



Urika®-GX Analytic Applications Guide

(2.2.UP00)

S-3015

Contents

1 About the Urika®-GX Analytic Applications Guide	4
2 Analytic Software Stack Components.....	6
3 Urika-GX Service Modes.....	7
4 Access Urika-GX Applications.....	10
4.1 Disable Framing on Urika Applications Interface (UAI).....	11
5 Authentication Mechanisms.....	13
6 Apache Hadoop Support.....	15
6.1 Load Data into the Hadoop Distributed File System (HDFS).....	16
6.2 Run a Simple Hadoop Job.....	17
6.3 Run a Simple Word Count Application Using Hadoop.....	18
6.4 Monitor Hadoop Applications.....	19
6.5 Use Tiered Storage on Urika-GX.....	20
6.6 Assign the HDFS/ptmp Directory to Use SSDs for Block Storage.....	22
6.7 Change the Default HDFS Storage Policy.....	22
7 Apache Spark Support.....	24
7.1 Monitor Spark Applications.....	26
7.2 Remove Temporary Spark Files from SSDs.....	28
7.3 Obtain Additional Temporary Space for Running Spark Jobs.....	28
7.4 Enable Anaconda Python and the Conda Environment Manager.....	29
7.5 Provide Kerberos Credentials to Spark.....	30
7.6 Redirect a Spark Job to a Specific Directory.....	31
7.7 Modify the Default Number of Maximum Spark Cores.....	31
7.8 Execute Spark Jobs on Kubernetes.....	33
7.9 Multi-tenant Spark Thrift Server on Urika-GX.....	35
8 Use Apache Mesos on Urika-GX	38
8.1 Access the Apache Mesos Web UI.....	40
8.2 Use mrun to Retrieve Information About Marathon and Mesos Frameworks.....	40
8.3 Clean Up Residual mrun Jobs.....	44
8.4 Launch an HPC Job Using mrun.....	45
8.5 Manage Resources on Urika-GX.....	46
8.6 Manage Long Running Services Using Marathon.....	48
8.7 Flex up a YARN sub-cluster on Urika-GX.....	51
9 Access the Jupyter Notebook UI.....	53
9.1 Create a Jupyter Notebook.....	54
9.2 Share or Upload a Jupyter Notebook.....	56

9.3 Create a Custom Python Based Kernel for JupyterHub.....	59
10 Get Started with Using Grafana.....	60
10.1 Urika-GX Performance Analysis Tools.....	62
10.2 Update the InfluxDB Data Retention Policy.....	62
11 Use Docker on Urika-GX.....	64
11.1 Image Management with Docker and Kubernetes.....	65
11.2 Run the Native Docker Engine on Marathon.....	66
12 Start Individual Kafka Brokers.....	68
13 Overview of the Cray Application Management UI.....	69
14 Update the InfluxDB Data Retention Policy.....	71
15 Manage the Spark Thrift Server as a Non-Admin User.....	73
16 Use Tableau® with Urika-GX.....	74
16.1 Connect Tableau to HiveServer2 Using LDAP.....	74
16.2 Connect Tableau to HiveServer2 Securely.....	78
16.3 Connect Tableau to the Spark Thrift Server	81
16.4 Connect Tableau to the Spark Thrift Server Securely.....	85
16.5 Connect Tableau to Apache Spark Thrift Server on a VM.....	89
16.6 Enable SSL for Spark Thrift Server of a Tenant.....	92
17 File Systems.....	94
18 Check the Current Service Mode.....	95
19 Fault Tolerance on Urika-GX.....	96
20 Default Urika-GX Configurations.....	97
20.1 Default Grafana Dashboards.....	99
20.2 Performance Metrics Collected on Urika-GX.....	110
20.3 Default Log Settings.....	114
20.4 Tunable Hadoop and Spark Configuration Parameters.....	115
20.5 Node Types.....	117
20.6 Service to Node Mapping.....	118
20.7 Port Assignments.....	121
20.8 Major Software Components Versions.....	124
21 Troubleshooting.....	126
21.1 Diagnose and Troubleshoot Orphaned Mesos Tasks.....	126
21.2 Analytic Applications Log File Locations.....	127
21.3 Clean Up Log Data.....	129
21.4 Ensure Long Running Spark Jobs Finish Executing.....	130
21.5 Troubleshoot Common Analytic and System Management Issues	130

1 About the Urika®-GX Analytic Applications Guide

This publication describes the analytic components, workload management and performance analysis tools of the Cray® Urika®-GX system.

Typographic Conventions

<code>Monospace</code>	Indicates program code, reserved words, library functions, command-line prompts, screen output, file/path names, key strokes (e.g., <code>Enter</code> and <code>Alt-Ctrl-F</code>), and other software constructs.
Monospaced Bold	Indicates commands that must be entered on a command line or in response to an interactive prompt.
<i>Oblique or Italics</i>	Indicates user-supplied values in commands or syntax definitions.
Proportional Bold	Indicates a graphical user interface window or element.
<code>\</code> (backslash)	At the end of a command line, indicates the Linux® shell line continuation character (lines joined by a backslash are parsed as a single line). Do not type anything after the backslash or the continuation feature will not work correctly.

Scope and Audience

The audience of this publication are users and system administrators of the Urika®-GX system. This publication is not intended to provide detailed information about open source and third party tools installed on the system. References to online documentation are included wherever applicable.

Record of Revision

Date	Release
September, 2018	2.2UP00
May, 2018	2.1UP00
December, 2017	2.0UP00
April, 2017	1.2UP00
December, 2016	1.1.UP00
August, 2016	1.0.UP00
March, 2016	0.5.UP00

- **New Information:**

Information about multitenant Spark.

- **Updated Content:**

- Updates to Tableau related information.

Trademarks

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, Urika-GX, Urika-XA, Urika-GD, and YARCDATA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYDOC, CRAYPAT, CRAYPORT, DATAWARP, ECOPHLEX, LIBSCI, NODEKARE. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

2 Analytic Software Stack Components

Cray Urika-GX is a high performance big data analytics platform, which is optimized for multiple workflows. It combines a highly advanced hardware platform with a comprehensive analytics software stack to help derive optimal business value from data. The system's analytics platform provides an optimized set of tools for capturing and organizing a wide variety of data types from different sources and executing a variety of analytic jobs.

Major components of the analytic software stack include:

- **Apache Hadoop** - Hadoop is a software framework that provides the means for distributed storing and processing of large data sets. In addition to the core Hadoop components, Urika-GX ships with a number of Hadoop ecosystem components for increased productivity.
- **Apache Spark** - Spark is a general data processing framework that simplifies developing fast, end-to-end big data applications. It provides the means for executing batch, streaming, and interactive analytics jobs. In addition to the core Spark components, Urika-GX ships with a number of Spark ecosystem components.
- **Cray Graph Engine (CGE)** - CGE is a highly optimized and scalable graph analytics application software, which is designed for high-speed processing of interconnected data. It features an advanced platform for searching very large, graph-oriented databases and querying for complex relationships between data items in the database.

3 Urika-GX Service Modes

Urika-GX features application and data security for a number of applications. Many security mechanisms play a role in the overall security architecture to ensure the system is protected against unauthorized access.

- **Default Mode** - All the installed analytic applications are available under the default service mode. Applications are configured with basic security levels wherever possible, though certain applications may provide no security features. More specifically, HDFS runs using its simple authentication mode under the default service mode.
- **Secure Mode** - Urika-GX uses Kerberos authentication while running under the secure mode. Only a limited number of analytic applications are available under this mode and those applications are configured to run in a secure configuration, the exact details of which vary by application. Typically, this means that applications that interact with HDFS require valid Kerberos credentials. Moreover, user interfaces are either disabled or require authentication under this mode.



CAUTION: If the Urika GX system is running in the secure mode, Cray does not recommend toggling back to the default mode while in production. In the default service mode, the security assurances provided by secure service mode are not in place and the security of data that was protected by secure mode may be compromised while running in the default mode. Cray cannot extend the secure mode security assurances to any system that has run in a production state in the default mode until that system has been fully re-deployed.

Table 1. List of Services Available Under the Default and Secure Service Modes

Service	Available in Default Service Mode	Available in Secure Service Mode
Cray Programming Environment	Yes	No
SELinux	Yes	No
Analytic Applications and Resource Management Tools		
ZooKeeper	Yes	Yes
Spark	Yes	Yes
Mesos Master	Yes	No
Mesos Slave	Yes	No
Kubernetes	No	Yes
HDFS NameNode	Yes	Yes
HDFS Secondary NameNode	Yes	Yes
HDFS DataNode	Yes	Yes
CGE	Yes	No

Spark History Server	Yes	No
Spark Thrift Server	Yes	No
YARN	Yes	No
YARN Resource Manager	Yes	No
YARN Node Managers	Yes	No
Hadoop Job History Server	Yes	No
Hadoop Application Timeline Server	Yes	No
Hive MetaStore	Yes	No
HiveServer2	Yes	No
Hive WebHCat	Yes	No
HUE	Yes	No
Oozie Server	Yes	No
Marathon	Yes	No
Grafana	Yes	No
InfluxDB	Yes	No
JupyterHub	Yes	No
System Management Tools		
HSS	Yes	Yes
Nagios	Yes	Yes
Cobbler	Yes	Yes
Secret Manager	Yes	Yes
Tenant management and proxy tools	Yes	Yes
Analytic programming environment components <ul style="list-style-type: none"> • R • Numpy • Scipy • GIT • Environment modules • glibc-devel • gcc • Python 34 • Python 27 • Scala • Apache Maven 	Yes	Yes

• Anaconda Python		
Miscellaneous Tools and UIs		
YAM	Yes	No
mrunc	Yes	No
Urika Application Management (UAM)	Yes	No
Urika-GX Application Interface (UAI)	Yes	No
HAProxy	Yes	Yes
Connectivity to Tableau	Yes	No
Docker	No	Yes

Any additional services installed on the system will use their own security mechanisms and will not be affected by Urika-GX's default and secure modes.

Table 2. Relationship Between Access Levels and Service Modes

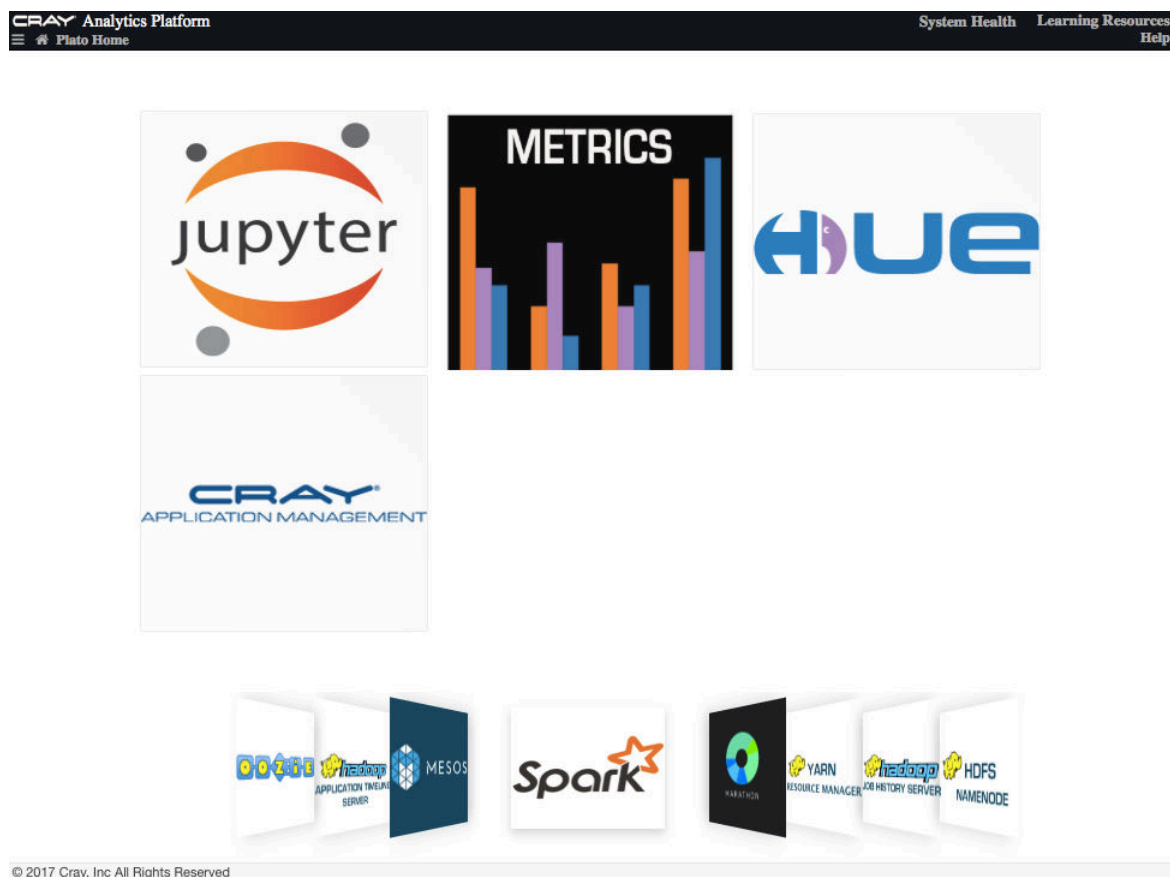
Mode	Restricted Access		Relaxed Access	
	As Member on Tenant	On Physical Node	As Member on Tenant	On Physical Node
Secure	Has proxied access to Spark and HDFS commands	No access	Has proxied access to Spark and HDFS commands	Has direct access to all the services supported in the secure mode
Default	Unsupported	Unsupported	Unsupported	Has direct access to all the services supported in the default mode

4 Access Urika-GX Applications

Access Urika-GX applications using:

- **Urika-GX Applications Interface (UAI).** UAI is the primary entry point to a number of web applications running on Urika-GX. Access this UI by pointing a browser at `http://hostname-login1`. This UI can also be used for viewing system health information and accessing learning resources, such as pre-installed Jupyter notebooks and Urika-GX PDF publications.

Figure 1. Urika-GX Applications Interface



- HTTP ports numbers that the applications are configured to run on. Use the ports configured in the HAProxy file if SSL is enabled.

TIP: If applications are accessed using ports/URLs, the UI of the accessed application will not display the top banner, which contains additional links, including **System Health**, **Learning Resources** and **Help**. Therefore, it is recommended to use the **Urika-GX Applications Interface** to access these applications.

Table 3. Accessing User Interfaces Using URLs and Ports (when SSL is disabled)

Application	URL(s)
Urika-GX Applications Interface	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:80</code> ○ <code>http://hostname-login1</code>
YARN Resource Manager	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:8088</code> ○ <code>http://hostname-login2:8088</code>
HUE	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:8888</code> ○ <code>http://hostname-login2:8888</code>
Oozie Server	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:11000/oozie</code> ○ <code>http://hostname-login2:11000/oozie</code>
Hadoop Job History Server	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:19888</code> ○ <code>http://hostname-login2:19888</code>
Hadoop Application Timeline Server	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:8188</code> ○ <code>http://hostname-login2:8188</code>
Marathon	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:8080</code> ○ <code>http://hostname-login2:8080</code>
Mesos Master	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:5050</code> ○ <code>http://hostname-login2:5050</code>
Spark History Server	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:18080</code> ○ <code>http://hostname-login2:18080</code>
Grafana	<code>http://hostname-login2:3000</code>
HDFS NameNode	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:50070</code> ○ <code>http://hostname-login2:50070</code>
HDFS Secondary NameNode	<ul style="list-style-type: none"> ○ <code>http://hostname-login1:50090</code> ○ <code>http://hostname-login2:50090</code>
Jupyter Notebook	<code>http://hostname-login1:7800</code>
Urika Application Management	<code>http://hostname-login1/applications</code>

4.1 Disable Framing on Urika Applications Interface (UAI)

Prerequisites

This procedure requires root privileges.

About this task

The default behaviour of UAI is to display the user interfaces for applications inside of a frame, which allows for providing a consistent navigation experience across the user interface. However, for some sites this may not be desirable. For example, in the case where SSL has been enabled, some applications may not display correctly due to browser security policies. It is possible to override this behaviour by changing the values of the `FRAME_APPLICATIONS` and `FRAME_SECURE_APPLICATIONS` variables found in the `/opt/cray/ui-application-management/default/application_management/settings.py` file.

Procedure

1. Log on to login node 1 as root.
2. Switch to the `/opt/cray/ui-application-management/default/application_management` directory.
3. Edit the `settings.py` file and set the values of the `FRAME_APPLICATIONS` and `FRAME_SECURE_APPLICATIONS` variables according to the desired behaviour.
 - If the first variable is set to `true`, all applications will always be framed, regardless of the setting of the second variable.
 - If the first variable is set to `false` but the second variable is set to `true`, only the applications for which SSL is enabled will be framed, whereas all the other applications will be displayed without frames.
 - If both the variables are set to `false`, all the applications will be displayed without frames.
4. Save and quit the `settings.py` file.
5. Restart the `httpd` service.

```
# service httpd restart
```

5 Authentication Mechanisms

Table 4. Authentication Mechanisms

Application	Authentication Mechanism
Cray Application Management UI	LDAP. Users can also log in with the default account shipped with the system. This account has the following credentials: username: admin password: admin
Urika-GX Applications Interface	Not available.
Documentation and Learning Resources UI	Not available.
Grafana UI	LDAP. The system also ships with a default account that can be used to log on to Grafana. The credentials of this account are: username: admin password: admin
Jupyter Notebook UI	LDAP. The system also ships with a default account that can be used to log on to Jupyter. The credentials of this account are: username: crayadm password: initial0
HUE UI	LDAP. The system also ships with a default admin account that can be used to log on to HUE. The credentials of this account are: username: admin password: initial0
Hadoop/Spark related UIs, such as Spark History Server, Hadoop History Server, YARN Resource Manager, etc.	Not available
Marathon UI	Not available.
Mesos UI	User name: LDAP user name

Application	Authentication Mechanism
	Password: Mesos secret, found in the <code>/security/secrets/userName.mesos</code> file, located on all the nodes.
Spark Thrift Server	Urika-GX ships with LDAP authentication enabled for Spark Thrift server. SSL authentication can be set up if required. For instructions, refer to <i>Urika®-GX System Administration Guide</i> . Storage based authorization is supported for Spark Thrift Server.
HiveServer2	LDAP authentication for HiveServer2 can be enabled if required. Storage based and SQL standard based authorizations are supported for HiveServer2. For more information, refer to <i>Urika®-GX System Administration Guide</i>
Nagios UI	Default credentials: <ul style="list-style-type: none">• User name: <code>crayadm</code>• Password: <code>initial0</code>

6 Apache Hadoop Support

Urika-GX ships with Hortonworks® Data Platform (HDP), which is a distribution package comprising of a number of Apache Hadoop projects. In addition to the core Hadoop elements, the following Hadoop ecosystem components are pre-installed and pre-configured on Urika-GX:

- **Apache™ Avro™** - Data serialization system.
- **Apache™ DataFu™** - Collection of libraries for working with large-scale data in Hadoop.
- **Apache™ Flume™** - Distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.
- **Apache™ Hive™** - Data warehouse framework that facilitates querying and management of large datasets residing in a distributed store/file system like the Hadoop Distributed File System (HDFS). Subcomponents of Apache Hive include:
 - **Apache™ HCatalog™** - Table and storage management layer for Hadoop that enables users with different data processing tools to more easily read and write data on the grid.
 - **Apache™ WebHCat™** - REST API for HCatalog, which acts as the storage management layer for Hadoop.
- **Apache™ Hue™** - Set of web applications that enable interacting with a Hadoop cluster and browsing HDFS.
- **Apache™ Kafka™** - Publish-subscribe messaging service.
- **Apache™ Mahout™** - Scalable machine learning and data mining library.
- **Apache™ Oozie™** - Job work-flow scheduling and coordination manager for managing the jobs executed on Hadoop.
- **Apache™ Parquet™** - Columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.
- **Apache™ Pig™** - Software framework which offers a run-time environment for execution of MapReduce jobs on a Hadoop cluster via a high-level scripting language called *Pig Latin*.
- **Apache™ Sqoop™** - Apache Sqoop is a tool designed for efficiently transferring the data between Hadoop and Relational Databases (RDBMS).
- **Apache™ HiveServer2™/Hive™ Thrift Server** - Apache HiveServer2 is a server interface that enables remote clients to execute queries against Hive and retrieve the results. The current implementation, based on Thrift RPC, is an improved version of HiveServer and supports multi-client concurrency and authentication. It is designed to provide better support for open API clients like JDBC and ODBC.

HiveServer2 and Spark Thrift server are used on Urika-GX for enabling connections from ODBC/JDBC clients, such as Tableau.
- **Apache™ ZooKeeper™** - Centralized coordination service that is responsible for maintaining configuration information, offering coordination in a distributed fashion, and a host of other capabilities.

None of the Urika-GX components are currently configured to send data to the Hadoop Application Timeline Server. Users must configure their own applications to send data to the Hadoop Application Timeline Server.

For more information, visit <http://hortonworks.com/> and <http://www.apache.org>

HDFS Data Storage

HDFS data is stored on the SSDs and HDDs of Urika-GX's compute nodes, which results in faster data transfer and lower latency.

Designated HDFS Data Nodes:

- 3 sub-rack/48 node system - `nid[00001-00015]`, `nid[00017-00029]`, and `nid[00033-00045]`
- 2 sub-rack/32 node system - `nid[00001-00007]`, `00009-00013`, `00017-00029]`
- Single sub-rack/16 node system - `nid[00001-00007]`, `[00009-00013]`, `[00017-00029]`

Executing Hadoop Commands on Urika-GX

Flex up the YARN sub-cluster using the `urika-yam-flexup` command before executing Hadoop commands. For more information, see the `urika-yam-flexup` man page.

6.1 Load Data into the Hadoop Distributed File System (HDFS)

Prerequisites

This procedure requires the HDFS service to be up and running.

About this task

The first step in using Hadoop is loading data into HDFS, which provides storage across the cluster nodes. The following set of steps retrieve a copies the content of books by Mark Twain and James Fenimore Cooper into files and then copies those files into HDFS.

Procedure

1. Log on to a login node.
2. Use the `wget` Linux utility to download content from Twain and Cooper's books.

`wget` is a computer program that retrieves content from web servers, to download the Twain and Cooper's works.

```
$ wget -U firefox http://www.gutenberg.org/cache/epub/76/pg76.txt
$ wget -U firefox http://www.gutenberg.org/cache/epub/3285/pg3285.txt
```

3. Rename the newly created files to names of choice.

```
$ mv pg3285.txt DS.txt
$ mv pg76.txt HF.txt
```

4. Load the content of both Twain and Cooper's books into the HDFS file system.



CAUTION: The following command will fail if the `/user/userID` directory does not exist. This is because users do not have write access to HDFS unless an administrator provides them a designated folder under `hdfs:///user`.


```
$ hdfs dfs -mkdir /user/userID
```

5. Move the DS.txt and HF.txt files to HDFS.

```
$ hdfs dfs -put HF.txt /user/userID
$ hdfs dfs -put DS.txt /user/userID
```

6. Load the article and compress the text file using the gzip utility.

```
$ gzip DS.txt
$ gzip HF.txt
```

7. Execute the hdfs dfs command.

```
$ hdfs dfs -put DS.txt.gz /user/userID
$ hdfs dfs -put HF.txt.gz /user/userID
```

8. Verify that the files have been loaded into HDFS.

```
$ hdfs dfs -ls /user/userID
Found 2 items
-rw-r--r-- 1 crayadm supergroup 459386 2012-08-08 19:34 /user/crayadm/DS.txt.gz
-rw-r--r-- 1 crayadm supergroup 597587 2012-08-08 19:35 /user/crayadm/HF.txt
```

6.2 Run a Simple Hadoop Job

About this task

The instructions documented in this procedure can be used to run a TeraSort benchmark, which is a simple Hadoop job. TeraSort uses the Regular MapReduce sorting, except for a custom partitioner that splits the mapper output into N-1 sample keys to ensure that each of the N reducer receives records with key K, such that $\text{sample}[i-1] \leq K < \text{sample}[i]$, where i is the reducer instance number.

This procedure can be considered as running an equivalent of a "Hello World" program. The goal of the TeraSort benchmark is to sort data as fast as possible. In the following instructions, data is generated using TeraGen, which is a MapReduce program for generating data. Furthermore, the results are validated via TeraValidate, which is a MapReduce program for validating that the output is sorted.



CAUTION: Please be aware that this example will fail on a 16 node Urika-GX as 16 node machines only have 9 HDFS data nodes and TeraSort expects to make 10 replications. Those who wish to still run TeraSort on a 16 node Urika-GX can increase the value of `dfs.replication.max` in `hdfs-site.xml`, but be aware there may be adverse effects of attempting to create more replications than there are HDFS datanodes in the system.

Procedure

1. Log on to a login node.
2. Flex up a YARN sub-cluster using the `urika-yam-flexup` command. For more information on using this command, see the `urika-yam-flexup` man page.



CAUTION: If YARN node managers are not flexed up when a Hadoop job is started, the Hadoop job will hang indefinitely until resources are provided by flexing up YARN node managers.

3. Generate the input data via TeraGen, using default MapReduce options.

In the following example, the command `hdfs dfs -rm -R /tmp/10gsort` is only needed if the `tmp/10gsort` directory already exists.

```
$ hdfs dfs -rm -R /tmp/10gsort
$ yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar teragen 100 /tmp/10gsort/input
```

4. Execute the TeraSort benchmark on the input data.

```
$ yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar terasort /tmp/10gsort/input /tmp/10gsort/output
```

5. Validate the sorted output data via TeraValidate.

```
$ yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar teravalidate /tmp/10gsort/output /tmp/10gsort/validate
```

6. Verify the success of the validation.

```
$ hdfs dfs -ls /tmp/10gsort/validate

Found 2 items
-rw-r--r--  3 user hdfs          0 2015-08-12 17:15
/tmp/10gsort/validate/_SUCCESS
-rw-r--r--  3 user hdfs        20 2015-08-12 17:15
/tmp/10gsort/validate/part-r-00000
```

6.3 Run a Simple Word Count Application Using Hadoop

About this task

The following code shows how to invoke the word counter program, which is included in the Hadoop example JAR file.

Procedure

1. Flex up a YARN sub-cluster using the `urika-yam-flexup` command. For more information on using this command, see the `urika-yam-flexup` man page.



CAUTION: If YARN Node Managers are not flexed up when a Hadoop job is started, the Hadoop job will hang indefinitely until resources are provided by flexing up YARN nodes

2. Remove the output directory if already exists.

```
$ hdfs dfs -rm /tmp/word_out
```

3. Execute the following command:

```
$ yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount \
/tmp/word_in /tmp/word_out
```

Where `/tmp/word_in` is the directory containing files whose words are to be counted and `/tmp/word_out` is the output directory

4. Verify the results.

```
$ hdfs dfs -cat /tmp/word_out/part*
```

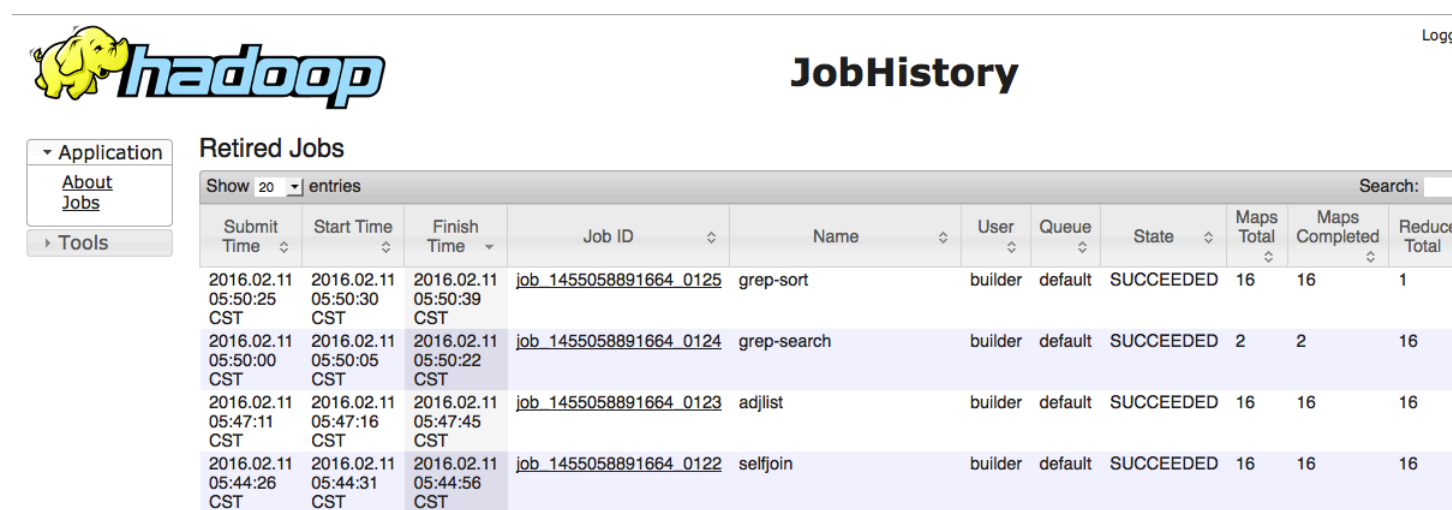
- Flex down the YARN sub-cluster using the `urika-yam-flexdown` command. For more information on using this command, see the `urika-yam-flexdown` man page.

