum allowed number of connections, then it can be that the maximum number of allowed client connections are already established. If that is the case, then 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. If this occurs, then a message should be written to the server error log with information about the problem. Another solution is to restart the server. A third option is to restart the connection handler using the LDIF connection handler to make it available again. To do this, 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

You can first 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, then new requests that arrive will be forced to wait in the work queue until a worker thread becomes available. If this is the case, then a stack trace obtained using the `jstack` command shows that all of the worker threads are busy and none of them are waiting for new requests to process.

Note: If all of the worker threads are tied up processing the same operation for a long time, the server will also issue an alert that it might be deadlocked, which may not actually be the case. All threads might be tied up processing unindexed searches.

- If a request handler is stuck performing some expensive processing for a client connection, then other requests sent to the server on connections associated with that request handler is forced to wait until the request handler is able to read data on those connections. If this is the case, then only some of the connections can experience this behavior (unless there is only a single request handler, in which it will impact all connections), and stack traces obtained using the `jstack` command shows that a request handler thread is continuously blocked rather than waiting for new requests to arrive. Note that this scenario is a theoretical problem and one that has not appeared in production.
- If the JVM in which the Metrics Engine is running is not properly configured, then it can be forced to spend a significant length of time performing garbage collection, and in severe cases, could cause significant interruptions in the execution of Java code. In such cases, a stack trace obtained from a `pstack` of the native process should show that most threads are idle but at least one thread performing garbage collection is active. It is also likely that one or a small number of CPUs is 100% busy while all other CPUs are mostly idle. The server will also issue an alert after detecting a long JVM pause (due to garbage collection). The alert will include details of the pause.
- If the JVM in which the Metrics Engine is running has hung for some reason, then 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 network or firewall configuration problem arises, then attempts to communicate with the server cannot be received by the server. In that case, a network sniffer like `snoop` or `tcpdump` should show that packets sent to the system on which the Metrics Engine is running are not receiving TCP acknowledgement.
- If the system on which the Metrics Engine is running has become hung or lost power with a graceful shutdown, then the behavior is often similar to that of a network or firewall configuration problem.

If it appears that the problem is with the Metrics Engine software or the JVM in which it is running, then you need to work with your authorized support provider to fully diagnose the problem and determine the best course of action to correct it.

Problems with the Directory Management Console

If a problem arises when trying to use the directory management console, then potential reasons for the problem may include the following:

- The web application container used to host the console is not running. If an error occurs while trying to start it, then consult the logs for the web application container.
- If a problem occurs while trying to authenticate to the web application container, then make sure that the target Metrics Engine is online. If it is online, then the access log may provide information about the reasons for the authentication failure.
- If a problem occurs while attempting to interact with a Directory Proxy Server instance using the Metrics Engine Management Console, then the access and error logs for that Metrics Engine instance might provide additional information about the underlying problem.

Providing Information for Support Cases

If a problem arises that you are unable to fully diagnose and correct on your own, then contact your authorized support provider for assistance. To ensure that the problem can be addressed as quickly as possible, be sure to provide all of the information that the support personnel may need to fully understand the underlying cause by running the `collect-support-data` tool, and then sending the generated zip file to your authorized support provider. It is good practice to run this tool and send the ZIP file to your authorized support provider before any corrective action has taken place.