6.4 Monitor Hadoop Applications

Hadoop jobs can be monitored using the following tools:

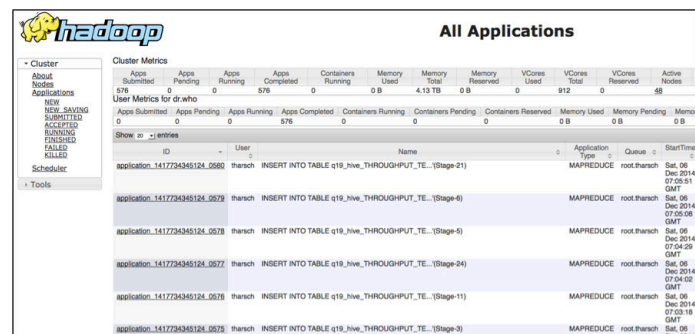
- Hadoop Job History Server UI**, which can be accessed via the **Urika-GX Applications Interface** UI or at: `http://hostname-login1:19888` and `http://hostname-login2:19888`. Accessing via the **Urika-GX Applications Interface** is recommended.

Figure 2. Hadoop Job History Server UI



- YARN Resource Manager UI**, which can be accessed via the **Urika-GX Applications Interface** or at `http://hostname-login1:8088` and `http://hostname-login2:8088`. Accessing via the **Urika-GX Applications Interface** is recommended.

Figure 3. YARN Resource Manager UI



- Cray Application Management UI**, which can be accessed via **Urika-GX Applications Interface** or at `http://hostname-login1/applications`

Figure 4. Cray Application Management UI

The screenshot shows the Cray Analytics Platform Application Management UI. At the top, there's a navigation bar with 'CRAY Analytics Platform' and 'System Health' links. Below it, the 'APPLICATION MANAGEMENT' section is active. The main content area is titled 'Job List' and features a table of jobs. A 'Filter by Type' modal is open, showing checkboxes for SPARK, YARN, CGE, MRUN, and OTHER. The table has columns for Job Id, Metrics, Job Name, Start Time, and Status. The first three rows of the table are visible, showing jobs with various IDs and names, all with a status of 'FINISHED'. The bottom of the table shows pagination: '1 to 3 of 3 (filtered from 35 total entries)' and a page number '25'.

6.5 Use Tiered Storage on Urika-GX

Storage policies define the policy HDFS uses to persist block replicas of a file to storage types as well as the desired storage type(s) for each replica of the file blocks being persisted. They allow for fallback strategies, whereby if the desired storage type is out of space then a fallback storage type is utilized to store the file blocks. The scope of these policies extends and applies to directories, and all files within them.

Storage policies can be enforced during file creation or at any point during the lifetime of the file. For storage policies that change during the lifetime of the file, HDFS introduces a new tool called Mover that can be run periodically to migrate all files across the cluster to correct storage types based on their storage policies.

Urika-GX implements the Hadoop 2.7.3 HDFS tiered-storage, combining both the SSDs and hard drives into a single storage paradigm. The HDFS NameNode considers each DataNode to be a single storage unit with uniform characteristics. By adding awareness of storage media, HDFS can make better decisions about the placement of block data with input from applications. An application can choose the distribution of replicas based on its performance and durability requirements. The storage policy will be DISK by default on Urika-GX if no storage policy is assigned. Changes to this default value can be made based on requirements and site policies.

Storage Types and Storage Policies

Each DataNode in HDFS is configured with a set of specific disk volumes as storage mount points, on which HDFS files persist.



CAUTION: It is recommended not to modify how volumes are tagged in `hdfs-site.xml`.

With Urika-GX, users can tag each volume with a storage type to identify the characteristic of storage media that represents the volume. For example, a mounted volume may be designated as an archival storage and another one as flash storage.

An example of the `hdfs-site.xml` file is shown below:

```
<property>
  <name>dfs.datanode.data.dir</name>
  <value>[SSD]file:///mnt/ssd/hdfs/dd,[DISK]file:///mnt/hdd-2/hdfs/dd</value>
</property>
```

The available space reported by HDFS commands represents the space consumed by HDDs and SSDs. Even though Urika-GX has a heterogeneous file system, the default storage type is `DISK`, unless explicitly set to use SSD.

TIP: If the system indicates that the disks have reached full capacity and HDFS commands indicate that there is still some space available, find out the actual space consumed by the storage type by adding up the available space on the individual directories on each node.

- For HDD = `/mnt/hdd-2/hdfs/dd`
- For SSD = `/mnt/ssd/hdfs/dd`

Available Policies

Urika-GX ships with the following pre-defined HDFS policies, which can be assigned to different HDFS directories and files.

```
$ hdfs storagepolicies -listPolicies
Block Storage Policies:
BlockStoragePolicy{HOT:7, storageTypes=[DISK], creationFallbacks=[], replicationFallbacks=[ARCHIVE]}
BlockStoragePolicy{ONE_SSD:10, storageTypes=[SSD, DISK], creationFallbacks=[SSD, DISK], replicationFallbacks=[SSD, DISK]}
BlockStoragePolicy{ALL_SSD:12, storageTypes=[SSD], creationFallbacks=[DISK], replicationFallbacks=[DISK]}
BlockStoragePolicy{LAZY_PERSIST:15, storageTypes=[RAM_DISK, DISK], creationFallbacks=[DISK], replicationFallbacks=[DISK]}
```

`ONE_SSD` attempts to place one replica of each block on SSD, whereas `ALL_SSD` attempts to place all replicas.

Warm and cold storage policies are not supported on Urika-GX.

Use Cases

1. An application creates a file and requests that block replicas be stored on a particular Storage Type. Variations include:
 - a. ALL replicas on the same storage type.
 - b. Replicas on different storage types, for example, two replicas on HDD, one on SSD, etc.
 - c. Application requests that the `Storage Type` setting is mandatory. For example, the operator places hot files, such as the latest partitions of a table on SSDs
2. An application changes the storage media of a file. Variations include:
 - a. For ALL replicas
 - b. For some of the replicas
 - c. Application requests that the new setting is mandatory.
3. The user creates quota for a particular storage media type at a directory.
4. Upon request, the user can move hot data to faster storage media based on access patterns.

Performance Considerations

- The impact of storage policies is minimal for smaller datasets, due to the presence of the OS buffer cache, which caches files in memory.

- For the TeraSort test most of the benefit of the SSDs can be obtained with the `ONE_SSD` policy, since we only need to read one replica of each block.

6.6 Assign the HDFS/ptmp Directory to Use SSDs for Block Storage

About this task

Follow this procedure to use SSDs as the block storage for the `HDFS/ptmp` directory.

Procedure

1. Create the `/ptmp` directory.

```
$ hdfs dfs -mkdir /ptmp
```

2. Change security settings of the `/ptmp` directory to 1777

```
$ hdfs dfs -chmod 1777 /ptmp
```

3. Set the storage policy of the `/ptmp` directory.

```
$ hdfs storagepolicies -setStoragePolicy -path /ptmp -policy ALL_SSD
Set storage policy ALL_SSD on /ptmp
```

4. Verify that the storage policy has been changed.

```
$ hdfs storagepolicies -getStoragePolicy -path /ptmp
The storage policy of /ptmp:
BlockStoragePolicy{ALL_SSD:12, storageTypes=[SSD], creationFallbacks=[DISK],
replicationFallbacks=[DISK]}
```

6.7 Change the Default HDFS Storage Policy

About this task

HDFS is configured to utilize both `DISK` and `SSD` in the `hdfs-site.xml` file. The default storage policy is `HOT`, therefore only `DISK` is used on Urika-GX. It should be noted that the HDFS UI reports the combined (`DISK` and `SSD`) space available to HDFS.

By default, Urika-GX ships pre-configured with an `/ALL_SSD/HDFS` directory and a `/ONE_SSD/HDFS` directory, with the associated storage policies. This procedure describes how to use SSDs for HDFS storage.

Procedure

1. Log on to a login node.

2. Become the user `hdfs`.

```
$ su -hdfs
```

3. Set the storage policy manually to `ONE_SSD` or `ALL_SSD` for that HDFS directory using the `hdfs storagepolicies` command.



WARNING: Spark scratch space (`spark.local.dir`) is also located on the SSDs. Setting the HDFS policy to `ONE_SSD` or `ALL_SSD` will reduce the scratch space available to Spark applications.

```
$ hdfs storagepolicies -setStoragePolicy -path path -policy policy
```

4. Verify that the storage policy has been changed.

```
$ hdfs storagepolicies -getStoragePolicy -path /ptmp
```

7 Apache Spark Support

Apache™ Spark™ is a fast and general engine for data processing. It provides high-level APIs in Java, R, Scala and Python, and an optimized engine.

Spark core and ecosystem components currently supported on the Urika-GX system include:

- **Spark Core, DataFrames, and Resilient Distributed Datasets (RDDs)** - Spark Core provides distributed task dispatching, scheduling, and basic I/O functionalities.
- **Spark SQL, DataSets, and DataFrames** - The Spark SQL component is a layer on top of Spark Core for processing structured data.
- **Spark Streaming** - The Spark Streaming component leverages Spark Core's fast scheduling capabilities to perform streaming analytics.
- **MLlib Machine Learning Library** - MLlib is a distributed machine learning framework on top of Spark.
- **GraphX** - GraphX is a distributed graph processing framework on top of Spark. It provides an API for expressing graph computations.

This section provides a quick guide to using Apache Spark on Urika-GX. Please refer to the official Apache Spark documentation for detailed information about Spark, as well as documentation of the Spark APIs, programming model, and configuration parameters.

Run Spark Applications

The Urika-GX software stack includes Spark configured and deployed to run under Mesos.

Mesos on Urika-GX is configured with three Mesos masters using ZooKeeper. To connect to Mesos, Spark's Master is set to:

```
mesos://zk://zoo1:2181,zoo2:2181,zoo3:2181/mesos
```

This is the default setting on Urika-GX and is configured via the Spark start up scripts installed on the system.

Spark on Urika-GX uses coarse-grained mode by default, but fine-grained can be enabled by setting `spark.mesos.coarse` to `false` in `SparkConf`.

To launch Spark applications or interactive shells, use the Spark launch wrapper scripts on `/opt/cray/spark2/default/bin` on login nodes. These scripts will be located in the user's path as long as the appropriate Spark module is loaded (it will be `spark/2.3.0` by default when users log in to the login nodes). These wrapper scripts include:

- `spark-shell`
- `spark-submit`
- `spark-sql`
- `pyspark`

- `sparkR`
- `run-example`

By default, `spark-shell` will start a small, 32 core interactive Spark instance to allow small-scale experimentation and debugging. To create a larger instance in the Spark shell, pass the `--total-executor-cores No_of_Desired_cores` command-line flag to `spark-shell`. To request a smaller or larger instance, again pass the `--total-executor-cores No_of_Desired_cores` command-line flag. Memory allocated to Spark executors and drivers can be controlled with the `--driver-memory` and `--executor-memory` flags. By default, 16 gigabytes are allocated to the driver, and 96 gigabytes are allocated to each executor, but this will be overridden if a different value is specified via the command-line, or if a property file is used.

Further details about starting and running Spark applications are available at <http://spark.apache.org>

Build Spark Applications

Urika-GX ships with Maven installed for building Java applications (including applications utilizing Spark's Java APIs), and Scala Build Tool (sbt) for building Scala Applications (including applications using Spark's Scala APIs). To build a Spark application with these tools, add a dependence on Spark to the build file. For Scala applications built with `sbt`, add this dependence to the `.sbt` file, such as in the following example:

```
scalaVersion := "2.11.8"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0"
```

For Java applications built with Maven, add the necessary dependence to the `pom.xml` file, such as in the following example:

```
<dependencies>
  <dependency> <!-- Spark dependency -->
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_2.11</artifactId>
    <version>2.2.0</version>
  </dependency>
</dependencies>
```

For detailed information on building Spark applications, please refer to the current version of Spark's programming guide at <http://spark.apache.org>.

Conda Environments

When the system is running in the default mode, PySpark on Urika-GX is aware of Conda environments. If there is an active Conda environment (the name of the environment is prepended to the Unix shell prompt), the PySpark shell will detect and utilize the environment's Python. To override this behavior, manually set the `PYSPARK_PYTHON` environment variable to point to the preferred Python. For more information, see [Enable Anaconda Python and the Conda Environment Manager](#) on page 29.

When the system is running in the secure mode, Spark jobs (running on Kubernetes) are not aware of Conda environments or user Python versions.

Spark Configuration Differences

Spark's default configurations on Urika-GX have a few differences from the standard Spark configuration:

- **Changes to improve execution over a high-speed interconnect** - The presence of the high-speed network on the system changes some of the tradeoffs between compute time and communication time. Because of this, the default settings of `spark.shuffle.compress` has been changed to `false` and that of

`spark.locality.wait` has been changed to 1. This results in improved execution times for some applications. If an application is running out of memory or temporary space, try changing this back to `true`.

- **Increases to default memory allocation** - Spark's standard default memory allocation is 1 Gigabyte to each executor, and 1 Gigabyte to the driver. Due to large memory nodes, these defaults were changed to 96 Gigabytes for each executor and 16 Gigabytes for the driver.
- **Mesos coarse-grained mode** - Urika-GX ships with this mode enabled as coarse-grained mode significantly lowers startup overheads.

Limitations

Spark Shells using Kubernetes (i.e., those launched under the secure service mode) will be limited to 16 cores and 60 GiB memory and this cannot be overridden at the command line. This is due to a limitation of the lack of native Spark Shell support in the Spark on Kubernetes project that Cray has provided a workaround for in this release.

7.1 Monitor Spark Applications

Urika-GX enables monitoring individual Spark applications as well as the list of completed Spark applications via Spark web UIs.

Use the `urika-service-mode` command to check the service mode the system is currently running in, as that dictates the availability of Spark web UIs. For more information, refer to the `urika-service-mode` man page and [Urika-GX Service Modes](#) on page 7.

Monitoring Individual Spark Applications

Every Spark application launches a web UI at port 4040. This UI displays useful information about the Spark application, such as:

- a list of scheduler stages and tasks
- a summary of the Spark Resilient Distributed Dataset (RDDs) sizes and memory usage
- environmental information
- information about the active/running executors

This UI can be accessed at: `http://hostname-login1:4040` or `http://hostname-login2:4040`. If multiple applications are launched, the subsequent jobs will run on port 4041, 4042, 4043, onwards.

View Completed Spark Applications

The Spark History Server displays information about completed Spark applications. Access the Spark History Server UI via the **Urika-GX Applications Interface**. Though this is the recommended method of accessing the Spark History Server UI, it can also be accessed via the port it runs on, i.e. at: `http://hostname-login1:18080` or `http://hostname-login2:18080`

Figure 5. Urika-GX Applications Interface

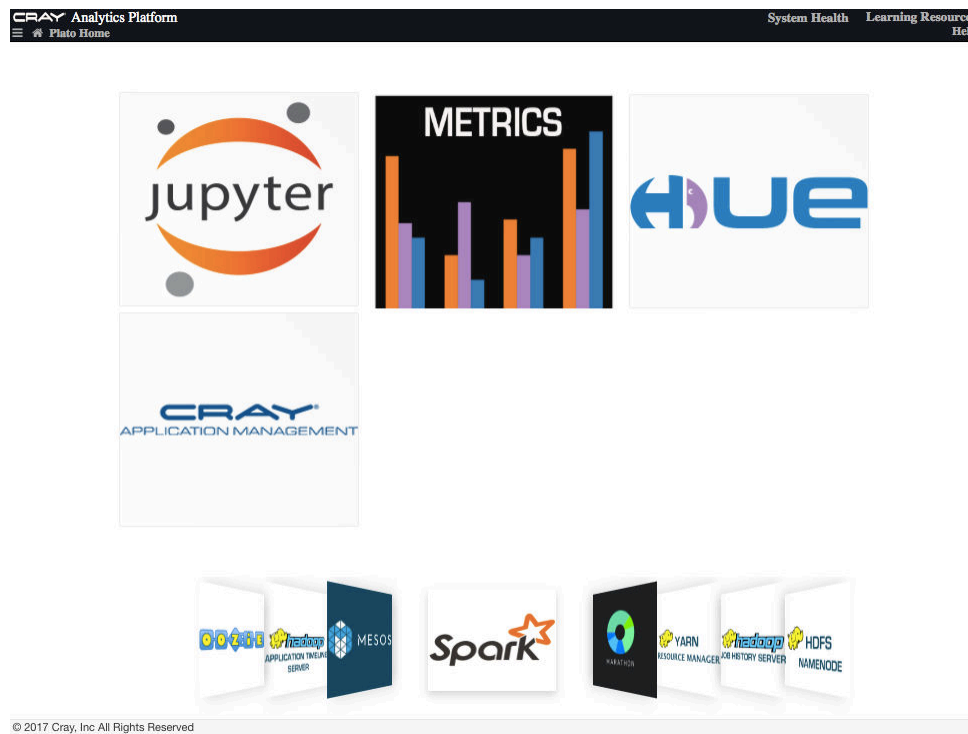


Figure 6. Spark History Server UI

Cray Analytics Platform System Health Learning Resources Help

Spark 2.0.1 History Server

Event log directory: hdfs://192.168.0.1:8020/user/spark/applicationHistory

Show entries Search:

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0262	Triangle Counts	2016-11-10 19:54:37	2016-11-10 19:54:45	8 s	builder	2016-11-10 19:54:45
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0261	Twitter Pagerank	2016-11-10 19:54:08	2016-11-10 19:54:29	21 s	builder	2016-11-10 19:54:32
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0260	Twitter Connected Components	2016-11-10 19:52:56	2016-11-10 19:54:03	1.1 min	builder	2016-11-10 19:54:03
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0259	SparkR-ML-example	2016-11-10 19:51:36	2016-11-10 19:52:50	1.2 min	builder	2016-11-10 19:52:50
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0258	SparkR-DataFrame-example	2016-11-10 19:51:13	2016-11-10 19:51:29	16 s	builder	2016-11-10 19:51:29
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0257	SparkR-data-manipulation-example	2016-11-10 19:50:44	2016-11-10 19:51:06	23 s	builder	2016-11-10 19:51:06
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0256	SparkR-DataFrame-example	2016-11-10 19:50:30	2016-11-10 19:50:36	7 s	builder	2016-11-10 19:50:36
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0255	Spark Hive Query Operations	2016-11-10 19:50:18	2016-11-10 19:50:27	8 s	builder	2016-11-10 19:50:27
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0254	Spark SQL	2016-11-10 19:50:09	2016-11-10 19:50:15	6 s	builder	2016-11-10 19:50:15
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0253	Spark Dataframes	2016-11-10 19:49:57	2016-11-10 19:50:07	11 s	builder	2016-11-10 19:50:07
local-1478807065810	Spark Pi	2016-11-10 19:44:24	2016-11-10 19:44:31	6 s	builder	2016-11-10 19:44:31
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0252	Spark Pi	2016-11-10 19:44:04	2016-11-10 19:44:08	5 s	builder	2016-11-10 19:44:08
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0251	Spark Pi	2016-11-10 19:43:43	2016-11-10 19:43:48	5 s	builder	2016-11-10 19:43:48
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0250	Spark Pi	2016-11-10 19:43:32	2016-11-10 19:43:37	5 s	builder	2016-11-10 19:43:37
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0249	movies_ALS.py	2016-11-10 19:37:33	2016-11-10 19:42:49	5.3 min	builder	2016-11-10 19:42:49
afe4dd32-b0f4-42a4-a07d-0ad05b9c8328-0248	Triangle Counts	2016-11-10 19:37:23	2016-11-10 19:37:30	8 s	builder	2016-11-10 19:37:30

© 2016 Cray, Inc All Rights Reserved

The preceding web UIs contain custom Cray enhancements that link Spark tasks in the UIs to Grafana dashboards that display compute node system metrics during the tasks' executions. These can be accessed by

clicking links in the **executor ID/host** column in the tasks table on the **Stage** tab, or by selecting the **Compare** check boxes of multiple tasks in the task table and clicking the **Compare** link at the top of the table.

7.2 Remove Temporary Spark Files from SSDs

Prerequisites

This procedure requires root privileges.

About this task

Spark writes temporary files to the SSDs of the compute nodes that the Spark executors run on. Ordinarily, these temporary files are cleaned up by Spark when its execution completes. However, sometimes Spark may fail to fully clean up its temporary files, such as, when the Spark executors are not shut down correctly. If this happens too many times, or with very large temporary files, the SSDs may begin to fill up. This can cause Spark jobs to fail or slow down.

Urika-GX checks for any idle nodes once per hour, and cleans up any left over temporary files. This is handled by a cron job running on one of the login nodes that executes the `/usr/sbin/cleanupssds.sh` script once per hour. Follow the instructions in this procedure if this automated clean up ever proves to be insufficient.

Procedure

1. Log on to one of the login nodes as root.
2. Kill all the processes of running Spark jobs.
3. Execute the `/usr/sbin/cleanupssds.sh` script.

```
# /usr/sbin/cleanupssds.sh
```

7.3 Obtain Additional Temporary Space for Running Spark Jobs

Prerequisites

Ensure that the login node is accessible.

About this task

Temporary directories can be configured to use both the SSDs and HDDs to provide additional temporary space for running large Spark jobs. Although using a combination of SSDs and HDDs for running Spark jobs provides additional flexibility for running Spark jobs requiring large amount of shuffle space, it is important to note that using just SSDs for running Spark jobs provides optimal performance.

Procedure

1. Log on to a login node, such as login1.

```
$ ssh login1
```

2. Create a file named `spark_local_dirs.hdd` in the home directory, as shown in the following example:

```
$ echo true >> /home/users/username/spark_local_dirs.hdd
```

The preceding example, creates a file named `spark_local_dirs.hdd` and adds `true` to its contents.

If the `spark_local_dirs.hdd` file exists in the home directory, additional temporary scratch space will be used while running Spark jobs. On the other hand, if this file does not exist in the home directory, the default option for temporary space will be used, i.e., just the SSDs will be used for running Spark jobs. Delete the `spark_local_dirs.hdd` file if it is required to revert to the default settings, using the following command:

```
$ rm /home/users/username/spark_local_dirs.hdd
```

7.4 Enable Anaconda Python and the Conda Environment Manager

About this task

In addition to the default system Python, Urika-GX also ships with the Anaconda Python distribution version 4.1.1, including the Conda package and environment manager. Users can enable and/or disable Anaconda for their current shell session by using environment modules.

Procedure

1. Log on to a login node.

`nid00030` is used as an example for a login node in this procedure.

2. Load the analytics module

```
$ module load analytics
```

3. Allocate resources, using workload management specific commands.

Example for allocating resources using Slurm.

```
$ salloc -N numberOfResources
```

Example for allocating resources using PBS Pro.

```
$ qsub -I -lnodes=numberOfResources
$ module load analytics
$ module load openmpi/gcc/64/3.0.0
```

The path shown in the preceding example for loading the openMPI module depends on the system.

4. Load the `anaconda3` module.

```
[user@nid00030 ~]$ module load anaconda3
```

Loading the anaconda3 module will make Anaconda Python the default Python, and enable Conda environment management.

5. Create a Conda environment.

The following example creates a Conda environment with `scipy` and all of its dependencies loaded:

```
[user@nid00030 ~]$ conda create --name scipyEnv scipy
```

IMPORTANT: Use the `conda config --add envs_dirs path_to_directory` command if it is required to set an alternate environments directory for Conda. `path_to_directory` must be a directory that is mounted within the container. This is particularly useful when the home directory space is limited.

6. Activate the Conda environment.

```
[user@nid00030 ~]$ source activate scipyEnv
```

For more information about Anaconda, refer to <https://docs.anaconda.com>. For additional information about the Conda environment manager, please refer to <http://conda.pydata.org/docs/>

7. Verify that the name of the environment is prepended to the shell prompt to ensure that the Conda environment has been activated.

In the following example, (`scipyEnv`) has been prepended in the prompt, which indicates that the Conda environment has been activated.

```
(scipyEnv) [user@nid00030 ~]$
```

Once the Conda environment has been activated, Python and PySpark will both utilize the selected environment. If it is not required to have PySpark utilize the environment, manually set `PYSPARK_PYTHON` to point to a different Python installation.

- To deactivate a Conda environment, use `source deactivate`:

```
(scipyEnv) [user@nid00030 ~]$ source deactivate
```

- To disable Anaconda and Conda, and switch back to the default system Python, unload the module:

```
(scipyEnv) [user@nid00030 ~]$ module unload anaconda3
```

For more information about Anaconda, refer to <https://docs.anaconda.com>. For additional information about the Conda environment manager, please refer to <http://conda.pydata.org/docs/>

7.5 Provide Kerberos Credentials to Spark

Prerequisites

Ensure that the Kerberos and Spark services are running.

About this task

Under the secure mode, Spark uses Kerberos to authenticate users with HDFS, thus allowing users to securely access their data on HDFS. While this is managed automatically for users on Urika-GX, it can also be done manually if required.

Procedure

1. Manually specify either of the following pairs of arguments to the `spark-shell` or `spark-submit` commands when launching a Spark shell or submitting a Spark job.
 - The `--principal PRINCIPAL` and `--keytab KEYTAB_FILE` arguments
 - The `--conf NAME=VALUE` arguments with `--conf spark.yarn.principal=PRINCIPAL` and `--conf spark.yarn.keytab=KEYTAB_FILE`
2. Ensure that the log output contains the values that were supplied.

```
INFO security.UserGroupInformation: Login successful for user user@REALM using keytab file /
path/to/keytab
```

An error message similar to the following will be returned if the credentials are incorrect:

```
Exception in thread "main" java.io.IOException: Login failure for user@REALM
from key tab /path/to/keytab: \
javax.security.auth.login.LoginException: Unable to obtain password from user
```

7.6 Redirect a Spark Job to a Specific Directory

About this task

This procedure provides instructions for redirecting Spark event logs to a directory of choice.

Procedure

1. Log on to a login node.
2. Execute the `spark-shell` command with the `--conf spark.eventLog.dir` argument, specifying the directory of choice.

```
$ spark-shell --conf spark.eventLog.dir=hdfs:///user/userName/sparkHistory
```

The preceding Spark job will not be displayed up on the Spark History Server

7.7 Modify the Default Number of Maximum Spark Cores

Prerequisites

Ensure that Spark is not running using the `urika-state` command. For more information, see the `urika-state` man page.

About this task

Defaults for `spark-shell` and `spark-submit` can be modified via the `--total-executor-cores` `NUM_CORES` flag.

Procedure

Change the number of default Spark maximum cores using either of the following mechanisms:

- Modify the default Spark maximum cores via Jupyter Notebook
 1. Access the Jupyter Notebook UI at `http://hostname-login1:7800` or using either the **Cray Application Interface** at `http://hostname`.
 2. Create a new notebook with the following content, replacing `NUM_CORES` with the desired number of maximum Spark cores.

- Example for Scala:

```
sc.stop()
sc = SparkContext(conf=SparkConf().set("spark.cores.max", "NUM_CORES"))
```

- Example for PySpark:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
val conf = new SparkConf().set("spark.cores.max", "NUM_CORES")
val sc = new SparkContext(conf)
```

- Example for SparkR:

```
sparkR.session(spark.cores.max = "NUM_CORES")
```

- Change the number of default Spark maximum cores via the command-line

1. Log on to the login node.

```
$ ssh login1
```

2. Set `spark.cores.max` to the desired value by executing the following commands, depending on the type of shell used.

- Example for Scala

```
spark-shell
sc.stop()
sc = SparkContext(conf=SparkConf().set("spark.cores.max", "NUM_CORES"))
```

- Example for PySpark

```
pyspark
import org.apache.spark.SparkContext
```



```
import org.apache.spark.SparkConf
val conf = new SparkConf().set("spark.cores.max", "NUM_CORES")
val sc = new SparkContext(conf)
```

- Example for SparkR

```
sparkr
sparkR.session(spark.cores.max = "NUM_CORES")
```

7.8 Execute Spark Jobs on Kubernetes

Spark jobs run inside containers, which are managed via Kubernetes on the Urika-GX system. This section provides some examples for executing Spark jobs, retrieving output, and viewing logs etc.

The system needs to be running in the secure mode and the user needs to be logged on a login node to run the examples shown in this section.

Running a Spark Pi Example Job

The following examples shows how to run a simple Spark Pi job inside a container. It uses *spark.app.name* as the Spark job's name.

```
$ spark-submit --class org.apache.spark.examples.SparkPi \
--conf spark.app.name=spark-pi \
local:///opt/spark/examples/target/scala-2.11/jars/spark-
examples_2.11-2.2.0-k8s-0.5.0.jar
```

The path to the JAR file must be relative to the path inside the container, not the path that exists on the system. Inside the container, the Spark home directory is */opt/spark* instead of */opt/cray/spark2/default*.

The preceding command produces output similar to the following (only a portion of the output is shown below for brevity):

```
2018-02-26 16:16:47 INFO  HadoopStepsOrchestrator:54 - Hadoop Conf
directory: /etc/hadoop/conf
2018-02-26 16:16:47 INFO  HadoopConfBootstrapImpl:54 -
HADOOP_CONF_DIR defined. Mounting Hadoop specific files
2018-02-26 16:16:48 WARN  NativeCodeLoader:62 - Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
2018-02-26 16:16:48 INFO  LoggingPodStatusWatcherImpl:54 - State
changed, new state:
  pod name: spark-pi-1519683406605-driver
  namespace: username
  labels: spark-app-selector ->
spark-027d506894bd4b2ca86692f03f9fab5a, spark-role -> driver
  pod uid: bceaf8b2-1b42-11e8-8b39-001e67d33475
  creation time: 2018-02-26T22:16:48Z
  service account name: spark
  volumes: spark-local-dir-0-tmp, hadoop-properties, spark-token-
pxz79
  node name: N/A
  start time: N/A
  container images: N/A
```

```
phase: Pending
status: []
.....
```

The output of the Spark Driver can be viewed by executing `kubectl logs pod_name` and looking at the pod's logs. The pod's name is displayed near the top of the console output, as shown in the preceding example. Execute the `kubectl logs pod_name` command and `grep` the output, as shown below:

```
$ kubectl logs spark-pi-1519683406605-driver | grep "is roughly"
Pi is roughly 3.1351356756783786
```

Spark Executor pods are cleaned up and deleted after they finish running. Therefore, their output is not accessible.

Running a Spark Pi Example Job

A Pyspark pi example job is very similar to a Scala Spark PI, but information is specified slightly differently. If there are any JAR files, they should be provided via the `--jars` flag.

```
$ bin/spark-submit --conf spark.app.name=pyspark-pi \
--jars local:///opt/spark/examples/target/scala-2.11/jars/spark-
examples_2.11-2.2.0-k8s-0.5.0.jar \
local:///opt/spark/examples/src/main/python/pi.py
```

Execute the `kubectl logs pod_name` command and `grep` the output, as shown below:

```
# kubectl logs pyspark-pi-1519684161476-driver | grep "is roughly"
Pi is roughly 3.141600
```

Using HDFS

The `HADOOP_CONF_DIR` parameter will automatically be set to the appropriate value for the current user during Spark start up.

How to Run Jobs and Use the Resource Staging Server

Simply provide the location of the Spark jar and files on the local file system and they will be loaded into the Spark Resource Staging Server so that resources will be available inside the Spark container.

```
$ bin/spark-submit --class TriangleCounts --conf spark.app.name=spark-
triangles \
/home/users/builder/nid00006/workspace/socrates-cactus-spark-tests/
target/scala-2.11/spark-tests_2.11-1.0.jar \
/user/builder/datasets/cactus-spark-triangles/small-triangles.txt
```

Check the results using the pod name.

```
$ kubectl logs spark-triangles-1520878896538-driver | grep "riangles:"
numTriangles: 10624230
Number of triangles: 3541410
```

Resource Configuration for Spark Jobs

The following Spark configuration settings may be used to control the amount of resources that Spark will request from Kubernetes for any job launched using `spark-submit` under the secure service mode, i.e., under Kubernetes:

Table 5. Spark Configuration Settings and their Default Values

Configuration Setting	Default Value	Notes
<code>spark.executor.instances</code>	5	Number of Spark executor containers that will be launched to provide job execution under Kubernetes
<code>spark.executor.cores</code>	1	Number of cores requested per executor
<code>spark.executor.memory</code>	96g	Amount of memory requested per executor
<code>spark.driver.cores</code>	1	Number of cores requested for the driver This should be increased if a job does a lot of work in the driver e.g. aggregations, result collection
<code>spark.driver.memory</code>	16g	Amount of memory requested per driver



CAUTION: Please be aware that due to Kubernetes's service scheduling mechanism, there are always some services running on all the nodes, using fractional CPU cycles. This may block any requests that attempt to use the maximum number of cores on the system because a small fraction of those cores is already allocated.

If a job is not showing any progress, find out the current status of the associated Kubernetes pod by running `kubectl describe pod pod-name`. If there are insufficient resources to launch a job, the system will return a message similar to the following at the end of the output:

```
Warning FailedScheduling 55s (x8 over 1m) default-scheduler 0/16 nodes are available: 1
PodToleratesNodeTaints, 16 Insufficient cpu.
```

7.9 Multi-tenant Spark Thrift Server on Urika-GX

Urika-GX enables tenants to have a tenant-isolated Hive MetaStore, in addition to the Spark Thrift Server. The metastore allows schema to be shared between `spark-sql` shells and Spark Thrift clients. The Spark Thrift

Server is an 'on demand' service, meaning that users launch it as required and stop it when it is not needed. The tenant metastore is a pod comprised of two containers running within the tenant's namespace. One of these containers runs the Hive metastore service, while the other runs a Postgres database.

NOTE:

The metastore allows schema to be shared between `spark-sql` shells and Spark Thrift clients

Hive Metastore Management

Tenant specific metastores are created as part of the tenant creation process. Tenants can use the following commands within their VM to interact with the metastore:

- `check-metastore` - Checks the status of the metastore, indicates to the user whether or not it is running and the IP address of the metastore.
- `start-metastore` - Starts the tenant metastore, if there is not one already running.
- `stop-metastore` - Stops the tenant metastore service.



CAUTION: The metastore must be running in order for the Thrift Server to launch successfully.

Spark Thrift Server Management

Tenants can use the following commands within their VM to interact with the Spark Thrift Server:

- `start-thriftserver` - Configures the tenant specific Spark Thrift Server and launches the Spark Thrift Server spark job in the tenant's Kubernetes namespace
- `stop-thriftserver` - Stops the tenant specific Spark Thrift Server job and does requisite cleanup.

The Spark Thrift Server is a shared service within a tenant, therefore only one Spark Thrift Server instance per tenant should be running.

Limitations

The current version of Spark on Urika-GX lacks support for long running spark jobs with secure HDFS. As a result, a Spark Thrift Server instance can only run as long as the HDFS delegation tokens are valid for. Currently this period is 1 day. After this time, the Spark Thrift Server instance can no longer query data in secure HDFS. If this happens, the Spark Thrift Server instance must be restarted using the stop/start commands above.

Authentication/Authorization

Tenant users authenticate to the Spark Thrift Server using their LDAP credentials. The Spark Thrift Server relies on tenant membership for authorizing users. Therefore, if a user is added as a member to a tenant via the `ux-tenant-add-user` command, the user will be automatically granted access to the Spark Thrift Server. No separate authorization mechanisms are necessary.

Administering Tenant Spark Thrift Server Components

It may prove useful for a system administrator to be able to check the status of a tenants Spark Thrift Server and/or metastore. They may do so using `kubectl` commands from the login node.

To see the state of the Kubernetes POD containing the Spark Thrift Server driver:

```
root@login1# kubectl get pods -n TENANT-VM thrift-server-TENANT-VM --show-all
```

To see if the Spark Thrift Server has fully initialized, assuming the previous command showed the POD was "Running"

```
root@login1# kubectl logs -n TENANT-VM thrift-server-TENANT-VM | grep 'Started ThriftHttpCLIService'
```

To see the state of the Kubernetes POD containing the Metastore server:

```
root@login1# kubectl get pods -n <TENANT-VM> metastore-TENANT-VM --show-all
```

The status should be 'Running' and the Status '2/2' (indicating both containers were successfully started).

To see if the Metastore server has fully initialized, assuming the previous command showed the POD was "Running"

```
root@login1# kubectl logs -c hive -n <TENANT-VM> metastore-TENANT-VM | grep 'schemaTool completed'
```

8 Use Apache Mesos on Urika-GX

Apache™ Mesos™ acts as the primary resource manager on the Urika-GX platform. It is a cluster manager that provides efficient resource isolation and sharing across distributed applications and/or frameworks. It lies between the application layer and the operating system and simplifies the process of managing applications in large-scale cluster environments, while optimizing resource utilization.

Architecture

Major components of a Mesos cluster include:

- **Mesos agents/slaves** - Agents/slaves are the worker instances of Mesos that denote resources of the cluster.
- **Mesos masters** - The master manages agent/slave daemons running on each cluster node and implements fine-grained sharing across frameworks using resource offers. Each resource offer is a list of free resources on multiple agents/slaves. The master decides how many resources to offer to each framework according to an organizational policy, such as fair sharing or priority.

By default, Urika-GX ships with three Mesos masters with a quorum size of two. At least two Mesos masters must be running at any given time to ensure that the Mesos cluster is functioning properly. Administrators can use the `urika-state` and `urika-inventory` commands to check the status of Mesos masters and slaves. Administrators can also check the status of Mesos by pointing their browser at `http://hostname-login1:5050` and ensuring that it is up. In addition, executing the `ps -ef | grep mesos` command on the login nodes displays the running Mesos processes.

Components that Interact with Mesos

- **Frameworks** - Frameworks run tasks on agents/slaves. The Mesos Master offers resources to frameworks that are registered with it. Frameworks decide either to accept or reject the offer. If a framework accepts the offer, Mesos offers the resources and the framework scheduler then schedules the respective tasks on resources. Each framework running on Mesos consists of two components:
 - A scheduler that is responsible for scheduling the tasks of a framework's job, within the accepted resources.
 - An executor process that is launched on agent/slave nodes to run the framework's tasks.
- In addition to the aforementioned components, Urika-GX also supports Marathon and `mrunch` (the Cray-developed application launcher) as ecosystem components of Mesos. `mrunch` is built upon Marathon commands for ease of use and running data secure distributed applications. The `mrunch` command sets up resources for CGE and HPC jobs.

On Urika-GX, all tasks launched directly from Marathon need to be run as user `marathon`, and cannot be run as any other user ID. If a user tries to launch applications/tasks as non-Marathon user, the application will fail with error "Not authorized to launch as `userID`". This behavior has no impact on Hadoop, Spark, `mrunch` and/or CGE jobs.

Role of HAProxy

Requests received on the login nodes for the following services are proxied using HAProxy to the Urika-GX compute nodes:

- YARN Resource Manager
- HDFS NameNode
- Secondary HDFS NameNode
- Hadoop Application Timeline Server
- Hadoop Job History Server
- Spark History Server
- Oozie

For services like Mesos Masters and Marathon, while there are 3 instances running, one of them is the active leader. Requests received by the login node are proxied to the currently active leader. If a leader runs into issues, one of the backup leaders take over and the requests are proxied to the current leader.

HAProxy can be configured to provide SSL. Some possible solutions are documented in the security section of "*Urika®-GX System Administration Guide*".

Viewing Mesos Metrics from the CLI

Use the Cray-developed `urika-mesos_metrics` script to view Mesos related details. This script is located in the `/root/urika-tools/urika-cli` directory on the SMW and needs to be run as root.

Following is a sample output of the `urika-mesos_metrics` script:

```
# urika-mesos_metrics
HTTP/1.1 200 OK
Proceeding further...
***** MESOS CLUSTER METRICS *****
Total cpus : 984
Used cpus : 0
Master elected : 1
***** MESOS FRAMEWORK METRICS *****
Frameworks active : 1
Frameworks connected : 1
Frameworks disconnected: 0
Frameworks inactive: 0
***** MESOS SLAVE METRICS *****
Slaves active : 41
Slaves connected : 41
Slaves disconnected: 0
Slaves inactive: 0
```

Troubleshooting information

- If the system indicates that the `mesos-slave` process is running, but it is not possible to schedule jobs, execute the following commands as root on each of the agent/slave node:

```
# systemctl stop urika-mesos-slave
# rm -vf /var/log/mesos/agent/meta/slaves/latest
# systemctl start urika-mesos-slave
```

- If the Mesos UI displays orphaned Mesos tasks, refer to [Diagnose and Troubleshoot Orphaned Mesos Tasks](#) on page 126 to debug the issue.
- Mesos logs are located at `/var/log/mesos`, whereas log files of Mesos framework are located under `/var/log/mesos/agent/slaves/` on the node(s) the service runs on.

For more information about Mesos, visit <http://mesos.apache.org>.

8.1 Access the Apache Mesos Web UI

Prerequisites

- Before performing this procedure use the `urika-state` script to verify that the Mesos service is running.
- Check the service mode by executing `urika-service-mode` to ensure that the system is running in the service mode required for Mesos to run.

About this task

The Mesos web UI can be used to monitor components of the Mesos cluster, such as the Mesos slaves, aggregated resources and frameworks.

Do not launch applications through Mesos directly because all the frameworks are pre-configured on Urika-GX. Only a few frameworks (Spark and Marathon) are pre-configured to authenticate with Mesos.

Procedure

1. Access the Mesos UI using once of the following mechanisms:
 - Point a browse at the **Urika-GX Applications Interface** at: `http://hostname-login1` and then select the Mesos icon. This is the recommended approach.
 - Point a browser at `http://hostname-login1:5050` or `http://hostname-login2:5050`
2. Enter a username and the Mesos secret in the **Authentication Required** pop up's **UserName** and **Password** fields respectively.

The Mesos secret can be retrieved from the `/security/secrets/userName.mesos` file, which is located on the SMW.

After logging on to the Mesos web UI, the users can view tasks in the summary page as well as resources reserved for that particular user. `crayadm` and `root` are global Mesos users that can view all the running frameworks and resource usage.

8.2 Use mrun to Retrieve Information About Marathon and Mesos Frameworks

Cray has developed the `mruntime` command for launching applications. `mruntime` enables running parallel jobs on Urika-GX using resources managed by Mesos/Marathon. In addition, this command enables viewing the currently active

Mesos Frameworks and Marathon applications and enables specifying how `mrunch` should redirect STDIN. It provides extensive details on running Marathon applications and also enables cancelling/stopping currently active Marathon applications.

The Cray Graph Engine (CGE) uses `mrunch` to launch jobs under the Marathon framework on the Urika®-GX system.



CAUTION: The `mrunch` command cannot be executed within a tenant VM or while the system is operating in the secure service mode. Both the `munge` and `ncmd` system services must be running for `mrunch`/CGE to work. If either service is stopped or disabled, `mrunch` will no longer be able to function

The `mrunch` command needs to be executed from a login node. Some examples of using `mrunch` are listed below:

Launch a job with `mrunch`

```
$ mrunch /bin/date
Wed Aug 10 13:31:51 CDT 2016
```

Display information about frameworks, applications and resources

Use the `--info` option of the `mrunch` command to retrieve a quick snapshot view of Mesos frameworks, Marathon applications, and available compute resources.

```
$ mrunch --info
Active Frameworks:
  IBM Spark Shell : Nodes[10] CPUs[ 240] : User[builder]
  Jupyter Notebook : Nodes[ 0] CPUs[  0] : User[urika-user]
  marathon : Nodes[20] CPUs[ 480] : User[root]
Active Marathon Jobs:
  /mrunch/cge/user.dbport-2016-133-03-50-28.235572
    : Nodes[20] CPUs[320/480] : user:user cmd:cge-
server
Available Resources:
    : Nodes[14] CPUs[336] idle nid000[00-13]
    : Nodes[30] CPUs[480] busy nid000[14-29,32-45]
    : Nodes[ 2] CPUs[??] down nid000[30-31]
```

In the example output above, notice the `CPUs[320/480]` indicates that while the user only specified `mrunch --ntasks-per-node=16 -N 20` (meaning the application is running on 320 CPUs), `mrunch` intends **ALL** applications to have exclusive access to each node it is running on, and thus will ask Mesos for **ALL** the CPUs on the node, not just the number of CPUs per node the user requested to run on.

Retrieve a summary of running Marathon applications

Use the `--brief` option of the `mrunch` command to obtain a more concise report on just the running Marathon applications and node availability.

```
$ mrunch --brief
N:20 CPU:320/480 <user> /mrunch/cge/
user.dbport-2016-133-03-50-28.235572 cge-server -d /mn...
Status: Idle:14 Busy:30 Unavail:2
```

Retrieve information on a specific Marathon application

Use the `--detail` option of the `mrunch` command to obtain additional information on a specific Marathon application. The application ID needs to be specified with this option.

```
$ mrunch --detail /mrunch/cge/user.dbport-2016-133-03-50-28.235572
Active Frameworks:
  IBM Spark Shell : Nodes[10] CPUs[ 240] : User[builder]
  Jupyter Notebook : Nodes[ 0] CPUs[  0] : User[urika-user]
  marathon : Nodes[20] CPUs[ 480] : User[root]
Active Marathon Jobs:
  /mrunch/cge/user.dbport-2016-133-03-50-28.235572
    : Nodes[20] CPUs[320/480] : user:<user> cmd:cge-
server
    : [nid00032.urika.com]: startedAt:
2016-05-12T17:05:53.573Z
    : [nid00028.urika.com]: startedAt:
2016-05-12T17:05:53.360Z
    : [nid00010.urika.com]: startedAt:
2016-05-12T17:05:53.359Z
    : [nid00007.urika.com]: startedAt:
2016-05-12T17:05:53.397Z
    : [nid00001.urika.com]: startedAt:
2016-05-12T17:05:53.384Z
    : [nid00019.urika.com]: startedAt:
2016-05-12T17:05:53.383Z
...
...
```

The additional information provided by the `--detail` option includes a list of all the node names the application is running on, and at what time the application was launched on those nodes.

Stop, cancel or abort a running Marathon application

Use the `--cancel` option of the `mrunch` command to stop, cancel or abort a running Marathon application. The application ID needs to be specified with this option.

```
$ mrunch --cancel /mrunch/cge/user.3750-2016-133-20-01-07.394582
Thu May 12 2016 15:01:21.284876 CDT[][mrunch]:INFO:App /mrunch/cge/user.
3750-2016-133-20-01-07.394582 has been cancelled
```

If the application has already been cancelled, or completes, or does not exist, the following message is displayed:

```
$ mrunch --cancel /mrunch/cge/user.3750-2016-133-20-01-07.394582
App '/mrunch/cge/user.3750-2016-133-20-01-07.394582' does not exist
```



CAUTION: The `root` user is allowed to use `mrunch --cancel` to kill any Marathon-started job. All other users can only kill the Marathon jobs they launched using the `mrunch` command. If a non-root user tries to use `mrunch --cancel` to cancel any Marathon job that was not launched by that user using `mrunch`, the system returns the following message:

```
mrunch: error: Users may only cancel their own mrunch jobs
```

Retrieve a list of nodes, CPU counts and available memory

- Use the `--resources` option of the `mrunc` command to obtain a complete list of nodes, CPU counts, and available memory.

```
$ mrunc --resources
NID  HEX  NODENAME  CPUs  MEM  STAT
0  0x00  nid00000  32  515758  idle
1  0x01  nid00001  32  515758  busy
2  0x02  nid00002  32  515758  idle
3  0x03  nid00003  32  515758  busy
4  0x04  nid00004  32  515758  busy
5  0x05  nid00005  32  515758  idle
6  0x06  nid00006  32  515758  idle
7  0x07  nid00007  32  515758  busy
8  0x08  nid00008  32  515758  busy
9  0x09  nid00009  32  515758  busy
10 0x0a  nid00010  32  515758  busy
11 0x0b  nid00011  32  515758  busy
12 0x0c  nid00012  32  515758  idle
13 0x0d  nid00013  32  515758  idle
14 0x0e  nid00014  32  515758  busy
15 0x0f  nid00015  32  515758  busy
16 0x10  nid00016  36  515756  idle
17 0x11  nid00017  36  515756  idle
18 0x12  nid00018  36  515756  busy
19 0x13  nid00019  36  515756  busy
20 0x14  nid00020  36  515756  idle
21 0x15  nid00021  36  515756  idle
22 0x16  nid00022  36  515756  busy
23 0x17  nid00023  36  515756  idle
24 0x18  nid00024  36  515756  idle
25 0x19  nid00025  36  515756  busy
26 0x1a  nid00026  36  515756  idle
27 0x1b  nid00027  36  515756  busy
28 0x1c  nid00028  36  515756  busy
29 0x1d  nid00029  36  515756  idle
30 0x1e  nid00030  0      0 unavail
31 0x1f  nid00031  0      0 unavail
32 0x20  nid00032  24  515758  busy
33 0x21  nid00033  24  515758  idle
34 0x22  nid00034  24  515758  idle
35 0x23  nid00035  24  515758  idle
36 0x24  nid00036  24  515758  idle
37 0x25  nid00037  24  515758  idle
38 0x26  nid00038  24  515758  busy
39 0x27  nid00039  24  515758  idle
40 0x28  nid00040  24  515758  idle
41 0x29  nid00041  24  515758  idle
42 0x2a  nid00042  24  515758  busy
43 0x2b  nid00043  24  515758  busy
44 0x2c  nid00044  24  515758  idle
45 0x2d  nid00045  24  515758  idle
```

Node names that are marked as `unavail` are hidden from Mesos as available compute resources, such as the login node (`nid00030`). In the example above, some nodes have 24 CPUs/node, some have 32 CPUs/node and some have 36 CPUs/node. While not a typical situation, `mrunc` does support this configuration, and a command such as `mrunc -n 144 -N`

4 would in fact be allowed to proceed, and would be limited to using 4 nodes on nid000[16-29], as they are the only ones with $(144/4 = 36)$ CPUs per node.

Configuration Files

When `mrunch` is invoked, it sets up some internal default values for required settings. `mrunch` will then check if any system defaults have been configured in the `/etc/mrunch/mrunch.conf` file. An example `mrunch.conf` file is shown below:

```
#
# (c) Copyright 2016 Cray Inc. All Rights Reserved.
#
# Anything after an initial hashtag '#' is ignored
# Blank lines are ignored.
#
#NCMDServer=nid00000
#MesosServer=localhost      # same as --host
#MesosPort=5050
#MarathonServer=localhost
#MarathonPort=8080
#DebugFLAG=False           # same as --debug
#VerboseFLAG=False         # same as --verbose
#JobTimeout=0-0:10:0       # ten minutes, same as --time
#StartupTimeout=30         # 30 seconds, same as --immediate
#HealthCheckEnabled=True   # Run with Marathon Health Checks enabled
#HCGracePeriodSeconds=5    # Seconds at startup to delay Health Checks
#HCIntervalSeconds=10      # Seconds between Health Check pings
#HCTimeoutSeconds=10       # Seconds to answer Health Check successfully
#HCMaxConsecutiveFailures=3 # How many missed health checks before app killed
```

If any of the lines above are not commented out, those values will become the new defaults for every `mrunch` invocation.

Additionally, after the system `/etc/mrunch/mrunch.conf` file is loaded (if it exists), the user's private `$HOME/.mrunch.conf` file will be loaded (if it exists). The following items should be noted:

- Any settings in the user's `$HOME/.mrunch.conf` file will take precedence over any conflicting settings in the system `/etc/mrunch/mrunch.conf` file.
- Any command-line arguments that conflict with any pre-loaded configuration file settings will take precedence for the current invocation of `mrunch`.

For more information, see the `mrunch(1)` man page.

8.3 Clean Up Residual mrunch Jobs

About this task

When the `mrunch --cancel` command does not completely clean up running `mrunch`-based jobs, the residual jobs need to be manually terminated, as described in this procedure.

Procedure

1. Launch an `mrunc` job

```
$ mrun sleep 100&
[1] 1883
```

2. Verify the launched `mrunc` job is running

```
$ mrun --brief
N: 1 CPU: 1/24 user1 /mrunc/2016-215-16-07-43.702792 sleep 100...
N:32 CPU: 512/768 user2 /mrunc/cge/user2.1025-2016-215-15-29-07.248656 -v /opt/
cray/cge...
Status: Idle:8 Busy:33 Unavail:5
```

3. Locate the process ID of the `mrunc` job

```
$ ps -fu $LOGNAME |grep mrun
user1 1883 19311 0 11:07 pts/8 00:00:00 /usr/bin/python /opt/cray/cge/
2.0.1045_re5a05d9_fe2.0.3_2016072716_jenkins/bin/mrun sleep 100
user1 1961 19311 0 11:08 pts/8 00:00:00 grep --color=auto mrun
```

4. Send a signal to terminate the `mrunc` job

```
$ kill 1883
```

5. Verify that the `mrunc` job is no longer running, and the idle node count has increased

```
$ mrun --brief
N:32 CPU: 512/768 users2 /mrunc/cge/user2.1025-2016-215-15-29-07.248656 -v /opt/cray/cge...
Status: Idle:9 Busy:32 Unavail:5
```

8.4 Launch an HPC Job Using `mrunc`

Prerequisites

Use the `urika-state` command to ensure that both the Mesos and Marathon services are running and to check if the system is operating in the service mode that supports using `mrunc`. For more information, see the `urika-state` man page and refer to [Urika-GX Service Modes](#) on page 7.

About this task

The `mrunc` command can be used to launch HPC jobs on the Urika-GX system.

The `mrunc` command cannot be executed within a tenant VM

Procedure

Launch the HPC job using the `mrunc` command, specifying the number of nodes to allocate.

In the following example, `my_hpc_app.exe` is used as an example for the name of the application to run.

```
$ mrun -n 32 -N 8 my_hpc_app.exe arg1 arg2 arg3
```

Refer to the `mrunch` man page for more information, further examples, environment variables, configuration files and command-line option descriptions of the `mrunch` command.

8.5 Manage Resources on Urika-GX

Mesos is used as the resource manager in the default service mode, whereas Kubernetes works as the resource manager in the secure services mode.

The resource management model of Mesos is different from traditional HPC schedulers. With traditional HPC schedulers, jobs request resources and the scheduler decides when to schedule and execute the job. Mesos on the other hand offers resources to frameworks that are registered with it. It is up to the framework scheduler to decide whether to accept or reject its resources. If framework rejects the offer, Mesos will continue to make new offers based on resource availability. Framework refers to implementation of different computing paradigms such as Spark, Hadoop, CGE etc.

For example, a user submits a spark job that requests 1000 cores to run. Spark registers as a framework with Mesos. Mesos offers its available resources to Spark. If Mesos offers 800 cores, Spark will either choose to accept the resources or reject it. If Spark accepts the resource offer, the job will be scheduled on the cluster. If it rejects the offer, Mesos will continue to make new offers.

Mesos Frameworks on Urika-GX

When users submit jobs to a Mesos cluster, frameworks register with Mesos. Mesos offers resources to registered frameworks. Frameworks can either choose to accept or reject the resource offer from Mesos. If the resource offer satisfies the resource requirements for a job, they accept the resources and schedule the jobs on the slaves. If the resource offer does not satisfy the resource requirements, frameworks can reject them. Frameworks will still be registered with Mesos. Mesos will update the resources it has at regular intervals (when an existing framework finishes running its job and releases the resources or when some other frameworks reject the resources) and continues to offer the resources to registered frameworks.

Each spark job is registered as a separate framework with Mesos. For each spark job that has been submitted, Mesos makes separate resource offers.

Marathon is registered as a single framework with Mesos. Marathon provides a mechanism to launch non-framework applications to run under Mesos. Marathon enables long-running services under Mesos such as databases, streaming engines etc. Cray has developed:

- the `mrunch` command, which sets up resources for CGE and HPC jobs. For more information, see the `mrunch` man page.
- scripts for setting up resources for YARN

These are submitted as applications to Marathon. Marathon negotiates for resources from Mesos and they get resources from Marathon.

Mesos tracks the frameworks that have registered with it. If jobs are submitted and there are no resources available, frameworks will not be dropped. Instead, frameworks will remain registered (unless manually killed by the user) and will continue to receive updated resource offers from Mesos at regular intervals. As an example, consider a scenario where there are four Spark jobs running and using all of the resources on the cluster. A user attempts to submit a job with framework γ and framework γ is waiting for resources. As each Spark job completes its execution, it releases the resources and Mesos updates its resource availability. Mesos will continue to give resource offers to the framework γ . γ can chose either to accept or reject resources. If γ decides to accept the

resources, it will schedule its tasks. If Y rejects the resources, it will remain registered with Mesos and will continue to receive resource offers from Mesos.

Allocation of resources to Spark by Mesos

Let us say that the `spark-submit` command is executed with parameters `--total-executor-cores 100 --executor-memory 80G`. Each node has 32 cores. Mesos tries to use as few nodes as possible for the 100 cores requested. So in this case it will start Spark executors on 4 nodes ($\text{roundup}(100 / 32)$). Each executor has been requested to have 80G of memory. Default value for `spark.mesos.executor.memoryOverhead` is 10% so it allocates 88G to each executor. So in Mesos, it can be seen that $88 * 4 = 352$ GB allocated to the 4 Spark executors. For more information, see the latest Spark documentation at <http://spark.apache.org/docs>

Additional points to note:

- On Urika-GX, the Mesos cluster runs in High Availability mode, with 3 Mesos Masters and Marathon instances configured with Zookeeper.
- Unlike Marathon, Mesos does not offer any queue. Urika-GX scripts for flexing clusters and the `mrunc` command do not submit their jobs unless they know the resource requirement is satisfied. Once the flex up request is successful, YARN uses its own queue for all the Hadoop workloads.
- Spark accepts resource offers with fewer resources than what it requested. For example, if a Spark job wants 1000 cores but only 800 cores are available, Mesos will offer those 800 to the Spark job. Spark will then choose to accept or reject the offer. If Spark accepts the offer, the job will be executed on the cluster. By default, Spark will accept any resource offer even if the number of resources in the offer is much less than the number of nodes the job requested. However, Spark users can control this behavior by specifying a minimum acceptable resource ratio; for example, they could require that Spark only accept offers of at least 90% of the requested cores. The configuration parameter that sets the ratio is `spark.scheduler.minRegisteredResourcesRatio`. It can be set on the command line with `--conf spark.scheduler.minRegisteredResourcesRatio=N` where N is between 0.0 and 1.0.
- `mrunc` and `flex` scripts do not behave the way Spark behaves (as described in the previous bullet). `mrunc` accepts two command-line options:
 - `--immediate=XXX` (default 30 seconds)
 - `--wait` (default `False`)

When a user submits an `mrunc` job, if more resources are needed than Mesos currently has available, the command will return immediately, showing system usage and how many nodes are available vs how many nodes are needed. If the user supplies the `--wait` option, this tells `mrunc` to not return, but instead continue to poll Mesos until enough nodes are available. `mrunc` will continue to poll Mesos for up to `--immediate` seconds before timing out. Finally, once Mesos informs `mrunc` there are enough resources available; `mrunc` will post the job to Marathon.

When the requested resources are not available, `flex` scripts will display the current resources availability and exit.

- With `mrunc`, the exact need must be met. If the user asks for 8 nodes, all CPU and memory on 8 nodes must be free for Marathon to accept the offer on behalf of `mrunc`.
- The Marathon API does not offer a way to ask if the needs of a job can be fully satisfied before a request can be submitted. Therefore, Mesos is queried for its resource availability.
- Users request for resources from Mesos to give to YARN via Cray developed scripts for starting NodeManagers. The request is submitted to Marathon. This is called flex up. Once users get the requested resources, they can run their Hadoop jobs/ Hive queries / Oozie work-flows. Once they complete this, they

release the resources back to Mesos via the Cray flex scripts. Flex scripts require the exact number of nodes to address requests and cannot run with fewer resources. When the number of resources requested in the flex up request does not match the current number of resources that are available with Mesos, an error message is displayed indicating that the number of resources available is less than the number of requested resources and that the user can submit a new flex up request.

- If the system is loaded and other frameworks (e.g. Spark) keep submitting smaller jobs, flex scripts may keep exiting if they do not receive the required number of nodes. This could lead to starvation of Hadoop jobs.

8.6 Manage Long Running Services Using Marathon

Marathon is used by the Cray-developed command named `mr` to allocate node resources from Mesos and launch application instances as needed. In addition, Cray-developed scripts for starting a cluster of YARN Node Managers are also launched through Marathon.

Before using Marathon, ensure that the system is running in the service mode that allows use of this service. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.



CAUTION: Unless it is required to shut down YARN nodes, analytic applications that use the Cray-developed scripts for flexing a cluster should not be deleted through the Marathon UI, as doing will lead to loss of YARN nodes.

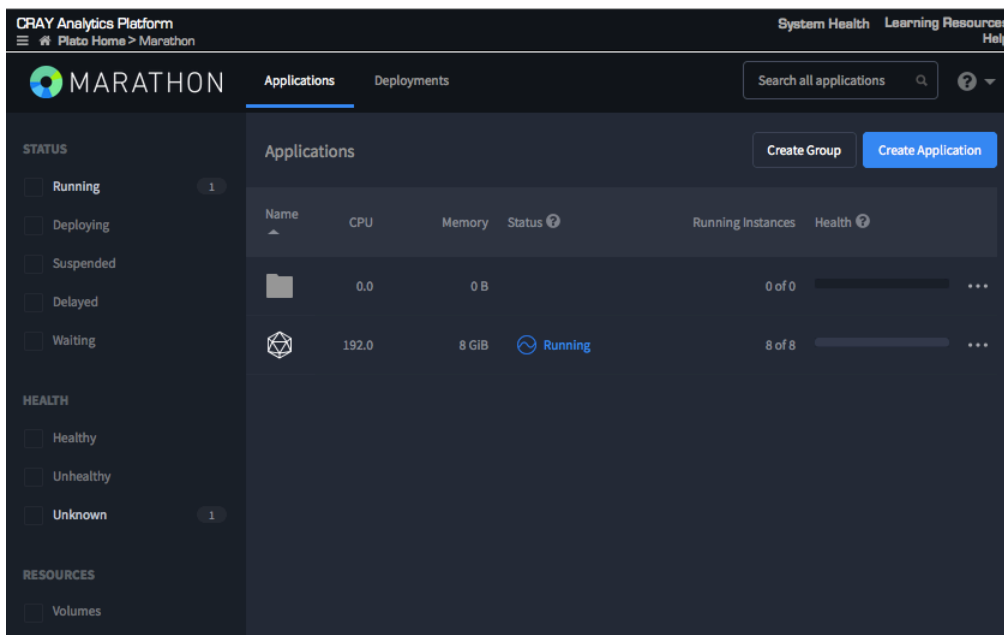
On the Urika-GX system, there are always three Mesos Masters and three Marathon instances running, while one of them is the active leader. Requests received by the login node are proxied to the currently active leader. If a leader runs into issues, one of the backup leaders take over and the requests are proxied to the current leader.

Access the Marathon web UI by selecting **Marathon** on the **Urika-GX Applications Interface**, located at: `http://hostname-login1`. Though this is the recommended method of accessing Marathon, it can also be accessed at the port it runs on, i.e. at `http://hostname-login1:8080` or `http://hostname-login2:8080`

Figure 7. Urika-GX Applications Interface



Figure 8. Marathon UI



Marathon also enables creating applications from the UI via the **Create Application** button, which displays the **New Application** pop up:

Figure 9. Create an Application Using the Marathon UI

Figure 10. Marathon UI

For additional information about using Marathon, select the help icon (?) and then select **Documentation**

8.7 Flex up a YARN sub-cluster on Urika-GX

Mesos is used as the main resource broker on Urika-GX, whereas Hadoop Yet Another Resource Manager (YARN) is the default resource manager for launching Hadoop workloads. To have YARN and Mesos co-exist on the system, Urika-GX features a number of scripts that are designed for scaling YARN cluster on Mesos. These scripts can expand or shrink a YARN cluster in response to events as per configured rules and policies. The cluster remains under the control of Mesos even when the cluster under the Mesos management runs other cluster managers. These scripts allows Mesos and YARN to co-exist and share resources with Mesos as the resource manager for Urika-GX. Sharing resources between these two resource allocation systems improves overall cluster utilization and avoids statically partitioning resources.

When a cluster is statically partitioned, a part of the cluster is reserved for running jobs. For example, a cluster may be statically partitioned to reserve x number of nodes out of a total of N number nodes to run only Hadoop jobs at any point of time. Under this configuration, if there are no Hadoop jobs running at a time, the reserved number nodes are idle, which reflects inefficient resource utilization. Mesos helps avoid static partitioning and also helps ensure proper resource utilization at any given point of time.

Once the Hadoop job has been launched and completed, the reserved resources need to be released back to Mesos. This is when the Cray developed script `urika-yam-flexdown` needs to be called. This script stops all the node managers of the named application, and then deletes the Marathon application. If there are more than one application running at a time and being managed by the Cray-developed flex scripts, this script will not stop the node managers of every application.

Cray-developed Scripts for Flexing Up/Flexing Down YARN Subcluster

The `urika-yam-flexup` and `urika-yam-flexdown` scripts are located in the `/opt/cray/urika-yam/default/bin` directory on the login nodes and can be executed on either of the two login nodes.

- To flex up, execute the `urika-yam-flexup` script, passing the number of nodes to be used as well as a unique name/identifier for the flex up request.

```
$ urika-yam-flexup --nodes Number_of_Nodes --identifier identifier_name --  
timeout timeoutInMinutes
```

For more information, see the `urika-yam-flexup` man page.

- To display the lists of existing applications and the resources allocated to each application, execute the `urika-yam-status` script.

```
$ urika-yam-status
```

For more information, see the `urika-yam-status` man page.

- To flex down, execute the `urika-yam-flexdown` script.
 - Executing the `urika-yam-flexdown` script as a root user

```
# urika-yam-flexdown --exact fullName
```

- Executing the `urika-yam-flexdown` script as a non-root user

```
$ urika-yam-flexdown --identifier name
```



CAUTION: The `urika-yam-flexdown` script needs to be passed the name of the Marathon application that has been used earlier to flex up. If the name of application is not provided, it will throw an error message and exit. If the application requested to flex down does not exist, the `urika-yam-flexdown` script will throw an error message saying no such application exists. If executing this script as a root user, please provide the complete/full name, as that returned by the `urika-yam-status` command. If executing this script as a non-root user, specify the same identifier as that used when the flex up request was issued.

- To flex down all the nodes, execute the `urika-yam-flexdown-all` script as root.

```
# urika-yam-flexdown-all
```

For more information, see the `urika-yam-flexdown-all` man page.

Timeout Intervals for Flex Up Requests

To release resources from YARN to Mesos and to ensure better resource utilization, a default timeout interval for flexing a job is defined in the `/etc/yam_conf` file. Users can provide a timeout value as a command-line argument to the flex up request to override the default value specified in `/etc/yam_conf`. The timeout interval can be configured either to a timeout value in minutes or it can be set to zero. If set to zero, the application will never timeout and will need to be manually flexed down. The default timeout value is 15 minutes, as specified in the `/etc/urika-yam.conf` file. The minimum acceptable timeout interval is 5 minutes.

Log locations

Logs related to the flex scripts are located under `/var/log/urika-yam.log` on login nodes

9 Access the Jupyter Notebook UI

Prerequisites

Before following this procedure, use the `urika-state` command to ensure that the Jupyter Notebook service is running.

About this task

Urika-GX comes pre-installed with the Jupyter Notebook, which is a web application that enables creating and sharing documents that contain live code, equations, visualizations, and explanatory text.

Urika-GX currently supports the following kernels:

- Bash
- R
- Python2
- Python3
- Spark 2.2.0
 - PySpark
 - Scala
 - sparkR



CAUTION: When using the Jupyter Notebook, if the Spark version changes, the predefined Jupyter Spark kernels will need to be updated to reflect those changes. Jupyter kernels configurations are stored under `/usr/local/share/jupyter/kernels` on login node 1.

TIP: If JupyterHub processes owned by the user remain running after the user has logged out from Jupyter these processes can be manually killed using the Linux kill command.

The Jupyter Notebook UI contains 3 tabs:

1. **Files** - Displays a list of existing notebooks on the left side of the screen, and enables creating new notebooks, folders, terminals, and text files via the **New** drop down on the right side of the screen. The **Upload** button can be used to upload an existing notebook.
2. **Running** - Displays a list of all the running elements, such as notebooks, and terminals, etc.
3. **Clusters** - Enables assigning a group of nodes to a configured cluster. Urika-GX does not ship with any pre-configured Python clusters.

Follow the steps in this procedure to access the Jupyter Notebook Web UI.

Procedure

1. Point a browser at `http://hostname-login1` and then select the **Jupyter** icon.

This presents the Jupyter Notebook's login screen.

2. Enter LDAP credentials to log on to the Jupyter Notebook UI.

The system also ships with a default admin account with `crayadm` and `initial0` as the username and password respectively. If a new user needs to be assigned as an admin, add that user to `c.Authenticator.admin_users` in `/etc/jupyterhub/jupyterhub_config.py`. Users logging in with this default account will have the ability to control (start and stop) single notebook servers launched by end users.

Select **Help>User Interface Tour** menu For more information about the Jupyter Notebook web UI,

9.1 Create a Jupyter Notebook

Prerequisites

Before following this procedure, use the `urika-state` command to ensure that the Jupyter Notebook service is running.

About this task

This procedure can be considered as a Hello World example for using the Jupyter Notebook.

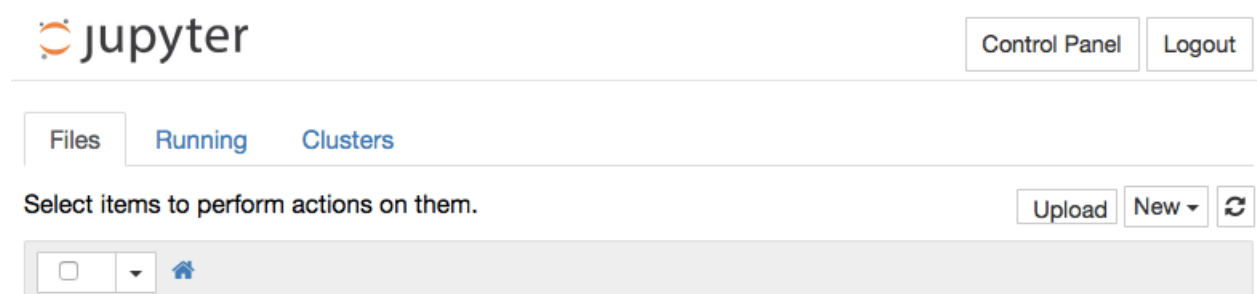


CAUTION: When using the Jupyter Notebook, the Spark Kernel configuration `PYTHONPATH` will have to be changed if the spark version changes. Python clusters are currently not supported on Urika-GX.

Procedure

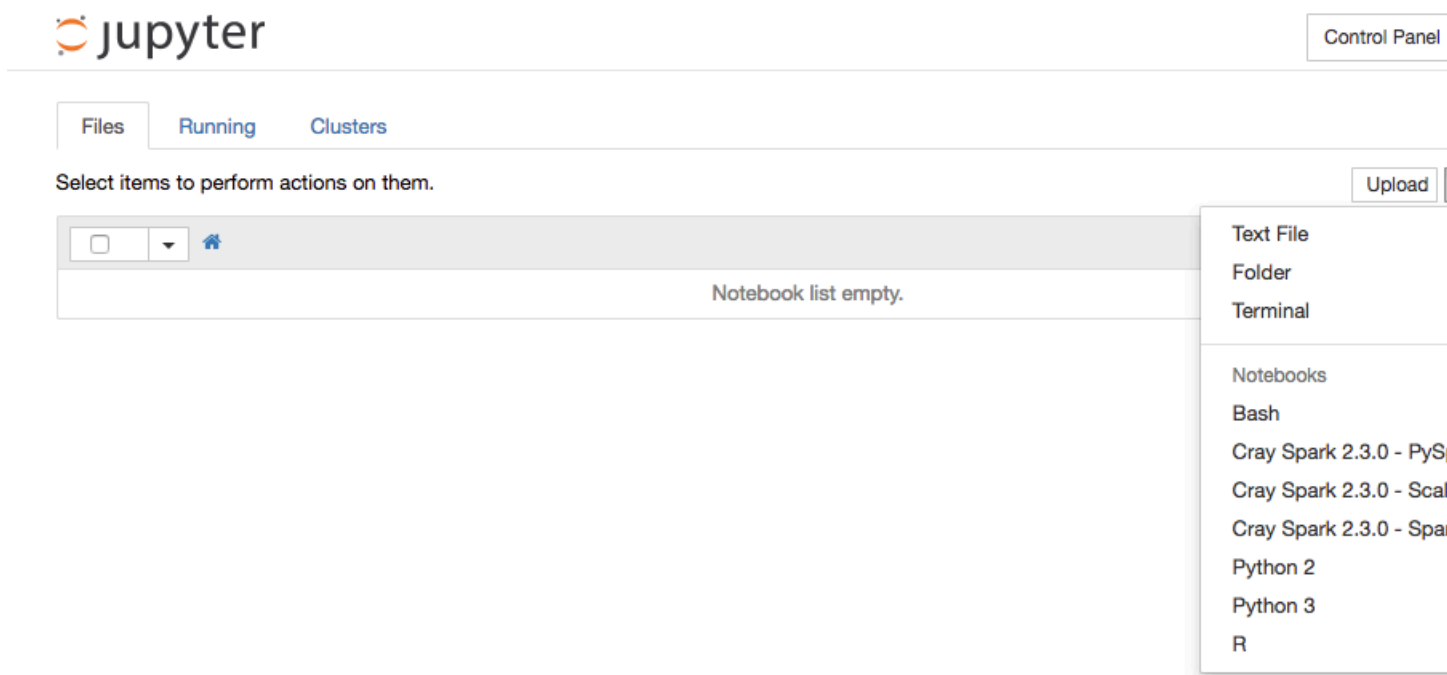
1. Access the Jupyter Notebook using the **Urika-GX Applications Interface** or via `http://hostname-login1:7800`. Access via the **Urika-GX Applications Interface** is recommended.

Figure 11. Jupiter Notebook Landing Page



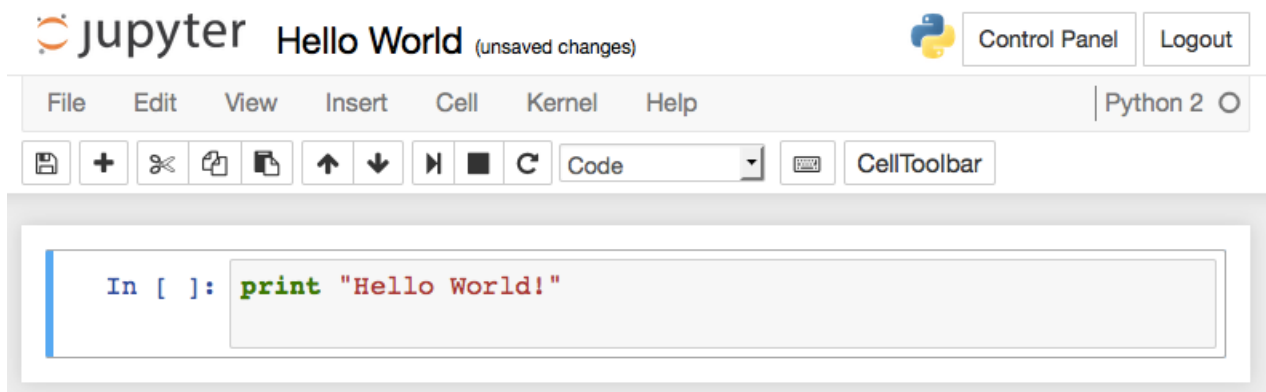
2. Select the desired type of notebook from the **New** drop down. In this example, Python2 is used.

Figure 12. Jupyter Notebook UI



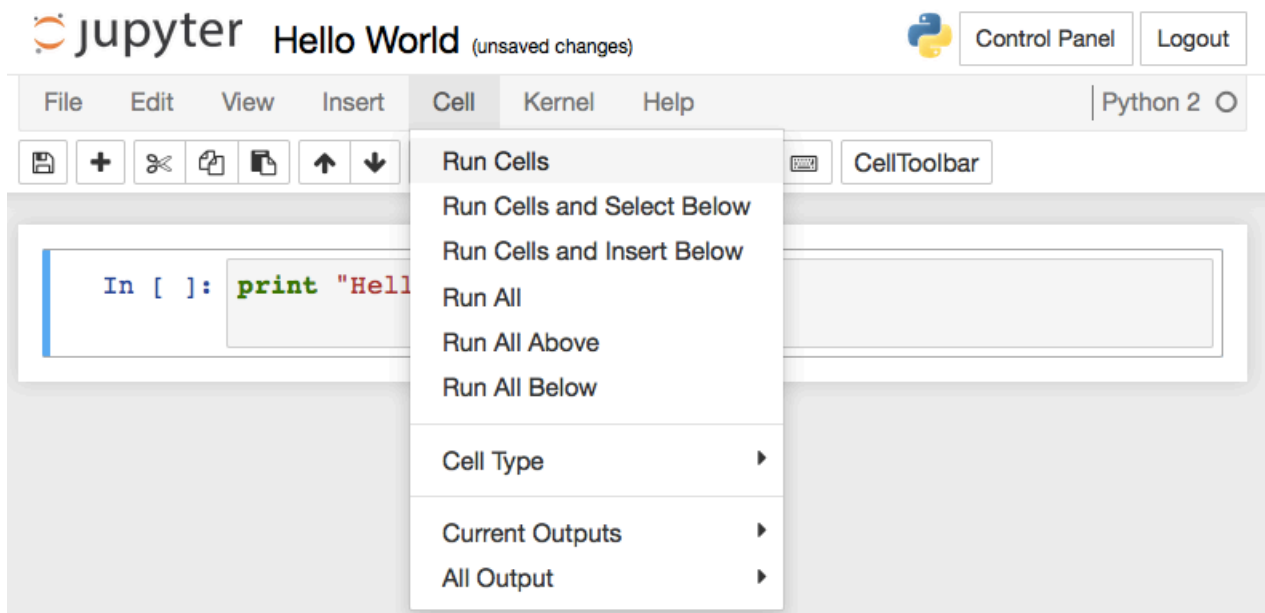
3. Specify a name for the new notebook by clicking on the default assigned name displayed at the top of the UI. For this example, 'Hello World' is used as the name for the notebook.
4. Enter Python code in the cell provided on the UI to display the text "Hello World".

Figure 13. Jupyter Notebook Enter Code UI



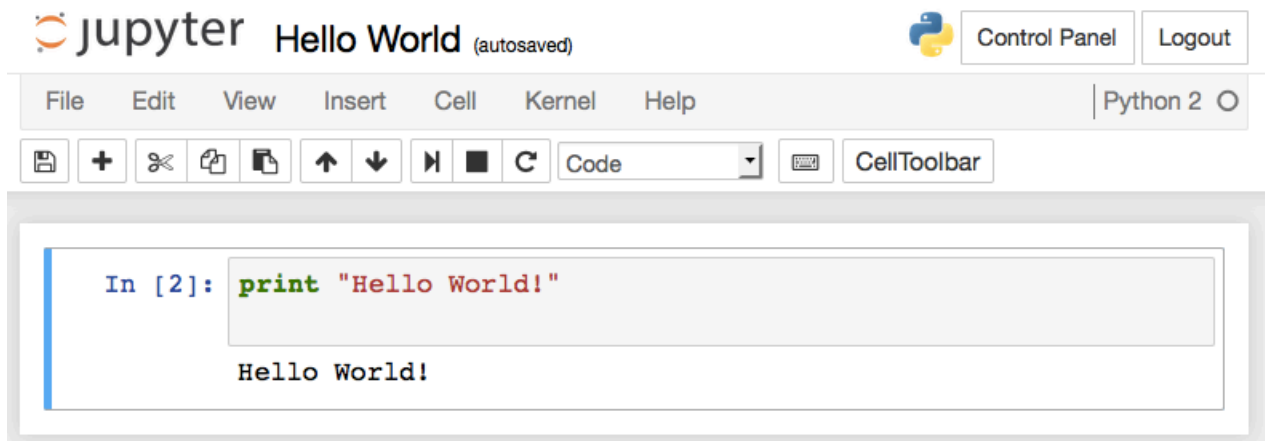
5. Select the **Run Cells** option from the **Cell** drop down

Figure 14. Select Run Cells



This displays the results of executing the code, as shown in the following figure:

Figure 15. Results



Add additional cells using the **Cell** drop down as needed. For additional information, select the **Help** drop down.

9.2 Share or Upload a Jupyter Notebook

Prerequisites

Before following this procedure, use the `urika-state` command to ensure that the Jupyter Notebook service is running.

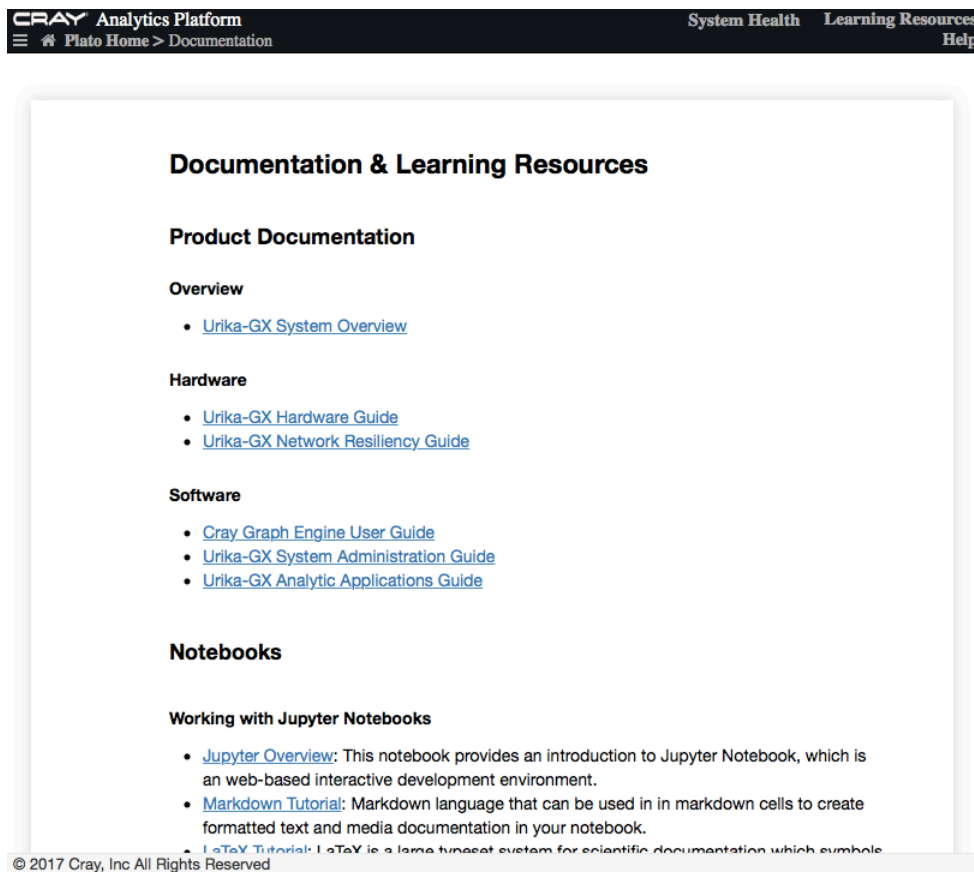
About this task

Currently, sharing URL links of Jupyter Notebooks is not directly supported. This procedure describes how to export a notebook and then share/upload it.

Procedure

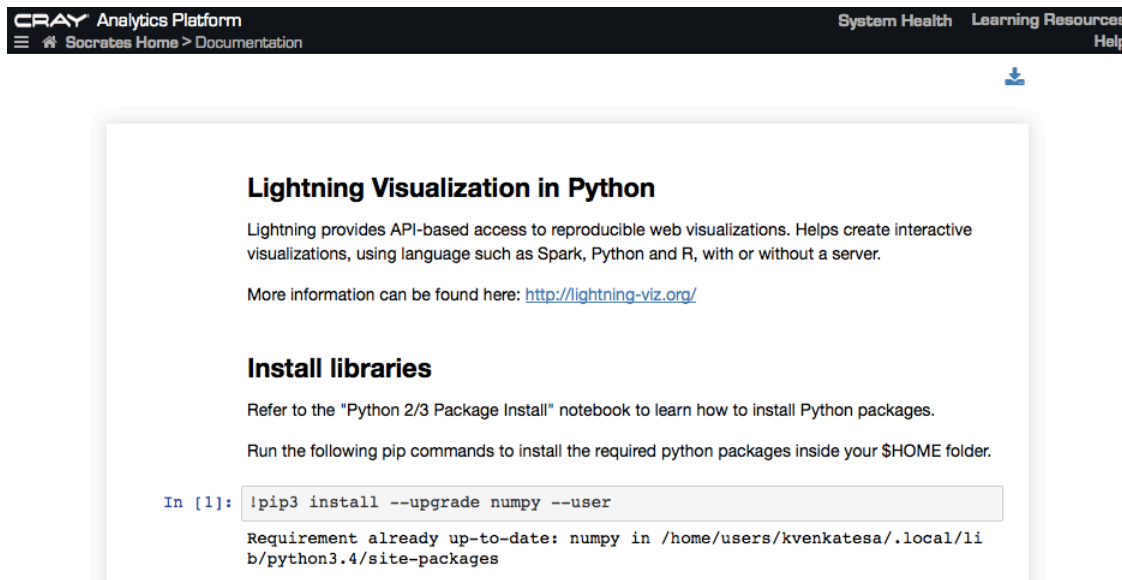
1. Select the **Learning Resources** link from the **Urika-GX Applications Interface**.
2. Select the notebook of choice from the list of Jupyter Notebooks listed in the **Notebooks** section of the **Documentation & Learning Resources** page.

Figure 16. Documentation & Learning Resources Page



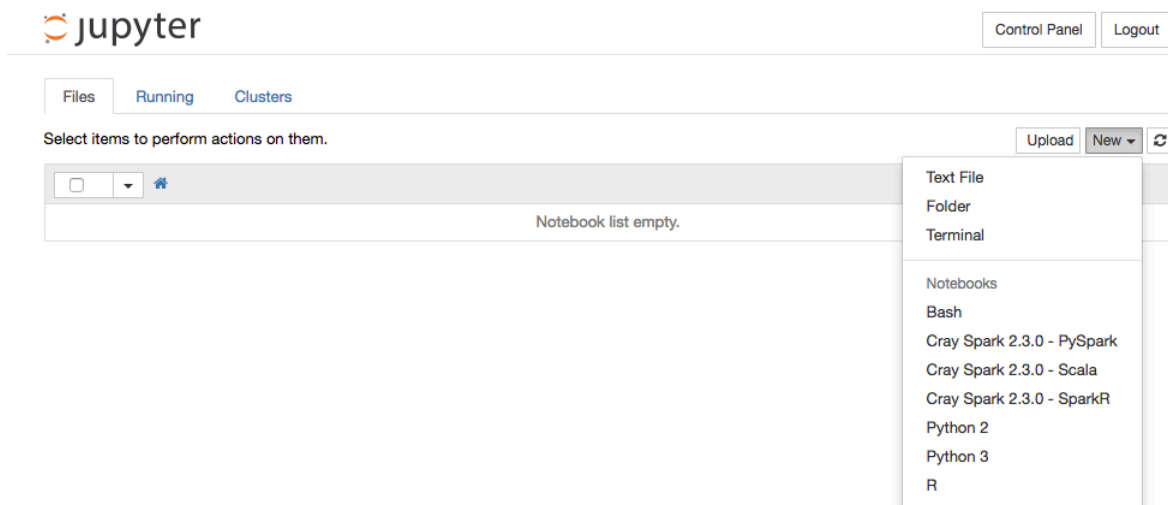
3. Select the blue download icon to download the notebook and save it to a location of choice.

Figure 17. Save Notebook



4. Select the home icon from the left of the UI to go back to the **Urika-GX Applications Interface** page and then select the icon for Jupyter Notebooks.
5. Upload the saved notebook to Jupyter using the **upload** button.

Figure 18. Select Jupyter Notebook from the Urika-GX Applications Interface



The uploaded notebook will appear in the list of notebooks on the **Running** tab on the interface

Stop any running notebooks before logging off.



CAUTION: If running notebooks are not stopped before logging off, they will continue running in the background, resulting in unnecessary resource utilization.

9.3 Create a Custom Python Based Kernel for JupyterHub

About this task

The following procedure can be used to create a custom Python based kernel for JupyterHub using any Anaconda environment a user wishes.

Procedure

1. Setup the environment on the command-line.

- a. Load the `anaconda` module.

```
$ module load anaconda3
```

- b. Create a Conda environment with the desired packages plus the `ipykernel` package.

```
$ conda create -n env ipykernel package-a package-b ...
```

Alternatively if the user has a pre-existing environment they can simply add `ipykernel` to it.

```
$ conda install -n env ipykernel
```

- c. Activate the environment.

```
$ source activate env
```

- d. Create a Jupyter kernel from the environment. The name specified via the `--name` flag must be unique amongst kernels created by the user, otherwise any pre-existing kernel of the same name will be overwritten

```
$ python -m ipykernel install --user --name env --display-name "Python (env)"
```

2. Go to JupyterHub and either create a new Notebook using the newly created kernel, or go to **Kernel > Change Kernel** within an existing Notebook to select the newly created kernel.

The kernel will be listed under the value passed to `--display-name` in the above steps. If the newly created kernel is not immediately visible, the user will need to first refresh their browser window.

If it is found that additional packages are required during the course of notebook development, add those packages by using `Conda install` to update the environment at the command-line. After doing so, go to **Kernel > Restart Kernel** in the notebook to pick up the updated environment. For proper display of notebook widgets, users may also need to add additional packages to their Conda environments, such as `ipywidgets`

10 Get Started with Using Grafana

Grafana is a feature-rich metrics, dashboard, and graph editor. It provides information about utilization of system resources, such as CPU, memory, I/O, etc.

Major Grafana components and UI elements include:

- **Dashboard** - The **Dashboard** consolidates all the visualizations into a single interface. Urika-GX ships with pre-defined dashboards for resource utilization information. For information about these dashboards, see [Default Grafana Dashboards](#) on page 99.
- **Panels** - The **Panel** is the basic visualization building block of the GUI. Panels support a wide variety of formatting options and can be dragged, dropped, resized, and rearranged on the **Dashboard**.
- **Query Editor** - Each Panel provides a **Query Editor** for the data source, which is InfluxDB on the Urika-GX system. Information retrieved from the **Query Editor** is displayed on the **Panel**.
- **Data Source** - Grafana supports many different storage back-ends for retrieving time series data. Each **Data Source** has a specific **Query Editor** that is customized for the features and capabilities that the particular **Data Source** exposes. Urika-GX currently supports InfluxDB as its data source.
- **Organization** - Grafana supports multiple organizations in order to support a wide variety of deployment models. Each **Organization** can have one or more **Data Sources**. All Dashboards are owned by a particular Organization.
- **User** - A **User** is a named account in Grafana. A user can belong to one or more **Organizations**, and can be assigned different levels of privileges via roles.
- **Row** - A **Row** is a logical divider within a **Dashboard**, and is used to group **Panels** together.

Roles

Users and **Organizations** can be assigned roles to specify permissions/privileges. These roles and their privileges are listed in the following table:

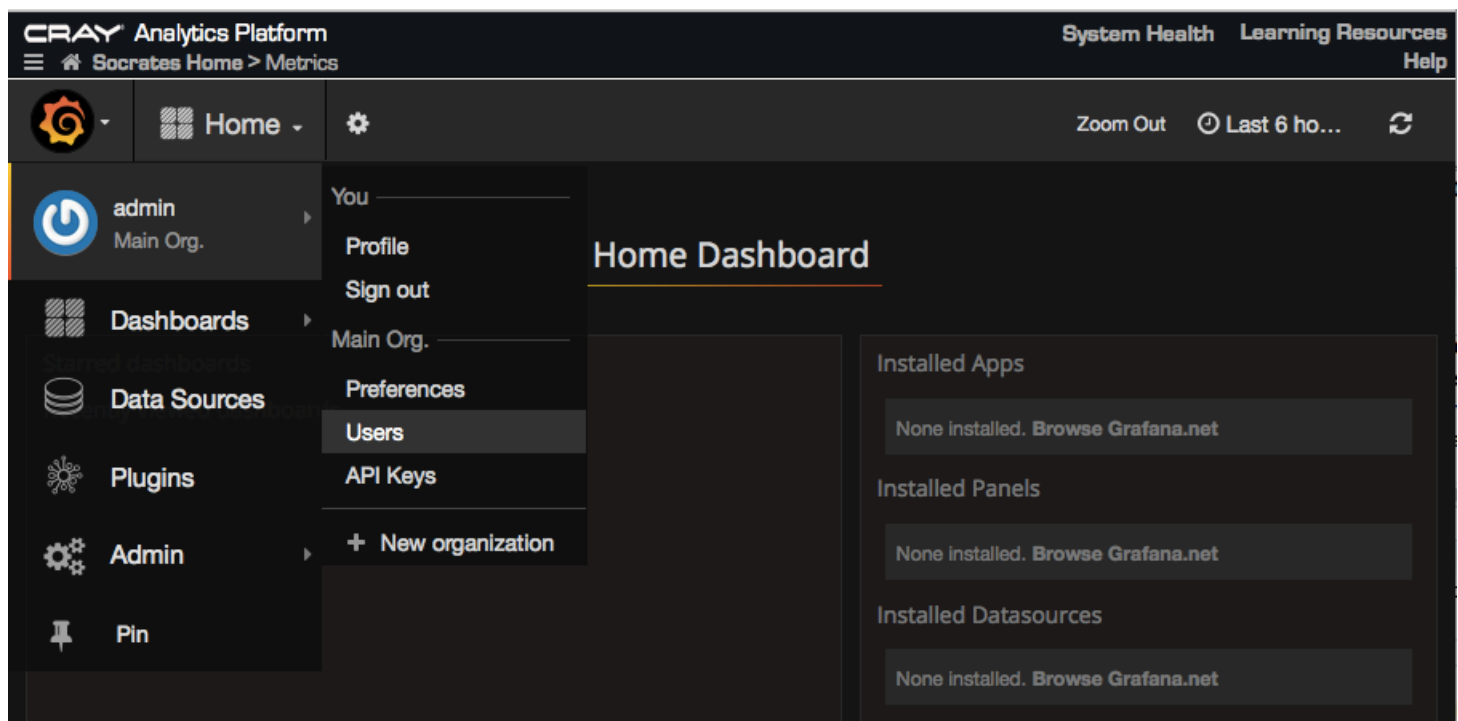
Table 6. Grafana Roles and Permissions

Role	Edit roles	View graphs	Edit/create copy of existing graphs	Add new graphs to existing dashboards	Create new/import existing dashboards
Admin	Yes	Yes	Yes	Yes	Yes (These dashboards persist between sessions)
Viewer (default role)	No	Yes	No	No	No

Role	Edit roles	View graphs	Edit/create copy of existing graphs	Add new graphs to existing dashboards	Create new/import existing dashboards
Editor	No	Yes	Yes	Yes	Yes (These dashboards get deleted when the editor logs out)
Read only editor	No	Yes	Yes	Yes	No

Each role type can also export performance data via the dashboard. When a new user signs up for the first time, the default role assigned to the user is that of a **Viewer**. The admin can then change the new user's role if needed. All users have a default role of a **Viewer** when they first log on to Grafana. Administrators can change roles assigned to users as needed using the **Admin** menu.

Figure 19. Change User Roles Using the Admin Menu



Since the time displayed on the Grafana UI uses the browser's timezone and that displayed on the Spark History server's UI uses the Urika-GX system's timezone, the timezones displayed on the two UIs may not be the same.

By default, Grafana's refresh rate is turned off on the Urika-GX system. Sometimes the Hadoop and Spark Grafana dashboards take longer to load than expected. The usual cause of this issue is the time it takes InfluxDB to return all of the requested data. To reduce the time for the Hadoop and Spark dashboards to display data, refer to [Update the InfluxDB Data Retention Policy](#) on page 62. Reducing the amount of data retained makes Grafana dashboards display faster.

10.1 Urika-GX Performance Analysis Tools

Performance analysis tools installed on the Urika-GX system can be broadly categorized as:

- Performance analysis tools for monitoring system resources. Urika-GX uses Grafana for monitoring system resource utilization.
- Debugging tools. The Spark Shell can be used for debugging Spark applications.
- Profiling tools - The Spark History Server can be used for profiling Spark applications. The Spark History Server contains custom Cray enhancements that link Spark tasks in the UIs to Grafana dashboards that display compute node system metrics during the tasks' executions. These can be accessed by clicking links in the executor ID/host column in the tasks table of the stage pages, or by selecting the compare checkboxes of multiple tasks in the task table and clicking the compare link at the top of the table.

10.2 Update the InfluxDB Data Retention Policy

Prerequisites

This procedure requires root privileges. Before performing this procedure, use the `urika-state` command to ensure that the system is operating in the service mode that supports using InfluxDB. For more information, see the `urika-state` man page and refer to [Urika-GX Service Modes](#) on page 7.

About this task

The data retention policy for InfluxDB defaults to infinite, i.e. data is never deleted. As a result, Spark and Hadoop Grafana dashboards may take longer to load and InfluxDB may take up more space than necessary. To reduce the amount of space being used by InfluxDB, the data retention policy for each database needs to be reduced, as described in this procedure. Reducing the data retention policy for Spark and Hadoop databases can reduce the load time of the Spark and Hadoop Grafana dashboards.

Procedure

1. Log on to login2 and become root.
2. Switch to the `/var/lib/influxdb/data` directory.

```
# cd /var/lib/influxdb/data
```

3. Use the `du` command to show how much space being used.

The sizes below are shown as examples. Actual sizes on the system may vary.

```
$ du -sh *
14G    Cray Urika GX
1.5G   CrayUrikaGXHadoop
906M   CrayUrikaGXSpark
21M    _internal
#
```

4. Connect to InfluxDB to view the current data retention policy.

```
# /bin/influx
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server
management, and monitoring.
Connected to http://localhost:8086 version 0.12.2
InfluxDB shell 0.12.2
> show retention policies on "Cray Urika GX"
name      duration      shardGroupDuration  replicaN  default
default    0              168h0m0s           1         true
```

5. Update the data retention policy according to requirements.

In this example the data retention duration is changed from 0 (forever) to 2 weeks (504 hours).

```
> alter retention policy default on "Cray Urika GX" Duration 2w
> show retention policies on "Cray Urika GX"
name      duration      shardGroupDuration  replicaN  default
default    504h0m0s       24h0m0s            1         true
> exit
```

The change will take a while to be applied. The default is 30 minutes.

6. Verify that the data retention change has taken effect

```
# du -sh *
3G      Cray Urika GX
1.5G     CrayUrikaGXHadoop
906M     CrayUrikaGXSpark
21M      _internal
```

11 Use Docker on Urika-GX

Docker enables packaging applications with their dependencies without having to worry about system configurations, thus making applications more portable.

Urika-GX features the infrastructure for running Docker containers (orchestrated by Marathon and Mesos) pulled from the public Docker Hub. Docker is disabled on Urika-GX by default. Use the `urika-state` command to check if the Docker service is running. Launching of Docker containers using Docker commands is not supported on Urika-GX. Use the Marathon Web UI for launching containers.

For more information, visit <https://www.docker.com>.

Examples

Control Docker

Request an administrator to execute the use `urika-start -s docker`, `urika-stop -s docker` and `urika-state` commands to start, stop and view the status of Docker, respectively.

Launch Containers

Containers are started like any typical Marathon application by entering the configuration into the Marathon web UI, or by using the `curl` command:

```
curl -X POST
      "http://hostname-login1:8080/v2/apps" -d @"$file" -H "Content-
type:
      application/json"
```

Considerations

- Resource constraints passed to Docker from Marathon/Mesos:
 - The `cpus` parameter is not a direct limitation on the number of CPUs available to a Docker container. It is not a limit on the speed of the CPUs. Docker containers retain access to all host CPUs.
 - Mesos builds a Docker run command, converting the `cpus` value into a value for Docker's `--cpu-shares` setting, which sets priority weight for that process relative to all others on the machine. An application run with `cpus=2` should receive twice the priority as one using `cpus=1`.
 - Mesos keeps track of total CPU resources, and subtracts the specified CPUs in the container definition from the total number available on the node, despite the container not being strictly limited to specific CPUs.
- Commands vs arguments - Marathon supports an `args` field in the JSON application. It is invalid to supply both `cmd` and `args` for the same application. The behavior of `cmd` is the value is wrapped by Mesos

via `/bin/sh -c '${app.cmd}'`. The `args` mode of specifying a command allows for safe usage of containerizer features like custom Docker entry points.

- Constraints - `"constraints": [{"hostname", "CLUSTER", "machine-nid00024.us.cray.com"}]`.

Refer to <https://github.com/mesosphere/marathon/blob/master/docs/docs/constraints.md#cluster-operator> for details about Marathon constraints.

- Privileged Mode and Arbitrary Docker Options - Marathon supports two keys for Docker containers: `privileged` and `parameters`. The `privileged` flag allows users to run containers in privileged mode. This flag is set to `false` by default. The `parameters` object allows users to supply arbitrary command-line options for the `docker run` command executed by the Mesos containerizer.



CAUTION: Any parameters passed in this manner are not guaranteed to be supported in the future, as Mesos may not always interact with Docker via the CLI.

```
"privileged": true,
  "parameters": [
    { "key": "hostname", "value": "a.corp.org" },
    { "key": "volumes-from", "value": "another-container" },
    { "key": "lxc-conf", "value": "..." }
  ]
```

11.1 Image Management with Docker and Kubernetes

About Docker

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security enables running many containers simultaneously on a given host. Because of the lightweight nature of containers, users can run more containers on a given hardware combination than if using virtual machines.

Images shipped with the system are managed by Docker when Urika-GX is operating under the default service mode. The SMW hosts Urika-GX's container repository, which is a container itself and can only host Cray developed containers currently.

NOTE: Building and managing new Docker images is currently not supported on Urika-GX.

For more information, visit <https://www.docker.com>.

About Kubernetes

Kubernetes is used for automating deployment, scaling and management of containerized applications. It groups containers that make up an application into logical units for easy management and discovery. It performs container management tasks, such as, running containers across many different machines, scaling up or down by adding or removing containers when demand changes, keeping storage consistent with multiple instances of an application, distributing load between the containers and launching new containers on different machines if something fails.

On Urika-GX, Kubernetes is used to manage containers in the secure service mode.

NOTE: Currently, Kubernetes supports only Spark images on Urika-GX.

For more information, visit <https://kubernetes.io/>.

About the Cray Spark Image

In order to run Spark on Kubernetes, Urika-GX ships with customized Spark images, which are based on the Spark version used on the system.

11.2 Run the Native Docker Engine on Marathon

Prerequisites

- The Docker engine needs to be running on the login and compute nodes. Use the `urika-state` command to check the status of Docker.
- `mesos-agent` needs to be configured to recognize the Docker container type.

About this task

This procedure describes how to enable the running and orchestration of Docker containers via Marathon and Mesos to facilitate rapid dynamic application development and service provisioning across the cluster.

Procedure

Define a container for use with Marathon.

Defining a container for use with Marathon is very similar to provisioning an application. The following information is provided to the Marathon UI via the Web UI or `curl` command.

```
{
  "id": "docker-demo",
  "cmd": "python3 -m http.server 8080",
  "cpus": 0.5,
  "mem": 1024.0,
  "instances": 1,
  "constraints": [["hostname", "CLUSTER", "socrates-nid00024.us.cray.com"]],
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "python:3",
      "network": "BRIDGE",
      "portMappings": [
        { "containerPort": 8080, "hostPort": 0, "servicePort": 9000,
"protocol": "tcp" },
        { "containerPort": 161, "hostPort": 0, "protocol": "udp" }
      ]
    },
    "volumes": [
      {
        "containerPath": "/mnt/lustre",
        "hostPath": "/mnt/lustre/<path>",
        "mode": "RW"
      }
    ]
  }
}
```

```

    }
  ],
  "healthChecks": [
    {
      "protocol": "HTTP",
      "portIndex": 0,
      "path": "/",
      "gracePeriodSeconds": 5,
      "intervalSeconds": 20,
      "maxConsecutiveFailures": 3
    }
  ]
}

```

In the preceding code block:

- `servicePort` is a helper port intended for performing service discovery using a well-known port per service. The assigned `servicePort` value is not used/interpreted by Marathon itself, but is supposed to be used by the load balancer infrastructure.
- The `servicePort` parameter is optional and defaults to 0.

A random port will be assigned if the value of `servicePort` is 0. If Marathon assigns a `servicePort` value, the value of `servicePort` will be unique across the cluster. The values for random service ports fall in the `[local_port_min, local_port_max]` range, where `local_port_min` and `local_port_max` are command-line options with default values of 10000 and 20000, respectively.

- The `protocol` parameter is optional and defaults to `tcp`. Possible values for this parameter include `tcp` and `udp`.

The preceding code block performs the following:

- Creates a container named `docker-demo`.
- Pulls the `python` image with a version specification of 3 from https://hub.docker.com/_/python/
- Sets constraints on the CPU and memory allocated for use by the container
- Runs a command in the container to start a simple HTTP server on port 8080
- Creates a network bridge for the container. This involves mapping:
 - the container port 8080 to a Marathon-assigned port on the host interfaces
 - the container port 9000 to host interfaces
 - the container port 161:udp to a Marathon-assigned port on the host interfaces
- Mounts the host filesystem location `/mnt/lustre/path` to the container filesystem location to `/mnt/lustre`, so it can be accessed by the container.
- Creates a basic Marathon HTTP health check.

For more information, visit <https://www.docker.com>.

12 Start Individual Kafka Brokers

About this task

Use the following instructions to start Kafka brokers.

Procedure

1. Log on to a login node.
2. Copy the default Kafka configuration file located at `/usr/hdp/current/kafka-broker/config/server.properties` to a local directory.
3. Update the configuration `log.dirs` parameter from `log.dirs=/tmp/kafka-logs` to the home directory.
To prevent log files from being locked down by users, the `log.dirs` file needs to be updated, as described in this procedure.
4. Redirect the Kafka server logs from `/var/log/kafka` directory.

Only the user `Kafka` has permissions to write to that directory, so starting as a non-Kafka or non-root user results in none of the server logs being recorded. So either the user can choose to not care about that, or before starting their Kafka broker overwrite `LOG_DIR`.

```
$ export LOG_DIR = ~/kafka
```

5. Start the Kafka broker.

```
$ /usr/hdp/current/kafka-broker/bin/kafka-server-start.sh ~/server.properties
```

13 Overview of the Cray Application Management UI

The **Cray Application Management UI** is shown in the following figure:

Figure 20. Cray Application Management UI

The screenshot displays the Cray Analytics Platform interface for Application Management. At the top, there's a navigation bar with 'Socrates Home > Application Management' and links for 'System Health' and 'Learning Res'. Below this is the 'CRAY APPLICATION MANAGEMENT' header. The main section is titled 'Job List' and features a table of jobs. A 'Filter by Type' modal is open, showing checkboxes for SPARK, YARN, CGE, MRUN, and OTHER. The table has columns for Job Id, Metrics, Job Name, Start Time, and Status. The first three rows of the table are visible, showing various job IDs and their statuses (FINISHED).

Job Id	Metrics	Job Name	Start Time	Status
rpnfile-2016-297-14-06-29.772470-ripple.cge.mrun	-	rpnfile-2016-297-14-06-29.772470-ripple.cge.mrun	2016-10-23 07:07:10	FINISHED
port.1025-2016-297-12-53-05.500114-ripple.cge.mrun	-	port.1025-2016-297-12-53-05.500114-ripple.cge.mrun	2016-10-23 05:55:59	FINISHED
rpnfile-2016-297-12-27-59.752131-ripple.cge.mrun	-	rpnfile-2016-297-12-27-59.752131-ripple.cge.mrun	2016-10-23 05:27:49	FINISHED

The **Search** field and **Quick Filters** drop down facilitate searching and filtering submitted jobs, based on the specified criteria. When these UI elements are used, the results are displayed in a table and the specified search/filter criteria is displayed at the top of these UI elements. A list of active jobs is displayed when no search/filtering criteria has been specified. By default, this shows jobs that were started in the last 24 hours, jobs that ended in the last 24 hours and jobs that are still running. Click on the text **active** to see a detailed description of functionality. If both the **Search** and **Quick Filters** UI elements are used at the same time, only jobs that match the selected quick filter will be displayed.

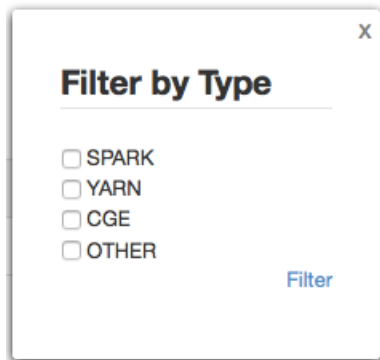
The table displayed on the UI contains information about submitted jobs and contains the following columns:

- **Job Id** - Displays the job ID for each submitted job. Selecting a job ID opens up another tab, which displays additional details of the job. For example, if a job ID for a Spark job is selected, a separate tab will be opened, displaying the Spark History Server.

ATTENTION: Selecting a job ID for a job having "OTHER", "MRUN" or "CGE" as the job type will open up the Mesos UI in a separate tab.

- **Metrics** - Displays links that can be used for displaying the graphical representation of the job's metrics on the Grafana UI, which opens up in a separate browser tab.
- **Job Name** - Displays the name of the submitted job.
- **Type** - Displays types of all the submitted jobs. Jobs can be filtered based on type using the filter icon provided on the UI.

Figure 21. Filtering by Type



- **User**- Displays the name of the user who submitted the job.
- **Start Time** - Displays the time the job started executing. Jobs can be filtered based on starting time using the filter icon provided on the UI.
- **End Time** - Displays the time the job finished executing. This column will be empty for jobs that have not finished executed yet.
- **Status** - Displays the current status of the job, which depends on the job's underlying framework.

To filter a job based on its status, click the filter icon in this column's header.

Make selections as needed and then select the **Filter** button on the pop-up. Click on the text **Failed** on the **Status** column to view logs for debugging failed jobs. For Spark jobs, this column contains a link titled **Finished**, which can be used to view and download logs that help identify whether or not the Spark job succeeded. Selecting this link will present a login screen, where users will need to enter their LDAP credentials.

IMPORTANT: If the user logged in with the default user account (having `admin/admin` as the username and password), the system will require the user to log in again with their LDAP or system credentials to view Spark executor logs.

- **Action** - The kill button displayed in this column enables killing a running job. This column will be empty for jobs that have finished executing. Users can only kill jobs submitted by themselves. However, the system administrator can delete any job.

NOTE: If the user logged in with the default user account (having `admin/admin` as the username and password), the system will require the user to log in again with their LDAP or system credentials to kill jobs of type "CGE" or "MRUN".

14 Update the InfluxDB Data Retention Policy

Prerequisites

This procedure requires root privileges. Before performing this procedure, use the `urika-state` command to ensure that the system is operating in the service mode that supports using InfluxDB. For more information, see the `urika-state` man page and refer to [Urika-GX Service Modes](#) on page 7.

About this task

The data retention policy for InfluxDB defaults to infinite, i.e. data is never deleted. As a result, Spark and Hadoop Grafana dashboards may take longer to load and InfluxDB may take up more space than necessary. To reduce the amount of space being used by InfluxDB, the data retention policy for each database needs to be reduced, as described in this procedure. Reducing the data retention policy for Spark and Hadoop databases can reduce the load time of the Spark and Hadoop Grafana dashboards.

Procedure

1. Log on to login2 and become root.
2. Switch to the `/var/lib/influxdb/data` directory.

```
# cd /var/lib/influxdb/data
```

3. Use the `du` command to show how much space being used.

The sizes below are shown as examples. Actual sizes on the system may vary.

```
$ du -sh *
14G    Cray Urika GX
1.5G    CrayUrikaGXHadoop
906M    CrayUrikaGXSpark
21M    _internal
#
```

4. Connect to InfluxDB to view the current data retention policy.

```
# /bin/influx
Visit https://enterprise.influxdata.com to register for updates, InfluxDB server
management, and monitoring.
Connected to http://localhost:8086 version 0.12.2
InfluxDB shell 0.12.2
> show retention policies on "Cray Urika GX"
name      duration  shardGroupDuration  replicaN  default
default    0         168h0m0s           1         true
```

5. Update the data retention policy according to requirements.

In this example the data retention duration is changed from 0 (forever) to 2 weeks (504 hours).

```
> alter retention policy default on "Cray Urika GX" Duration 2w
> show retention policies on "Cray Urika GX"
name      duration      shardGroupDuration  replicaN  default
default    504h0m0s      24h0m0s              1         true
> exit
```

The change will take a while to be applied. The default is 30 minutes.

6. Verify that the data retention change has taken effect

```
# du -sh *
3G    Cray Urika GX
1.5G   CrayUrikaGXHadoop
906M   CrayUrikaGXSpark
21M    _internal
```


15 Manage the Spark Thrift Server as a Non-Admin User

Prerequisites

- Ensure that the system is running in the service mode that allows use of the Spark Thrift Server. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- This procedure requires Tableau Desktop (version 10.2) and Simba Spark ODBC driver to be installed on the client machine:
- If using a MAC, the following procedure requires version 10.11 of the operating system.

About this task

Cray recommends to have the Spark Thrift to be started up by administrators, however, users can use the following instructions if they need to start up their own Spark Thrift server.



CAUTION: It is recommended for multiple users (admin and non-admin) to use the same Spark Thrift server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown. In addition, though it is possible for multiple users to connect to each other's Spark Thrift server, doing so can result in loss of connectivity if the server is brought down by the user who brings it up. If a user who starts up the Spark Thrift server brings it down, other users may experience loss of connection issues.

Procedure

1. Copy the `spark-env.sh`, `spark-defaults.conf` and `hive-site.xml` files to the local `$HOME` directory.

```
$ cp /opt/cray/spark2/default/conf/spark-env.sh $HOME
$ cp /opt/cray/spark2/default/conf/spark-defaults.conf $HOME
$ cp /opt/cray/spark2/default/conf/hive-site.xml $HOME
```

2. Modify the `hive-site.xml` configuration file to set `hive.server2.thrift.port` to a non-conflicting port.
3. Increase the number of compute nodes if needed by editing the `spark-defaults.conf` configuration file and changing the default value of `spark.cores.max` from 32 to the desired value.
4. Execute the `start-thriftserver.sh` script.

```
$ /opt/cray/spark2/default/sbin/start-thriftserver.sh
```

5. Stop the Spark Thrift server when finished using it.

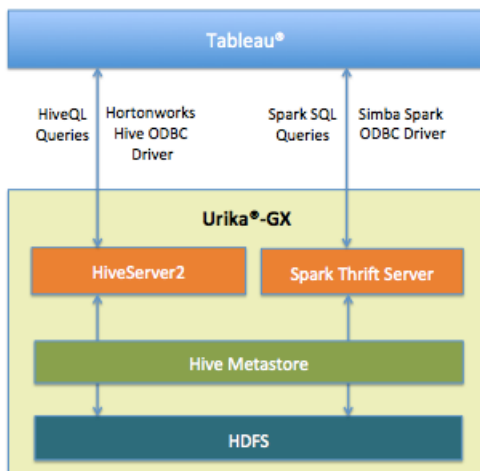
```
$ /opt/cray/spark2/default/sbin/stop-thriftserver.sh
```

16 Use Tableau® with Urika-GX

Tableau is a desktop application running on either Windows or Mac. Urika-GX features connectivity to Tableau® for data visualization. Tableau connects easily to Urika-GX, as well as to other data sources. It enables viewing data as visually appealing visualizations called dashboards, quickly and easily.

The following figure depicts how Tableau connects to Urika-GX via the Hive and SparkSQL Thrift servers:

Figure 22. Urika-GX connectivity to Tableau



16.1 Connect Tableau to HiveServer2 Using LDAP

Prerequisites

- Ensure that the system is running in the service mode that allows use of Tableau and HiveServer2. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- Ensure that LDAP authentication for Tableau to HiveServer2 is enabled. For more information, refer to section 'Enable LDAP for Connecting Tableau to HiverServer2 of the *'Urika®-GX System Administration Guide'*.
- This procedure requires the following software to be installed on the external client machine connected to Urika-GX:
 - Tableau Desktop (version 10.2)
 - Hortonworks Hive ODBC driver.

- If using a Mac, the following procedure requires version 10.11 or later of the OS X operating system.
- Request an administrator to ensure that the Hive service is up and running.

About this task



CAUTION: It is recommended for multiple users to use the same Hive server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown.

Procedure

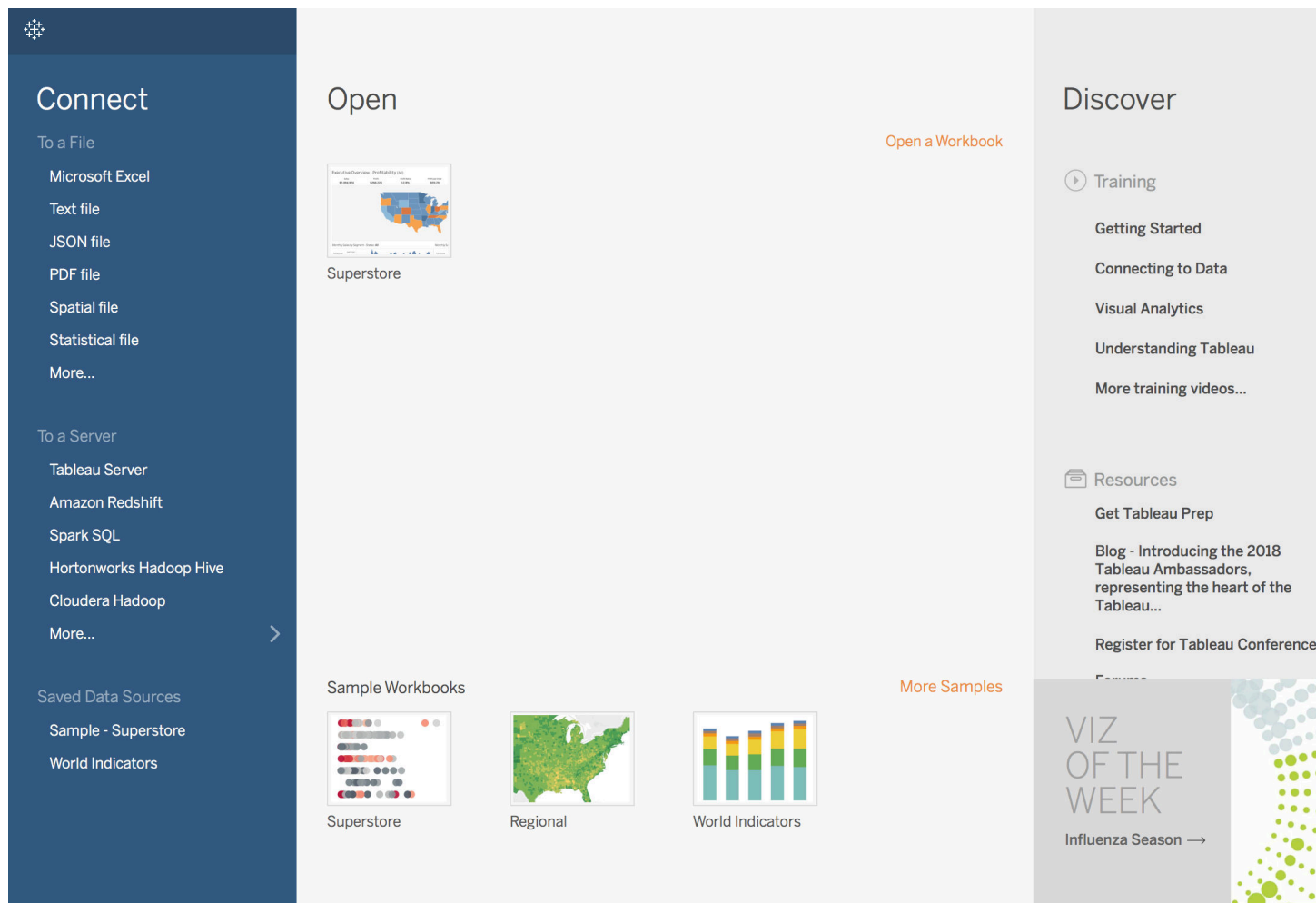
1. Log on to a Urika-GX system's login node.
2. Request an administrator to flex up the YARN cluster.

NOTE: Cray recommends that YARN clusters for Tableau connectivity be flexed up only by administrators on Urika-GX. Administrators should use the `urika-yam-flexup` command and specify a timeout value of 0 when using this procedure. For more information, administrators should see the `urika-yam-flexup` man page or refer to the section titled '*Flex up a YARN sub-cluster on Urika-GX*' of the '*Urika®-GX Analytic Applications Guide*'.

3. Launch the Tableau application on a client machine.

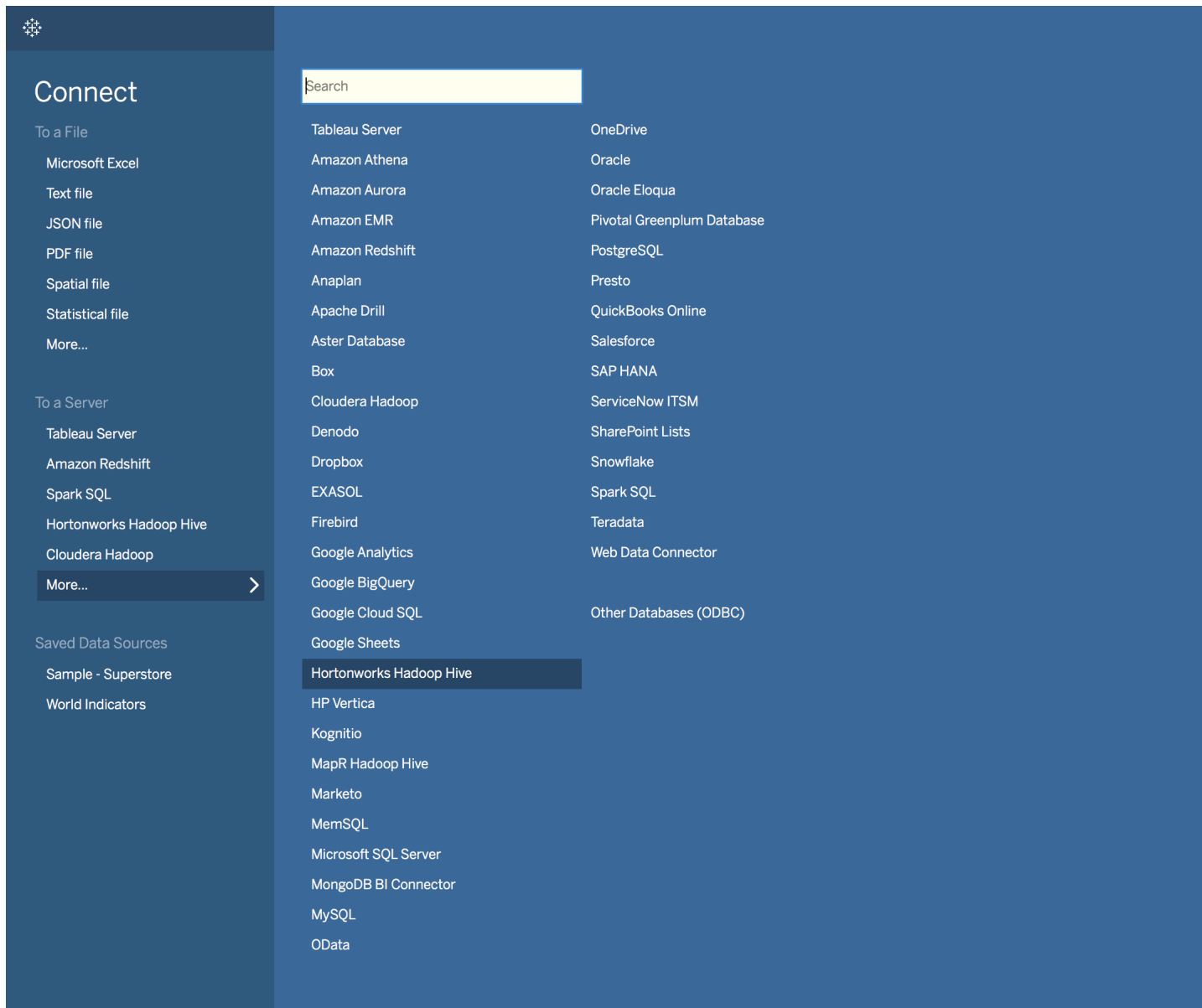
This will start the Tableau application and bring up the Tableau UI on the user's desktop.

Figure 23. Tableau UI



4. Navigate to **Connect > To a Server > More**
5. Select **Hortonworks Hadoop Hive** from the list of servers.

Figure 24. Selecting Hortonworks Hadoop Hive Server



6. Populate the server connection pop up.

- Enter `hostname-login1` in the **Server** field, where `hostname` is used as an example for the name of the machine and should be replaced with the actual machine name when following this step.
- Enter `10000` in the **Port** field.
- Select **HiveServer2** from the **Type** drop down.
- Select **User Name** from the **Authentication** drop down.
- Enter values in the **Username** and **Password** fields.
- Select the **Sign In** button.

Figure 25. Connect HiveServer2 to Tableau

Hortonworks Hadoop Hive

Server: Port:

Enter information to sign in to the server:

Type:

Authentication:

Transport:

Username:

☐ Require SSL

[Initial SQL...](#) [Sign In](#)

7. Perform data visualization/exploration tasks as needed using Tableau.
8. Request an administrator to flex down the YARN cluster.

NOTE: Cray recommends that YARN clusters for Tableau connectivity be flexed down only by administrators on Urika-GX. For more information, administrators should see the `urika-yam-flexdown` man page or refer to the section titled 'Flex up a YARN sub-cluster on Urika-GX' of the 'Urika®-GX Analytic Applications Guide'.

16.2 Connect Tableau to HiveServer2 Securely

Prerequisites

- Ensure that the system is running in the service mode that allows use of Tableau and HiveServer2. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- This procedure requires the following software to be installed on the client machine:
 - Tableau Desktop (version 10.2)
 - Hortonworks Hive ODBC driver.
- If using a Mac, the following procedure requires version 10.11 of the OS X operating system.
- Request an administrator to ensure that the Hive service is up and running.

About this task



CAUTION: It is recommended for multiple users to use the same Hive server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown.

Procedure

1. Request an administrator to enable SSL.

Administrators should follow instructions listed in section *'Enable SSL on Urika-GX'* of the *'Urika-GX System Administration Guide'* to enable SSL. To connect to the HiveServer2 from Tableau, the `ServerCertificate.crt` SSL certificate must be present on the machine running Tableau and needs to be added to Tableau.

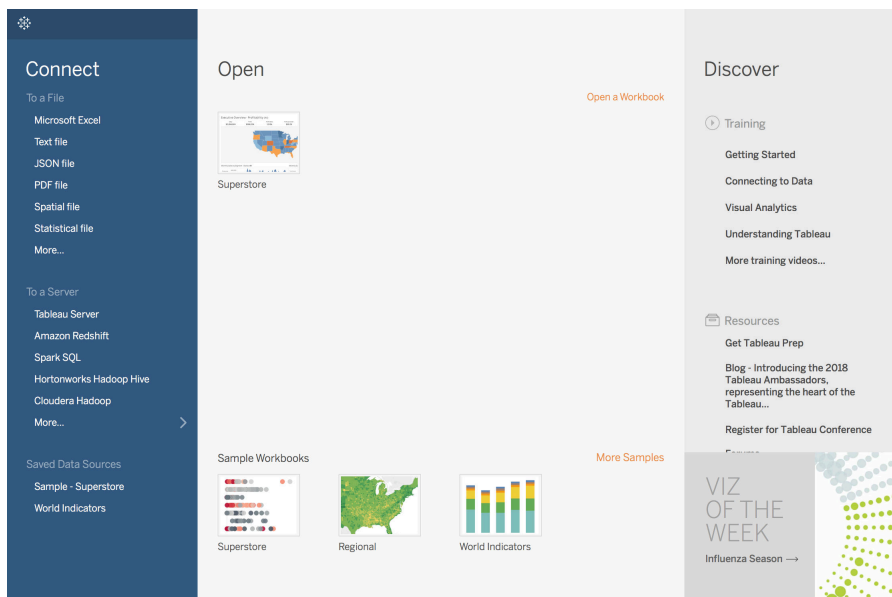
2. Log on to a Urika-GX system's login node.
3. Request an administrator to flex up the YARN cluster.

NOTE: Cray recommends that YARN clusters for Tableau connectivity be flexed up only by administrators on Urika-GX. Administrators should use the `urika-yam-flexup` command and specify a timeout value of 0 when using this procedure. For more information, administrators should see the `urika-yam-flexup` man page or refer to the section titled *'Flex up a YARN sub-cluster on Urika-GX'* of the *'Urika®-GX Analytic Applications Guide'*.

4. Launch the Tableau application on a client machine.

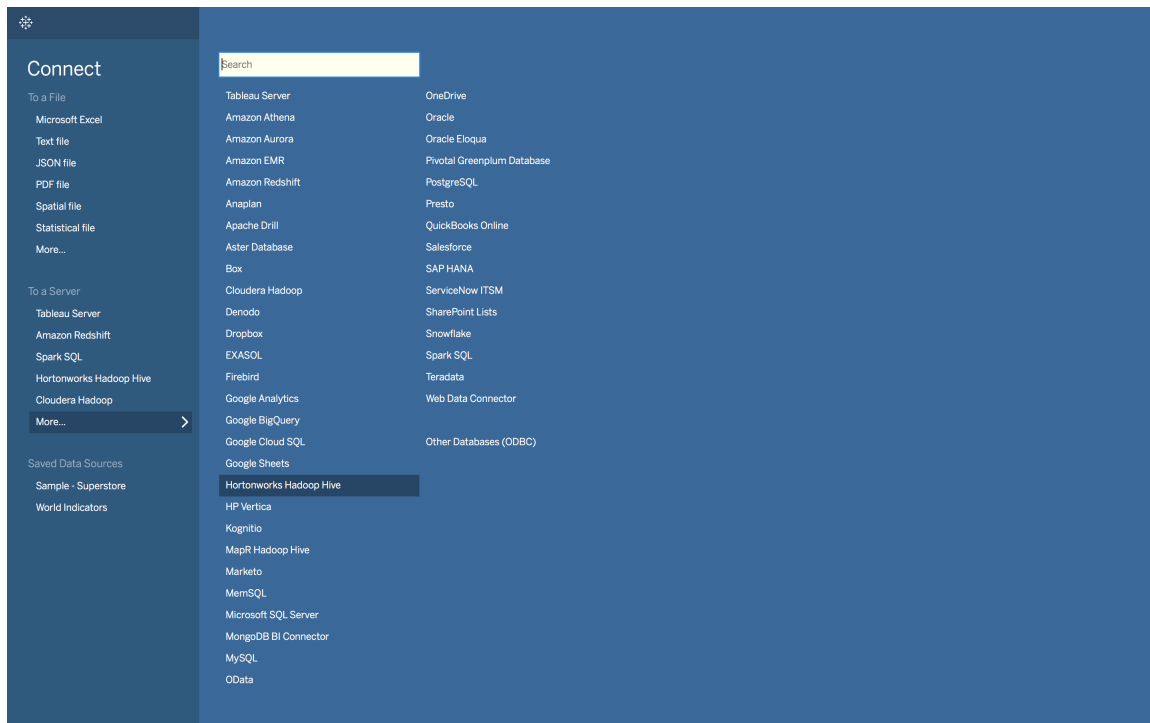
This will present the Tableau UI.

Figure 26. Tableau UI



5. Navigate to **Connect > To a Server > More**
6. Select **Hortonworks Hadoop Hive** from the list of servers.

Figure 27. Selecting Hortonworks Hadoop Hive Server



7. Populate the server connection pop up.

- a. Enter *machine-login1* in the **Server** field, using the FQDN to ensure that it matches the domain name for the SSL certificate. *machineName* is used as an example for the name of the machine and should be replaced with the actual machine name when following this step.
- b. Enter 10000 (which is the port number configured in HA Proxy) in the **Port** field.
- c. Select **HiveServer2** from the **Type** drop down.
- d. Select **User Name and Password (SSL)** from the **Authentication** drop down.
- e. Enter values in the **Username** and **Password** fields.
- f. Select the **Require SSL** check-box.
- g. Click on the **No custom configuration file specified (click to change)...** link.

Figure 28. Connect HiveServer2 to Tableau Securely

Hortonworks Hadoop Hive

Server: Port:

Enter information to sign in to the server:

Authentication:

Transport:

Username:

☒ Require SSL
No custom configuration file specified (click to change)...

[Initial SQL...](#)

- h. Select **Use the following custom SSL certificate** option on the **Configure SSL certificate** pop up.

Figure 29. Tableau Configure SSL Pop up

Configure SSL Certificate

Configure and Use Public SSL Certificate

Choose a public SSL certificate to be used with this connection to verify the identity of the remote server.

☐ Do not use a custom SSL certificate file

☒ Use the following custom SSL certificate file

- i. Select the **Browse** button to select the SSL certificate file.
- j. Select the **OK** button.
- k. Select the **Sign In** button.
8. Perform data visualization/exploration tasks as needed using Tableau.
9. Request an administrator to flex down the YARN cluster.

NOTE: Cray recommends that YARN clusters for Tableau connectivity be flexed down only by administrators on Urika-GX. For more information, administrators should see the `urika-yam-flexdown` man page or refer to the section titled 'Flex up a YARN sub-cluster on Urika-GX' of the 'Urika®-GX Analytic Applications Guide'.

16.3 Connect Tableau to the Spark Thrift Server

Prerequisites

- Ensure that the system is running in the service mode that allows use of Tableau and Spark Thrift Server. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- This procedure requires the following software to be installed on the client machine, which is an external machine connected to Urika-GX:
 - Tableau Desktop (version 10.2)
 - Simba Spark ODBC driver.
- If using a Mac, the following procedure requires version 10.11 of the OS X operating system.

About this task

The Spark Thrift Server provides access to SparkSQL via JDBC or ODBC. It supports almost the same API and many of the features as those supported by HiveServer2. On Urika-GX, HiveServer2/Hive Thrift Server and Spark Thrift server are used for enabling connections from ODBC/JDBC clients, such as Tableau.

The Spark Thrift server ships pre-configured with LDAP authentication.

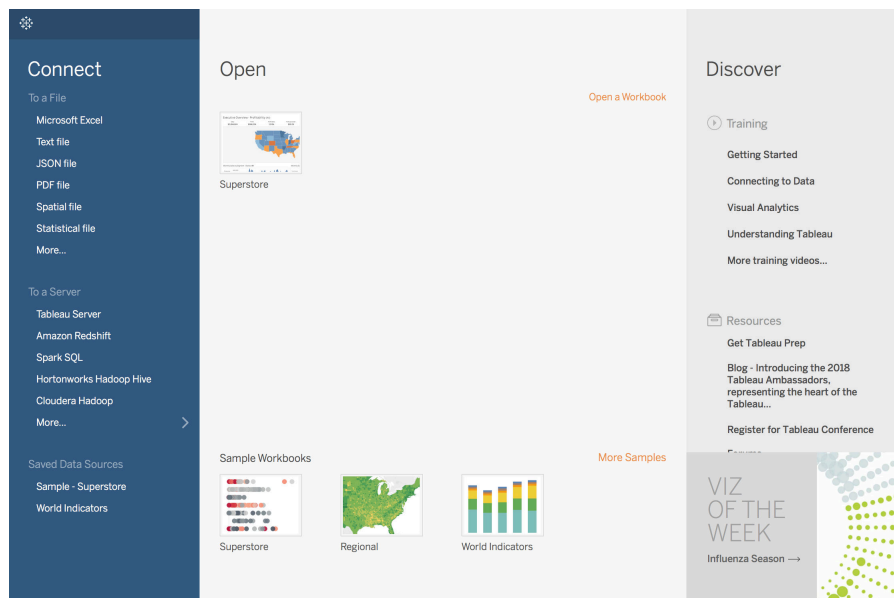


CAUTION: The recommended approach for using Tableau with the Spark Thrift server on Urika-GX is to have multiple users (admin and non-admin) to use the same Spark Thrift server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown. In addition, though it is possible for multiple users to connect to each other's Spark Thrift server, doing so can result in loss of connectivity if the server is brought down by the user who brings it up.

Procedure

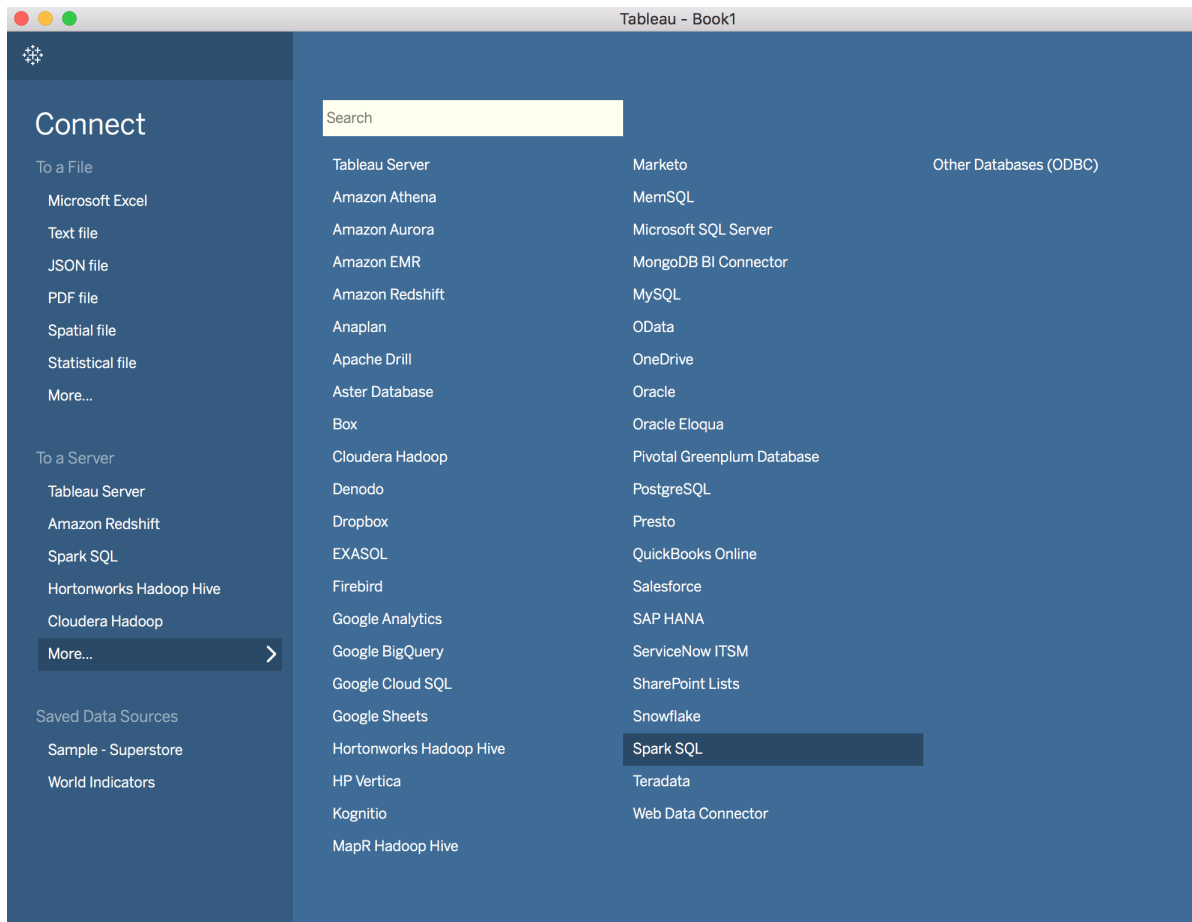
1. Request the administrator to verify that the Spark Thrift server is running and to start it if it is not already up.
Administrators should refer to the section titled, '*Control the Spark Thrift Server*' in the '*Urika®-GX System Administration Guide*' to start the Spark Thrift server. Cray recommends to have the Spark Thrift server stopped by administrators. In order to start the Spark Thrift server non-admins have started without administrative privileges, non-admins should refer to [Manage the Spark Thrift Server as a Non-Admin User](#) on page 73.
2. Launch the Tableau application on a client machine.
This will present the Tableau UI.

Figure 30. Tableau UI



3. Navigate to **Connect > To a Server > More**
4. Select **Spark SQL** server from the list of servers.

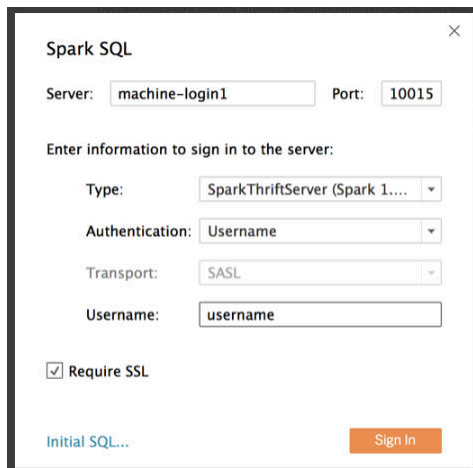
Figure 31. Selecting Spark SQL Server



5. Populate the server connection pop up.

- a. Enter *hostname-login1* in the **Server** field, where *hostname* is used as an example for the name of the machine and should be replaced with the actual machine name when following this step.
- b. Enter **10015** in the **Port** field.
- c. Select **SparkThriftServer (Spark 1.1 and later)** from the **Type** drop down.
- d. Select **User Name and Password** from the **Authentication** drop down.
- e. Enter values in the **Username** and **Password** fields.
- f. Select the **Sign In** button.

Figure 32. Tableau's Spark Connection Pop up



The image shows a 'Spark SQL' connection dialog box. It has a title bar with a close button. Inside, there are fields for 'Server' (containing 'machine-login1') and 'Port' (containing '10015'). Below these is a section titled 'Enter information to sign in to the server:'. This section contains four dropdown menus: 'Type' (set to 'SparkThriftServer (Spark 1...)', 'Authentication' (set to 'Username'), 'Transport' (set to 'SASL'), and 'Username' (containing 'username'). There is a checkbox labeled 'Require SSL' which is checked. At the bottom left is a link 'Initial SQL...' and at the bottom right is an orange 'Sign In' button.

6. Perform data visualization/exploration tasks as needed.
7. Request an administrator to stop the Spark Thrift server service.

Administrators should refer to the section titled, '*Control the Spark Thrift Server*' in the '*Urika®-GX System Administration Guide*' to stop the Spark Thrift server. Cray recommends to have the Spark Thrift server stopped by administrators. In order to stop the Spark Thrift server non-admins have started without administrative privileges, non-admins should refer to '*Manage the Spark Thrift Server as a Non-Admin User*' of the '*Urika-GX Analytic Applications Guide*'.

16.4 Connect Tableau to the Spark Thrift Server Securely

Prerequisites

- Ensure that the system is running in the service mode that allows use of Tableau and Spark Thrift Server. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- This procedure requires the following software to be installed on the client machine:
 - Tableau Desktop (version 10.2)
 - Simba Spark ODBC driver.
- If using a Mac, the following procedure requires version 10.11 of the OS X operating system.

About this task



CAUTION: The recommended approach for using Tableau with the Spark Thrift server on Urika-GX is to have multiple users (admin and non-admin) to use the same Spark Thrift server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown. In addition, though it is possible for multiple users to connect to each other's Spark Thrift

server, doing so can result in loss of connectivity if the server is brought down by the user who brings it up.

Procedure

1. Request an administrator to verify that SSL is enabled.

Administrators should follow instructions listed in section '*Enable SSL on Urika-GX*' of the '*Urika-GX System Administration Guide*' to enable SSL.

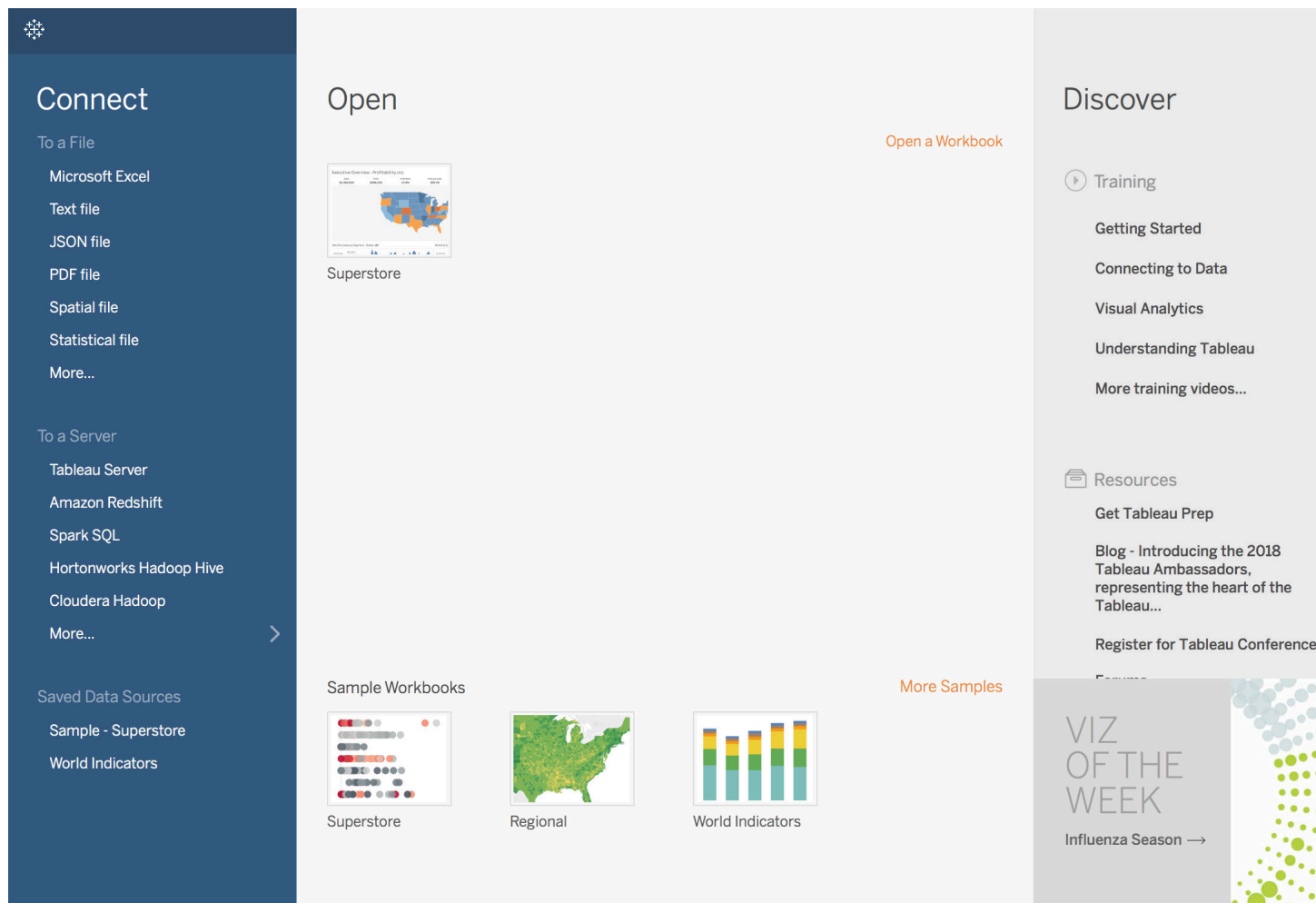
2. Request the administrator to verify that the Spark Thrift server is running and to start it if it is not already up.

Administrators should refer to the section titled, '*Control the Spark Thrift Server*' in the '*Urika-GX System Administration Guide*' to start the Spark Thrift server. Cray recommends to have the Spark Thrift server stopped by administrators. In order to start the Spark Thrift server non-admins have started without administrative privileges, non-admins should refer to '*Manage the Spark Thrift Server as a Non-Admin User*' of the '*Urika-GX Analytic Applications Guide*'.

3. Launch the Tableau application.

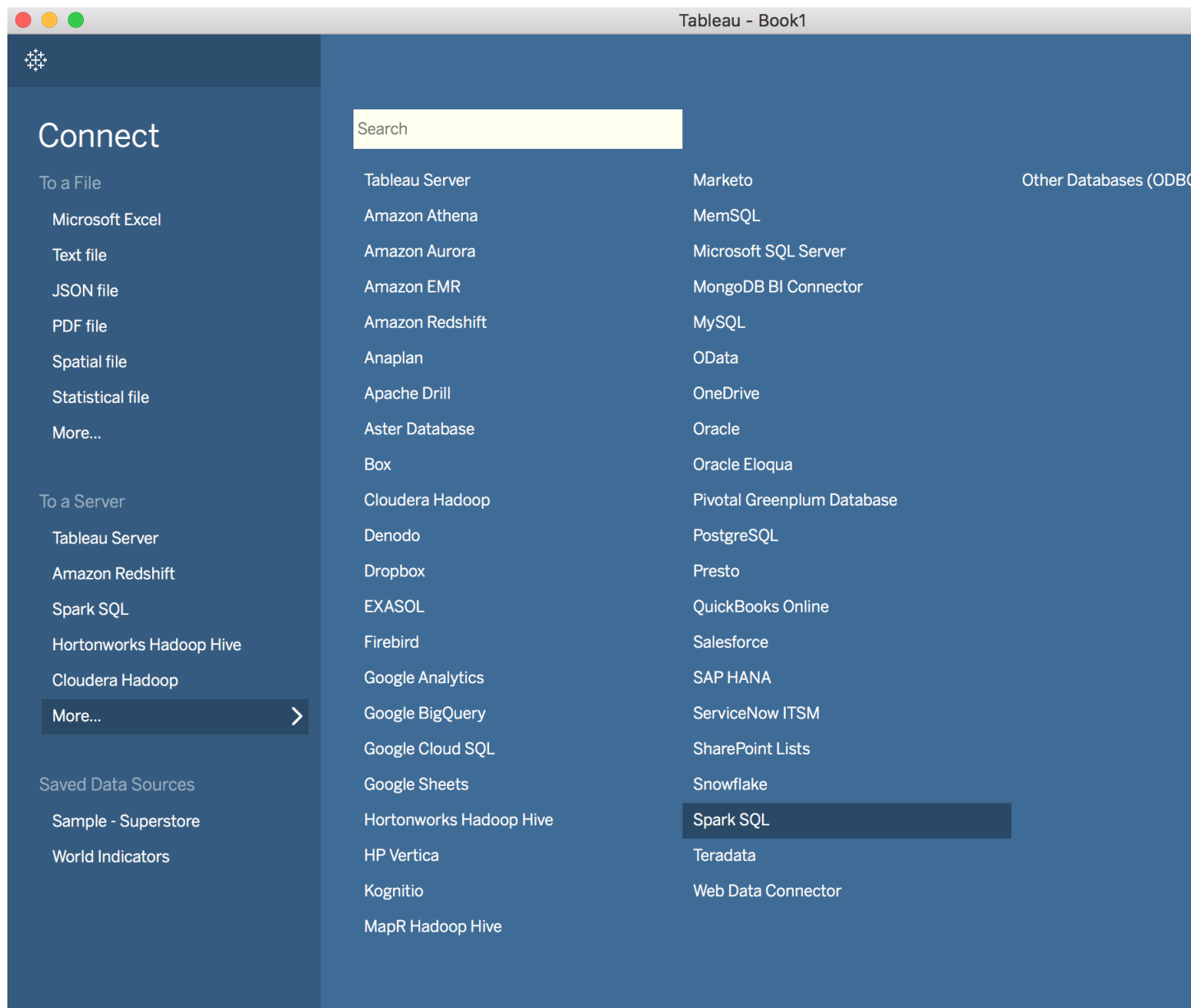
This will present the Tableau UI.

Figure 33. Tableau UI



4. Navigate to **Connect > To a Server > More**
5. Select **Spark SQL** server from the list of servers.

Figure 34. Selecting Spark SQL Server



6. Populate the server connection pop up.

- Enter *machine-login1* in the **Server** field, using the FQDN to ensure that it matches the domain name for the SSL certificate. *machine* is used as an example for the name of the machine and should be replaced with the actual machine name when following this step.
- Enter 10015 in the **Port** field.
- Select **SparkThriftServer (Spark 1.1 and later)** from the **Type** drop down.
- Select **User Name** from the **Authentication** drop down.
- Enter values in the **Username** field.

- f. Select the **Require SSL** check box
- g. Select the **Sign In** button.

Figure 35. Tableau's Spark Connection Pop up

- 7. Perform data visualization/exploration tasks as needed.
- 8. Request an administrator to stop the Spark Thrift server.

Administrators should refer to the section titled, '*Control the Spark Thrift Server*' in the '*Urika-GX System Administration Guide*' to stop the Spark Thrift server. Cray recommends to have the Spark Thrift server stopped by administrators. In order to stop the Spark Thrift server non-admins have started without administrative privileges, non-admins should refer to '*Manage the Spark Thrift Server as a Non-Admin User*' of the '*Urika-GX Analytic Applications Guide*'.

16.5 Connect Tableau to Apache Spark Thrift Server on a VM

Prerequisites

- Ensure that the system is running in the service mode that allows use of Tableau and Spark Thrift Server. Execute the `urika-state` or `urika-service-mode` commands to check the service mode. For more information, refer to the `urika-state` or `urika-service-mode` man pages and see [Urika-GX Service Modes](#) on page 7.
- This procedure requires the following software to be installed on the client machine:
 - Tableau Desktop (version 10.2)
 - Simba Spark ODBC driver.

Since Tableau Desktop version 10.2, the Mac driver for Spark SQL is installed by default with Tableau Desktop
- If using a Mac, the following procedure requires version 10.11 of the OS X operating system.

About this task



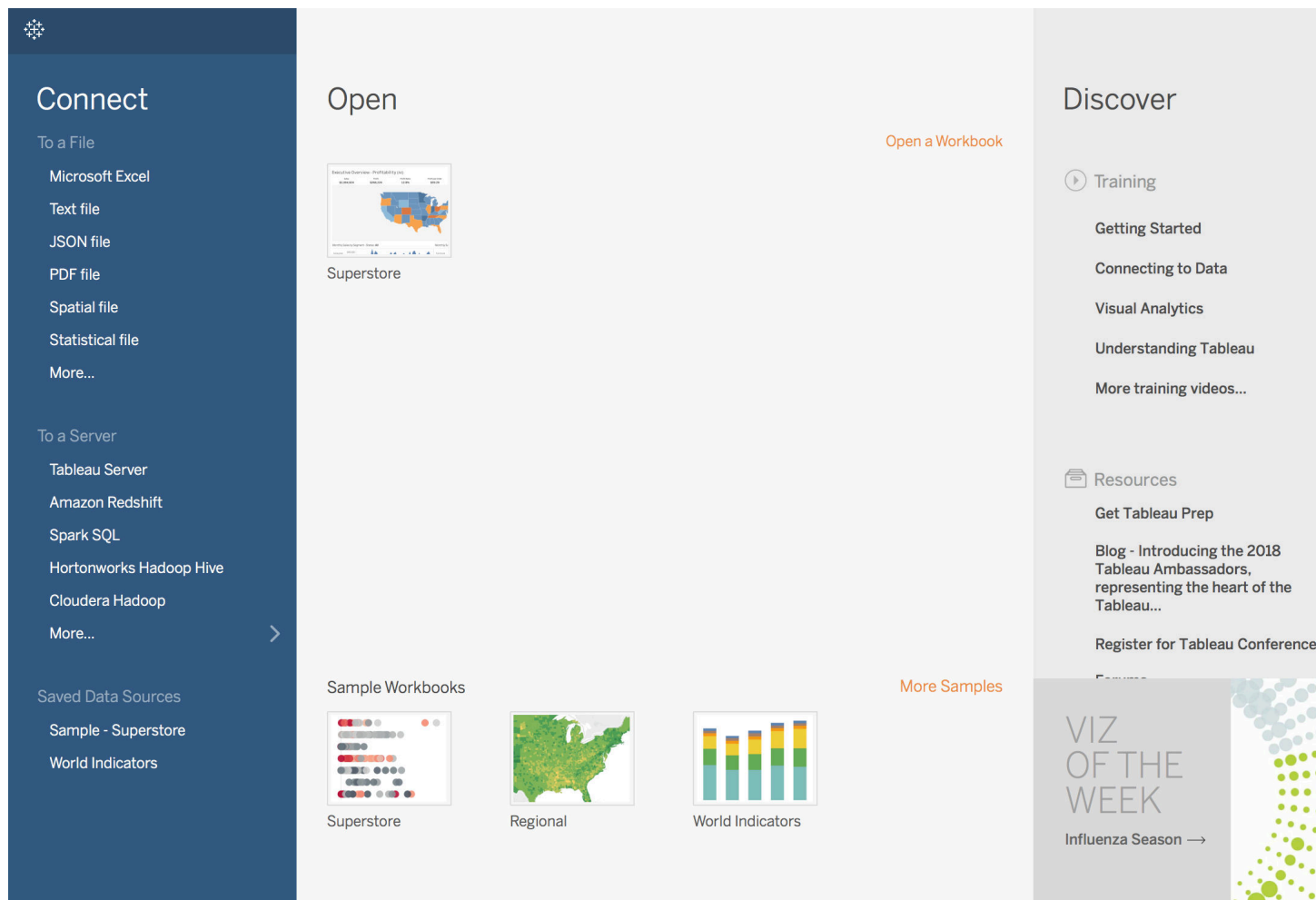
CAUTION: The recommended approach for using Tableau with the Spark Thrift server on Urika-GX is to have multiple users (admin and non-admin) to use the same Spark Thrift server (that has been started by an administrator) instead of spinning up their own individual servers, as doing so could result in resource lockdown. In addition, though it is possible for multiple users to connect to each other's Spark Thrift server, doing so can result in loss of connectivity if the server is brought down by the user who brings it up.

Procedure

1. Launch the Tableau application.

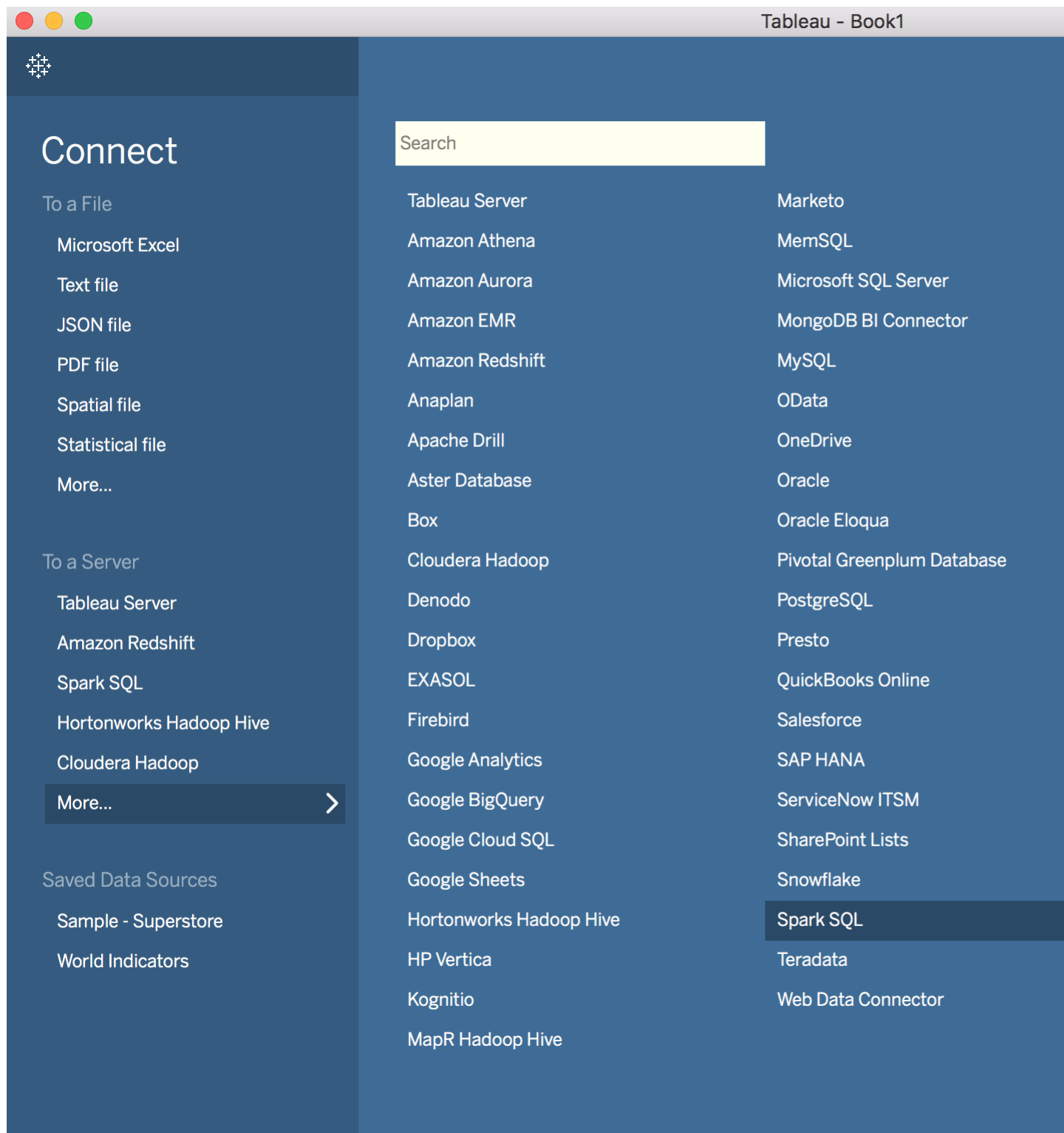
This will present the Tableau UI.

Figure 36. Tableau UI



2. Navigate to **Connect > To a Server > More**
3. Select **Spark SQL** server from the list of servers.

Figure 37. Selecting Spark SQL Server



4. Populate the server connection pop up.

- a. Enter **external ip or hostname** in the **Server** field, where *hostname* is used as an example for the name of the machine and should be replaced with the actual machine name when following this step.
- b. Enter **10000** in the **Port** field.
- c. Select **SparkThriftServer (Spark 1.1 and later)** from the **Type** drop down.
- d. Select **User Name and Password** from the **Authentication** drop down.
- e. Enter values in the **Username** and **Password** fields.
- f. Select **HTTP** in transport mode and enter HTTP Path as **sparkthrift**
- g. Select the **Sign In** button.

Figure 38. Connect Tableau to the Spark Thrift Server Using SSL

Spark SQL

Server: Port:

Enter information to sign in to the server:

Type:

Authentication:

Transport:

Username:

Password:

HTTP Path:

☐ Require SSL

[Initial SQL...](#)

5. Perform data visualization/exploration tasks as needed.
6. Request an administrator to stop the Spark Thrift server.

Administrators should refer to the section titled, '*Control the Spark Thrift Server*' in the '*Urika-GX System Administration Guide*' to stop the Spark Thrift server. Cray recommends to have the Spark Thrift server stopped by administrators. In order to stop the Spark Thrift server non-admins have started without administrative privileges, non-admins should refer to '*Manage the Spark Thrift Server as a Non-Admin User*' of the '*Urika-GX Analytic Applications Guide*'.

16.6 Enable SSL for Spark Thrift Server of a Tenant

Prerequisites

This procedure requires root privileges.

About this task

This procedure provides instructions for enabling SSL for Spark Thrift Server of a tenant

Procedure

1. Convert the CA certificate file to Java keystore format by using one of the following methods:

- Update the keystore file.
 1. Rename the Java keystore file to the filename `keystore`
 2. Place the `keystore` file in the `/global/tenants/TENANT_NAME/sts/SSL/` directory
- Create a symbolic link to the keystore path:

```
# ln -s /path/to/real/keystore/file /global/tenants/TENANT_NAME/sts/ssl/keystore
```

2. Verify that the `keystore` file exists:

```
# /bin/ls -l /global/tenants/TENANT_NAME/sts/ssl/keystore
```

3. Modify the `/global/tenants/TENANT_NAME/hive/conf/hive-site.xml` file to make the following changes:
 - a. Change the value of `hive.server2.use.SSL` to `true`.
 - b. Change the value of `hive.server2.keystore.password` to the keystore's password
 - c. Stop and the start the Spark Thrift Server if it is currently running.
4. Enable or disable the SSL mode for this Spark Thrift Server, depending on requirements.
 - To enable the SSL mode:
 1. Modify the `hive-site.xml` file to change the value of `hive.server2.use.SSL` to `true`
 2. Stop and the start the Spark Thrift Server if it is currently running.
 - To disable the SSL mode:
 1. Modify the `hive-site.xml` file to change the value of `hive.server2.use.SSL` to `false`
 2. Stop and the start the Spark Thrift Server if it is currently running.

17 File Systems

Supported file system types on Urika-GX include:

- **Internal file systems**

- Hadoop Distributed File System (HDFS) - Hadoop uses HDFS for storing data. HDFS is highly fault-tolerant, provides high throughput access to application data, and is suitable for applications that have large data sets. Urika-GX also features tiered HDFS storage. HDFS data is transferred over the Aries network.
- Network File System (NFS) - The Urika-GX SMW hosts NFS, which is made available to every node via the management network.
- `/mnt/lustre` - This is a directory that hosts Lustre file system data if DAL/Sonexion is used.



CAUTION: Avoid using NFS for high data transfers and/or large writes as this will cause the network to operate much slower or timeout. NFS, as configured for Urika-GX home directories, is not capable of handling large parallel writes from multiple nodes without data loss. Though it is possible to configure NFS to handle parallel writes, it would require a hard mount, which would have undesired consequences.

File Locations

- Home directories are mounted on (internal) NFS, with limited space
 - Distributed file system (Lustre), if provisioned, is mounted at `/mnt/lustre` and is suitable for larger files.
- Lustre mounts are isolated, with individual tenants having their own mount point.

18 Check the Current Service Mode

Prerequisites

This procedure requires root privileges on the SMW.

About this task

Urika-GX supports two service modes, which dictate the list of services available. These modes include:

- Default
- Secure

Use the following instructions to determine the service mode the system is currently running in.

Procedure

1. Log on to the SMW as root.

```
# ssh root@hostname-smw
```

2. Display the current service mode by using one of the following options:

- Execute the `urika-state` command. This displays the current service mode as well as the status of all the services that are supported in that mode.
- Execute the `urika-service-mode` command.

```
# urika-service-mode
Current mode is: default
```

For more information, refer to the `urika-service-mode` and `urika-state` man pages.

19 Fault Tolerance on Urika-GX

Fault tolerance refers to the ability of a system to continue functioning with minimal intervention in spite of failures and its ability to cope with various kinds of failures.

Urika-GX features fault tolerance to ensure resiliency against system failures. Failed jobs are rescheduled automatically for optimized performance.

- **Zookeeper** - Zookeeper enables highly reliable distributed coordination. On Urika-GX, there are 3 Zookeeper instances running with a quorum of 2. On Urika-GX, Mesos and Marathon use Zookeeper to help provide fault tolerance.
- **Hadoop** - Hadoop is highly fault-tolerant. Whenever there is a failure in the execution of a Hadoop/MapReduce job, the corresponding process is reported to the master and is rescheduled.
 - **HDFS** - HDFS is the data store for all the Hadoop components on Urika-GX. The Secondary HDFS NameNode periodically stores the edits information and the FS Image. HDFS NameNode is the single point of failure. In case of a HDFS NameNode failure, the Hadoop administrator can start the Hadoop cluster with the help of these FS images and edits. Information in the file system is replicated, so if one data node goes down, the data is still available.
- **Spark** - Spark uses a directed acyclic lineage graph to track transformations and actions. Whenever there is failure, Spark checkpoints the failure in the graph and reschedules the next set of computations from that checkpoint on another node.
- **Mesos** - Mesos is the main resource broker for Urika-GX and runs in high availability mode. There are 3 masters running at all times, so that even if one fails, one of the remaining two masters is elected as the leader and there is no disturbance in the process of resource brokering. Furthermore, the Mesos UI is configured using HA proxy to detect the active Mesos master and direct incoming request to it directly.
- **Marathon** - Marathon is a Mesos framework/scheduler that is used to launch and manage long-running services on a cluster. There are three Marathon instances running at all times on the Urika-GX system. If an active Marathon instance goes down, one of the backup Marathon instances is assigned as the leader. Services defined within Marathon are launched and managed anywhere on the cluster where there are available resources. If a Mesos task fails, Marathon will accept more resources from Mesos and another task will be launched, usually on a different node. Re-scheduling of the tasks is usually a framework related decision.

20 Default Urika-GX Configurations

The following table document some basic configuration settings that Urika-GX ships with. This is not an exhaustive list.

Table 8. Urika-GX Default Configurations

Component	Configuration parameter and file location
Spark	<ul style="list-style-type: none"> • <code>spark.shuffle.compress: false</code> • <code>spark.locality.wait: 1 second</code> • Event logging: enabled • Default cores: 8 for <code>spark-shell</code>, 128 for <code>spark-submit</code> and everything else • Default memory allocation: <ul style="list-style-type: none"> ◦ 96 Gigabytes for each executor ◦ 16 Gigabytes to the driver
Mesos	Name of the cluster is configured in an Ansible file located at: <code>/etc/mesos-master/cluster</code> . If this configuration has to be changed specific to the site, it should be done at the time of configuration. The change should be made in the Ansible file's <code>cluster_name</code> variable.
Cray Graph Engine (CGE)	<p>Default logging level: INFO</p> <p>NVP settings - See the 'Cray® Graph Engine User Guide'.</p> <p>Configuration files are located under:</p> <ul style="list-style-type: none"> • <code>CGE_CONFIG_FILE_NAME</code> • <code>CGE_CONFIG_DIR_NAME/cge.properties</code> • <code>Current_Working_Directory/cge.properties</code> • <code>Data_Directory/cge.properties</code> • <code>Home_Directory/.cge/cge.properties</code>
mrunch	<ul style="list-style-type: none"> • <code>NCMDServer=nid00000</code> • <code>MesosServer=localhost # same as --host</code> • <code>MesosPort=5050</code> • <code>MarathonServer=localhost</code> • <code>MarathonPort=8080</code> • <code>DebugFLAG=False # same as --debug</code>

Component	Configuration parameter and file location												
	<ul style="list-style-type: none">VerboseFLAG=False # same as --veboseJobTimeout=0-0:10:0 # ten minutes, same as --timeStartupTimeout=30 # 30 seconds, same as --immediateHealthCheckEnabled=True # Run with Marathon Health Checks enabledHCGracePeriodSeconds=5 # Seconds at startup to delay Health ChecksHCIntervalSeconds=10 # Seconds between Health Check pingsHCTimeoutSeconds=10 # Seconds to answer Health Check successfullyHCMaxConsecutiveFailures=3 # How many missed health checks before app killed												
Flex scripts: <ul style="list-style-type: none">urika-yam-statusurika-yam-flexupurika-yam-flexdownurika-yam-flexdown-all	The default timeout interval is 15 minutes. The configuration file is located at /etc/urika-yam.conf. The recommended default is 15, unit for timeout is minutes. This can be changed per site requirements.												
Grafana	<ul style="list-style-type: none">When a new user signs up for the first time, the default role assigned to the user is that of a Viewer. The admin can then change the new user's role if needed.By default, Grafana's refresh rate is turned off on the Urika-GX system.The default timezone displayed on Grafana is in UTC.Default Grafana roles and permissions are depicted in the following table: Table 9. Default Grafana Roles and Permissions <table><tr><th>Role</th><th>Edit roles</th><th>View graphs</th><th>Edit/create copy of existing graphs</th><th>Add new graphs to existing dashboards</th><th>Create new/import existing dashboards</th></tr><tr><td>Admin</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X (These dashboards persist between sessions)</td></tr></table>	Role	Edit roles	View graphs	Edit/create copy of existing graphs	Add new graphs to existing dashboards	Create new/import existing dashboards	Admin	X	X	X	X	X (These dashboards persist between sessions)
Role	Edit roles	View graphs	Edit/create copy of existing graphs	Add new graphs to existing dashboards	Create new/import existing dashboards								
Admin	X	X	X	X	X (These dashboards persist between sessions)								

Component	Configuration parameter and file location				
	Role	Edit roles	View graphs	Edit/create copy of existing graphs	Add new graphs to existing dashboards
	Viewer (default role)		X		X
	Editor		X	X	X
	Read only editor		X	X	X

20.1 Default Grafana Dashboards

The default set of Grafana dashboards shipped with Urika-GX include:

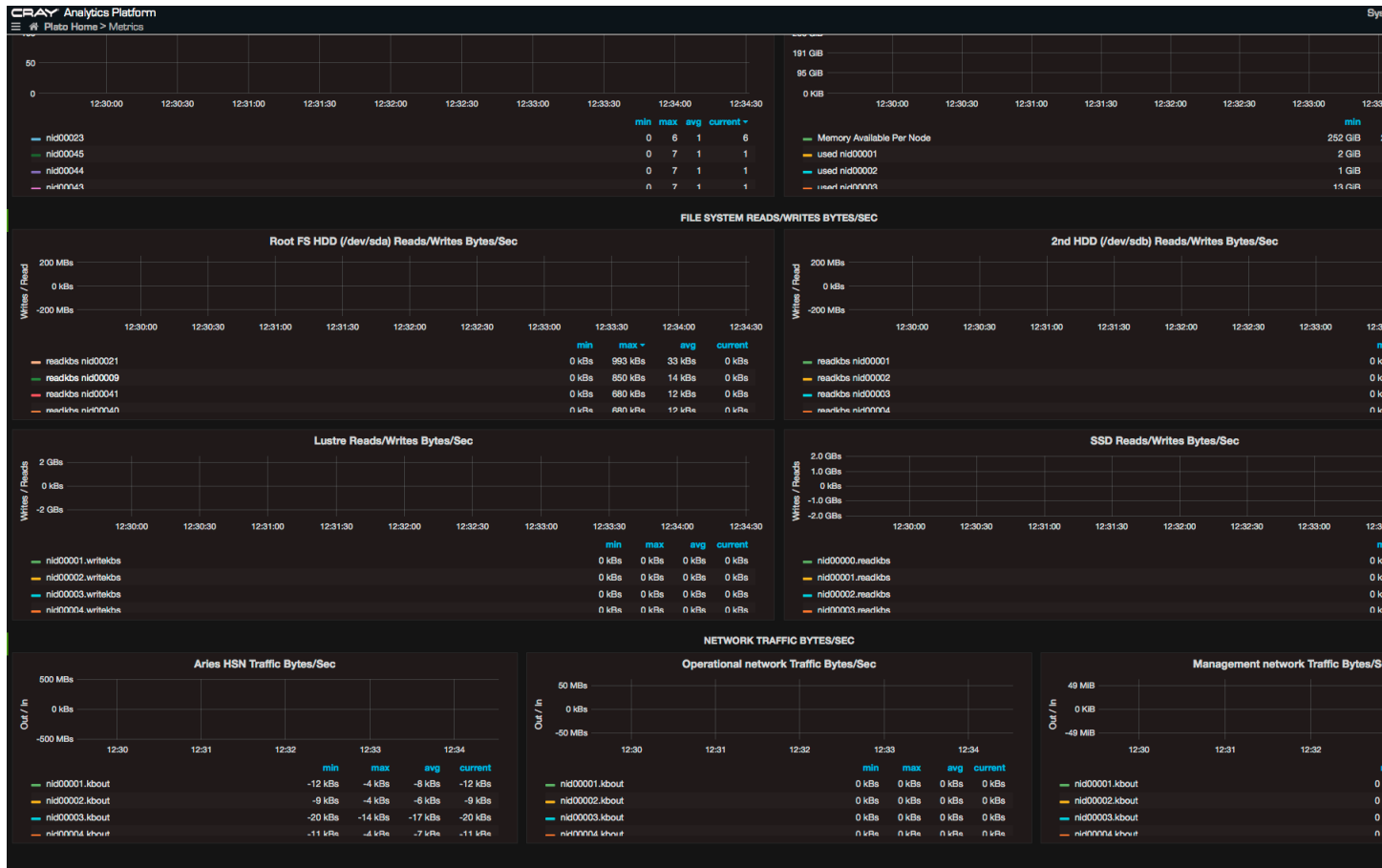
- **Aggregate Compute Node Performance Statistics** - Displays graphs representing statistical data related to network, CPU, and I/O utilization for all Urika-GX nodes.

This dashboard contains the following graphs:

- **CPU AND MEMORY**
 - **CPU utilization** - Displays the overall CPU utilization for all nodes.
 - **Memory utilization** - Displays the overall memory used by all nodes.
- **FILE SYSTEM DATA RATES**
 - **SSD Reads/Writes Bytes/Second** - Displays SSD utilization for all nodes.
 - **Second Hard Drive (/dev/sdb) Reads/Writes Bytes/Sec** - Displays utilization of secondary hard disk space for all nodes.
 - **Root FS HDD (/dev/sda) Reads/Writes Bytes/Sec** - Displays utilization of root files system on the hard drive for all nodes.
 - **Lustre Reads/Writes Bytes/Second** - Displays the aggregate Lustre I/O for all the nodes.
- **NETWORK READS AND WRITES**
 - **Aries HSN Bytes/Sec In/Out** - Displays the overall Aries network TCP traffic information for nodes. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.

- **Operational network Bytes/sec In/Out** - Displays the overall operational network traffic information for all nodes.
- **Management network Bytes/sec In/Out** - Displays the overall management network traffic information for all nodes.

Figure 39. Aggregate Compute Node Performance Statistics



- **Basic Metrics** - Displays graphs representing statistical data related to network, CPU, and I/O utilization for the Urika-GX system, as a whole.

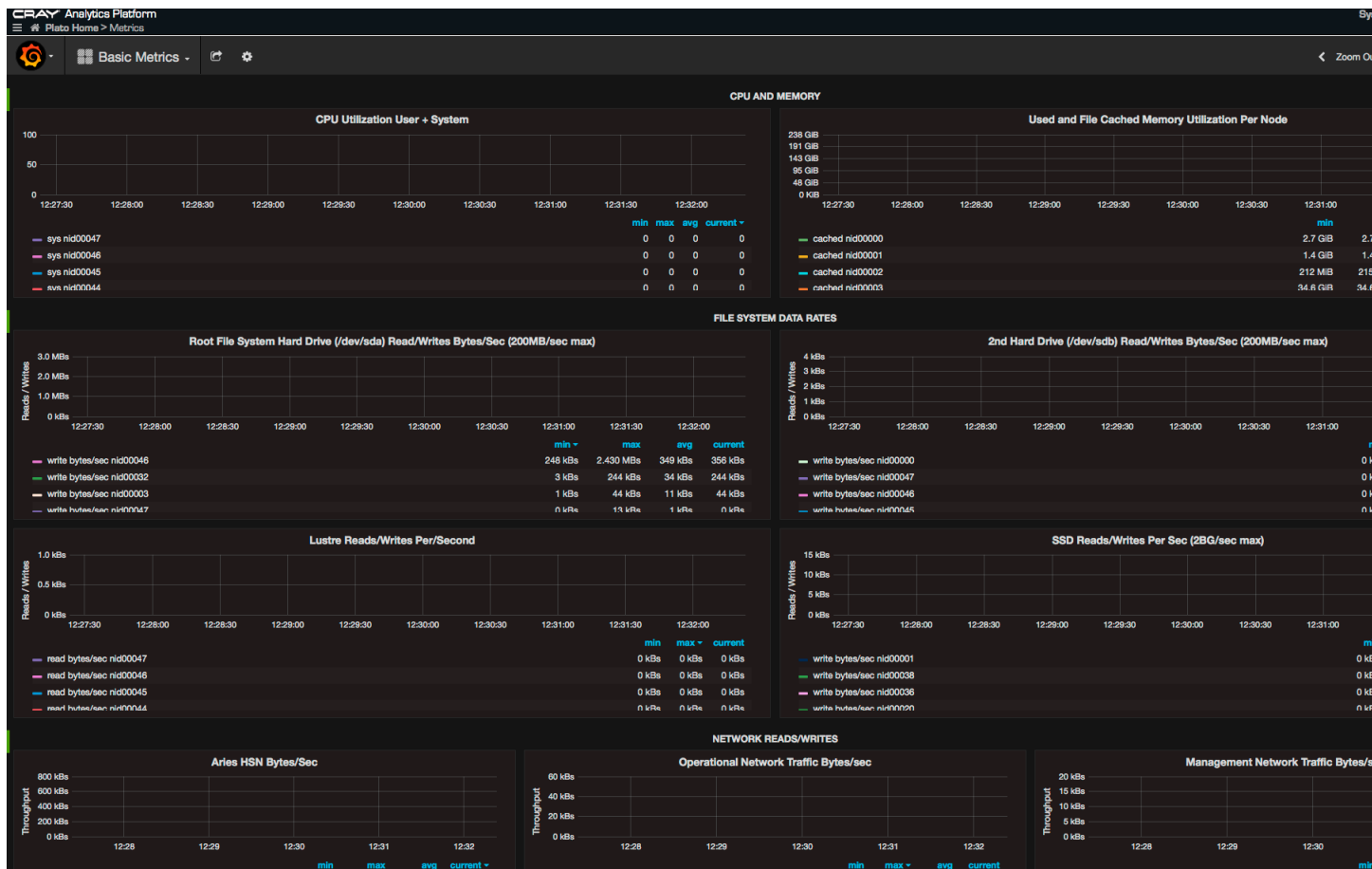
TIP: It is recommended that administrators use the **Basic Metrics** dashboard before using other dashboards to retrieve a summary of the system's health.

This dashboard contains the following graphs:

- **CPU AND MEMORY**
 - **Used and File Cached Memory** - Displays the used and file cached memory for each node.
 - **CPU Utilization User + System** - Displays the CPU utilization by the user and system for each node
- **FILE SYSTEM DATA RATES**
 - **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec (200MB/sec max)** - Displays information about the usage of memory on the root file system for each node.

- **2nd Hard Drive (/dev/sdb) Read/Writes** - Displays information about the usage of memory on the secondary hard drive of each node.
- **Lustre Read/Writes Bytes/Second** - Displays the number of Lustre reads/writes for each node.
- **SSD Read/Writes Bytes/Second** - Displays the number of reads/writes of the SSDs installed on the system.
- **NETWORK READS/Writes**
 - **Aries HSN Bytes/Sec In/Out** - Displays the Aries network TCP traffic information for each node. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.
 - **Operational network Bytes/sec In/Out** - Displays the overall operational network traffic information for each node.
 - **Management network Bytes/sec In/Out** - Displays the overall management network traffic information for each node.
- **FILE SYSTEM UTILIZATION**
 - **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec (200MB/sec max)** - Displays information about the usage of memory on the root file system for each node.
 - **2nd Hard Drive (/dev/sdb) Read/Writes** - Displays information about the usage of memory on the secondary hard drive of each node.
 - **Lustre Read/Writes Per/Second** - Displays the number of Lustre reads/writes for each node.
 - **SSD Read/Writes Per/Second** - Displays the number of reads/writes of each node's SSDs.
- **NETWORK ERRORS AND DROPPED PACKETS**
 - **Aries HSN Dropped Packets and Errors Per Sec** - Displays the number of dropped packets/errors per second for the Aries network TCP traffic information for each node. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.
 - **Operational network Dropped Packets and Errors Per Sec** - Displays the number of dropped packets/errors per second for the operational network TCP traffic information for each node.
 - **Management network Dropped Packets and Errors Per Sec** - Displays the number of dropped packets/errors per second for the management network TCP traffic information for each node.

Figure 40. Basic Metrics Dashboard



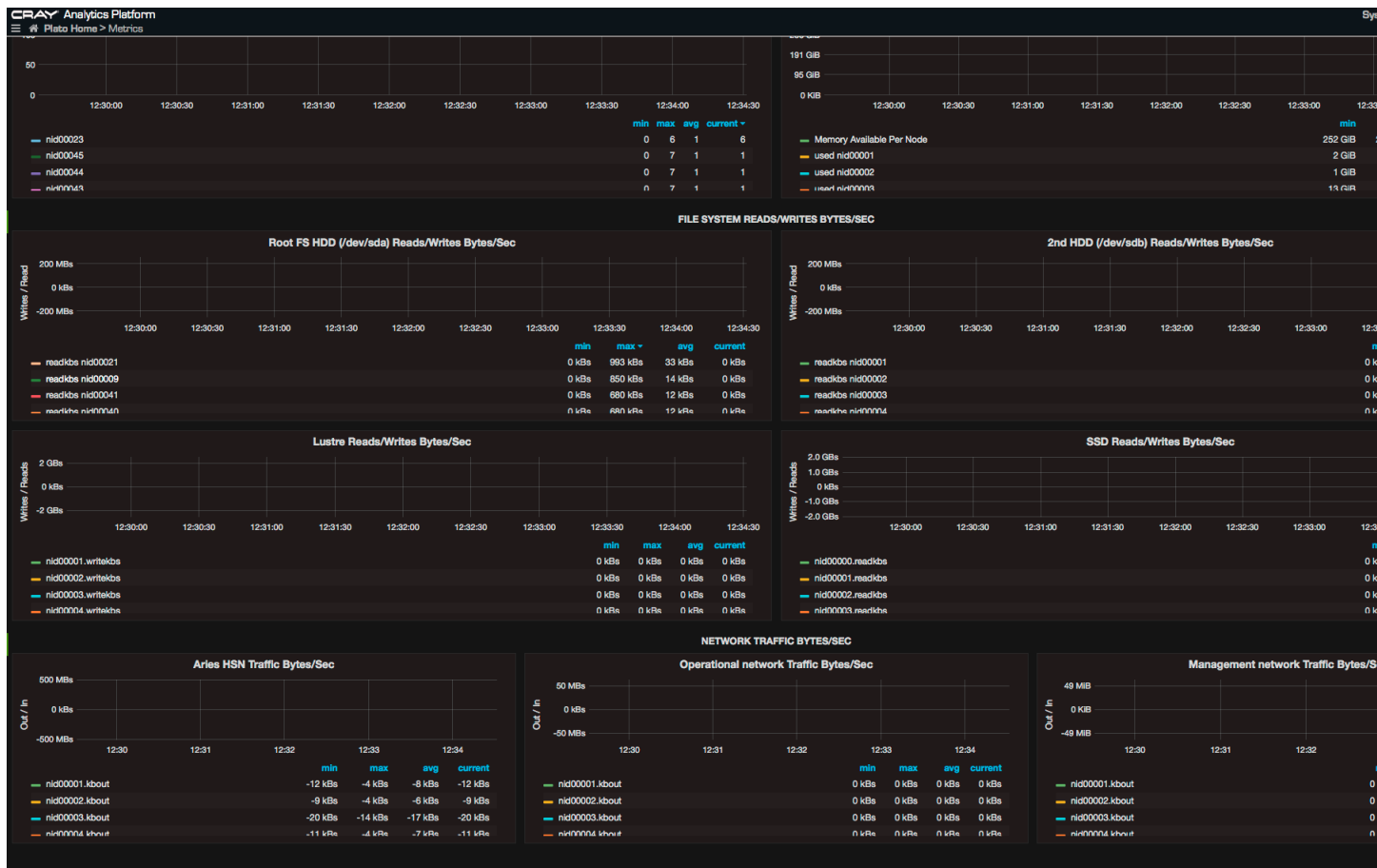
- **Compute Node Performance Statistics** - Displays graphs representing statistical data related to network, CPU, and I/O utilization for all Urika-GX compute nodes.

This dashboard contains the following graphs:

- **CPU MEMORY UTILIZATION**
 - **CPU utilization** - Displays the overall CPU utilization for all the compute nodes.
 - **Memory utilization** - Displays the overall memory (in KB or GB) used by all the compute nodes.
- **FILE SYSTEM READS/WRITE BYTES/SEC**
 - **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec (200MB/sec max)** - Displays information about the usage of memory on the root file system by compute nodes..
 - **2nd Hard Drive (/dev/sdb) Read/Writes** - Displays information about the usage of memory by compute nodes on the secondary hard drive.
 - **Lustre Read/Writes Per/Second** - Displays the number of Lustre reads/writes by compute nodes.
 - **SSD Read/Writes Per/Second** - Displays the number of reads/writes of the compute node SSDs installed on the system.
- **NETWORK READS/Writes**

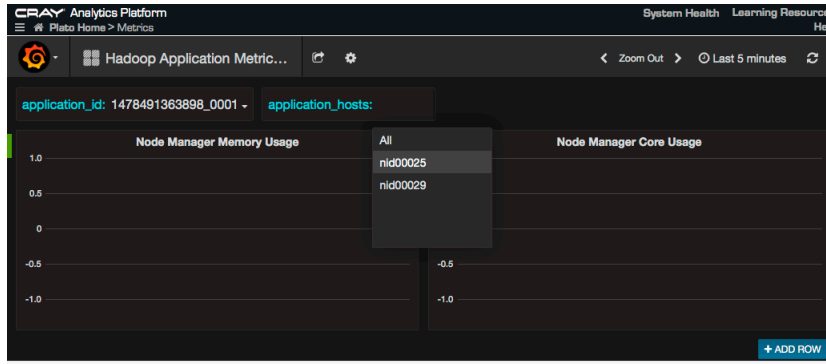
- **Aries HSN Bytes/Sec In/Out** - Displays the Aries network TCP traffic information for compute nodes. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.
- **Operational network Bytes/sec In/Out** - Displays the overall operational network traffic information for compute nodes.
- **Management network Bytes/sec In/Out** - Displays the overall management network traffic information for compute nodes.

Figure 41. Compute Node Performance Statistics



- **Hadoop Application Metrics** - This section contains the following graphs:
 - **Node Manager Memory Usage** - Displays the average memory usage per application in mega bytes per second (MBPS). The Y-axis is in MBPS.
 - **Node Manager Core Usage** - Displays the average CPU usage per application in MilliVcores . The Y-axis refers to MilliVcores.

Figure 42. Hadoop Applications Metrics Dashboard

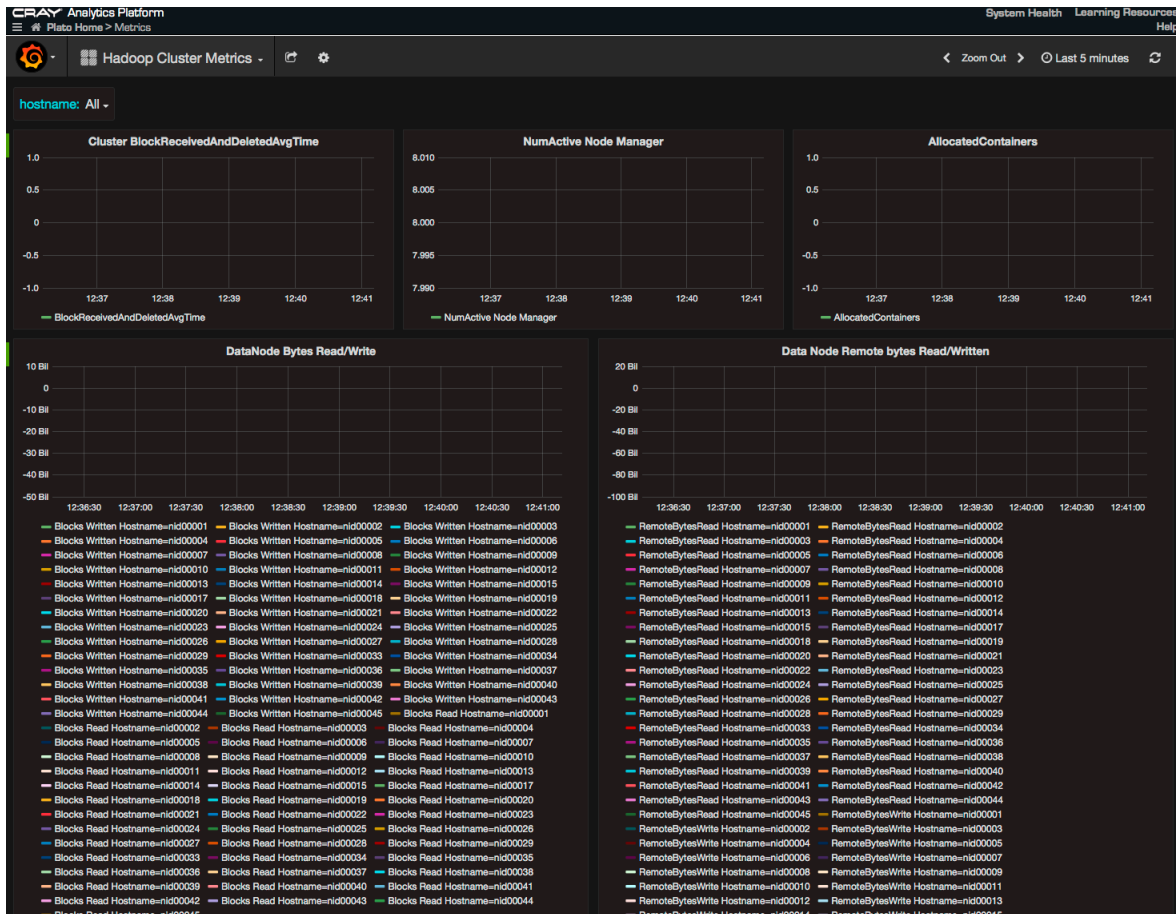


- **Hadoop Cluster Metrics** - Displays graphs representing statistical data related to Hadoop components, such as HDFS Data Nodes and HDFS Name Nodes.

This dashboard contains the following sections:

- **Cluster BlockReceivedAndDeletedAvgTime** - Displays the average time in milliseconds for the hdfs cluster to send and receive blocks. The Y-axis represents time in milliseconds.
- **NumActive Node Manager** - Displays the number of Node Managers up and running in the Hadoop cluster at a given time. The Y-axis represents a linear number.
- **AllocatedContainers** - Displays the number of allocated YARN containers by all the jobs in the hadoop cluster. The Y-axis represents a linear number.
- **DataNode Bytes Read/Write** - Displays the number of bytes read or write per node from local client in the hadoop cluster. The Y-axis refers to a linear number. Read is shown on the positive scale. Write is shown on the negative scale.
- **Data Node Remote bytes Read/Written** - Displays the number of bytes read or written per node from remote clients in the Hadoop cluster. The Y-axis represents a linear number. The number of reads are shown on the positive scale. The number of writes are shown on the negative scale.

○ Figure 43. Hadoop Cluster Metrics Dashboard



- **Non-compute Node Performance Statistics** - Displays graphs representing statistical data related to network, CPU, and I/O utilization for all the non-compute (I/O and login) nodes of Urika-GX.

This dashboard contains the following graphs:

- **CPU MEMORY UTILIZATION**

- **CPU utilization** - Displays the overall CPU utilization for I/O and login (non-compute) nodes.
- **Memory utilization** - Displays the overall memory (in KB or GB) used by all the I/O and login nodes.

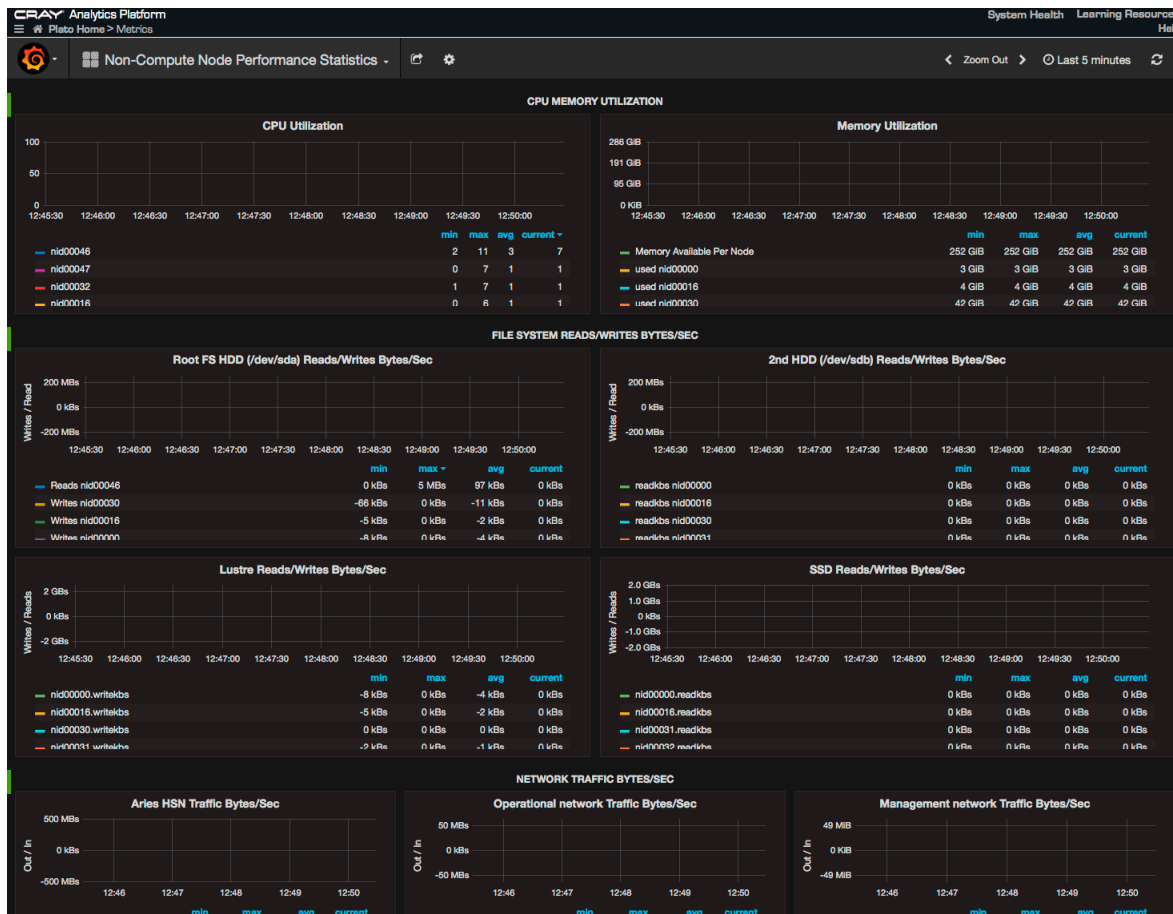
- **FILE SYSTEM READS/WRITE BYTES/SEC**

- **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec (200MB/sec max)** - Displays information about the usage of memory on the root file system by I/O and login nodes.
- **2nd Hard Drive (/dev/sdb) Read/Writes** - For I/O and login nodes, displays information about the usage of memory by compute nodes on the secondary hard drive.
- **Lustre Read/Writes Bytes/Second** - Displays the number of Lustre reads/writes by I/O and login nodes.
- **SSD Read/Writes Bytes/Second** - Displays the number of SSD reads/writes of the I/O and login nodes.

- **NETWORK READS/Writes**

- **Aries HSN Bytes/Sec In/Out** - Displays the Aries network TCP traffic information for I/O and login nodes. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.
- **Operational network Bytes/sec In/Out** - Displays the overall operational network traffic information for I/O and login nodes.
- **Management network Bytes/sec In/Out** - Displays the overall management network traffic information for I/O and login nodes.

Figure 44. Non-compute Node Performance Statistics



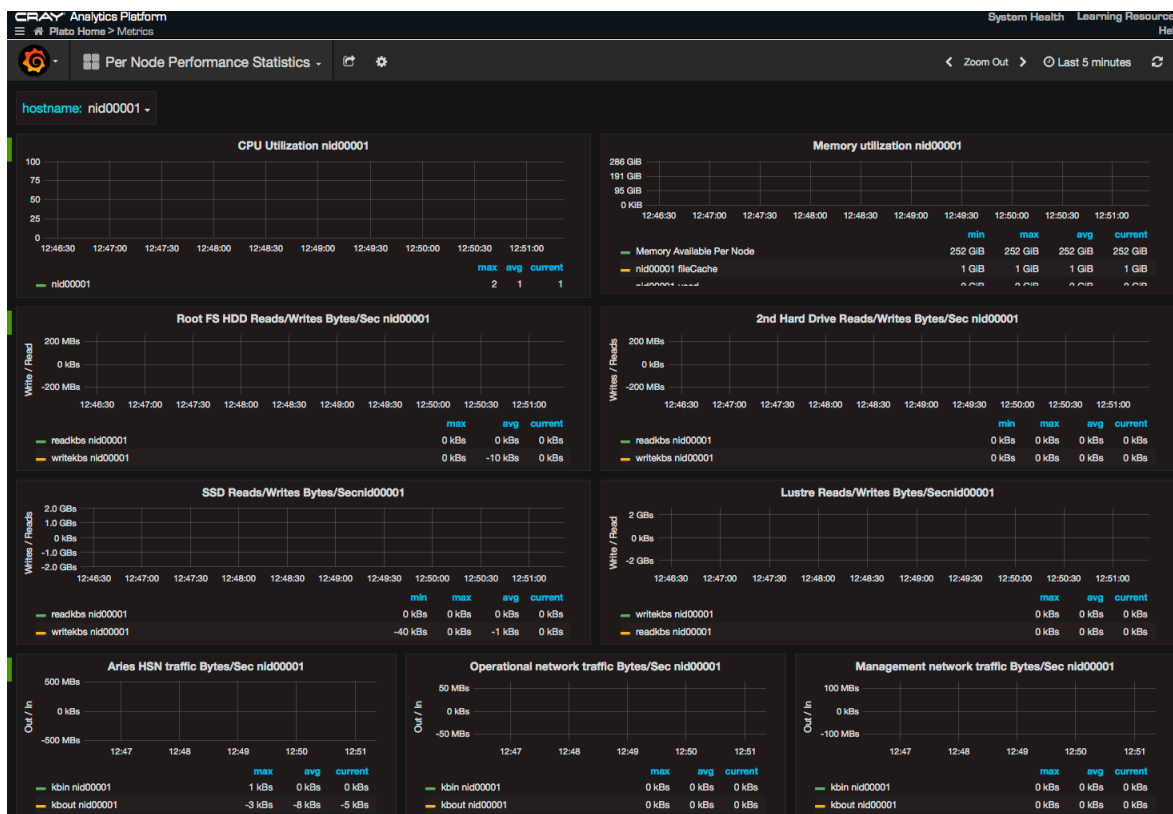
- **Per Node Performance Statistics** - Displays graphs representing statistical data related to network, CPU, and I/O utilization for individual Urika-GX nodes. The node's hostname can be selected using the **hostname** drop down provided on the UI.

This dashboard contains the following graphs:

- **CPU utilization** - Displays the CPU utilization for the selected node.
- **Memory utilization** - Displays the memory (in KB or GB) used by the selected node.
- **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec** - Displays the HDD SDA utilization for the selected node.
- **2nd Hard Drive (/dev/sdb) Read/Writes** - Displays the HDD SDB utilization for the selected node.
- **SSD Read/Writes Bytes/Second** - Displays the number of SSD reads/writes per second for each node.

- **Lustre Read/Writes Bytes/Second** - Displays the number of Lustre reads/writes by the selected node.
- **Aries HSN Bytes/Sec In/Out** - Displays the Aries network TCP traffic information for the selected node. Note that non-TCP Aries traffic, including most traffic generated by CGE, is not shown here.
- **Operational network Bytes/sec In/Out** - Displays the overall operational network traffic information for the selected node.
- **Management network Bytes/sec In/Out** - Displays the overall management network traffic information for the selected node.
- **NFS HDFS Lustre Percentage Used** - Displays the percentage of NFS, HDFS and Lustre used by the selected node.
- **File System Percent Used** - Displays the percentage of file system used by the selected node.

Figure 45. Per Node Performance Statistics



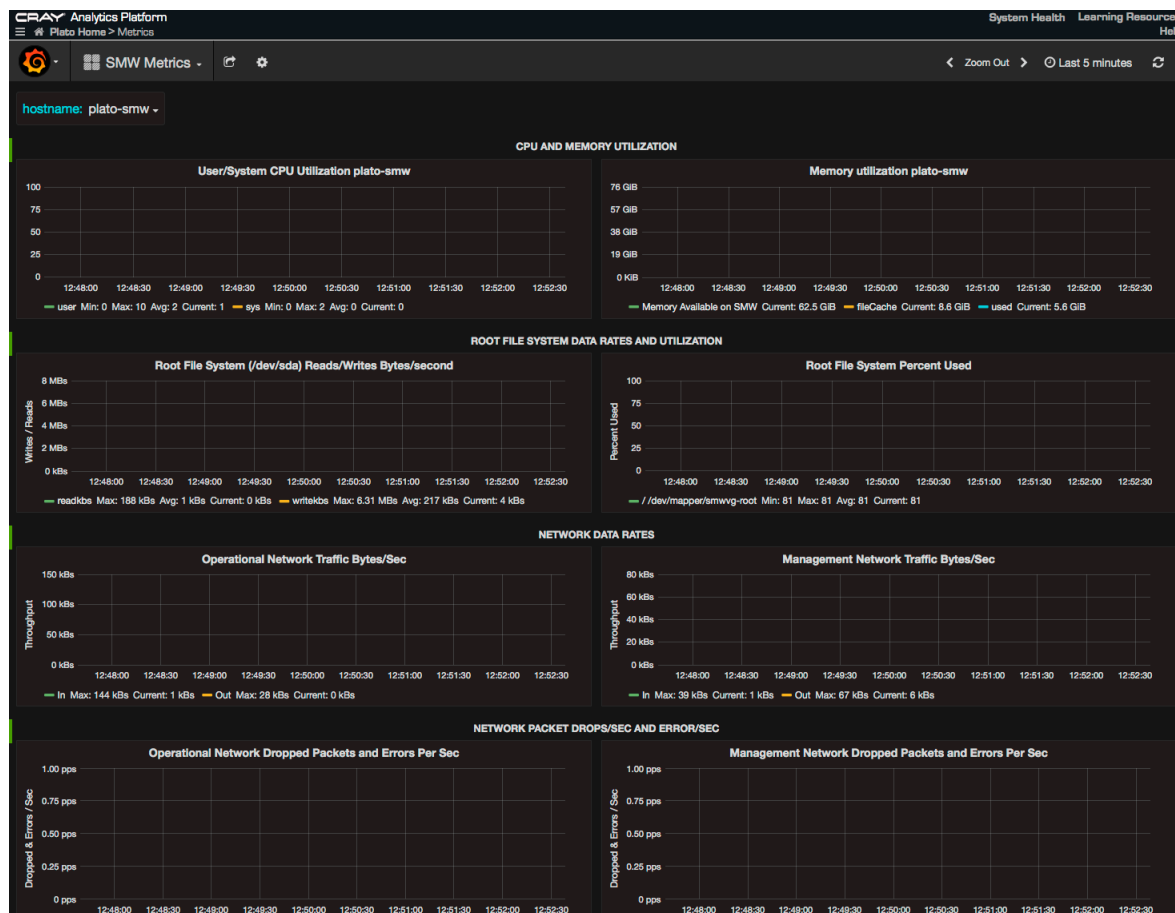
- **SMW Metrics/** - Displays graphs representing statistical data related to SMW's resources, such as the CPU's memory utilization, root file system, etc.

This dashboard contains the following sections:

- **CPU and MEMORY UTILIZATION** - Displays the memory consumption by the SMW's CPU.
 - User/System CPU Utilization *MACHINE_NAME*
 - Memory utilization
- **Root File System Data Rates and Utilization** - Displays memory usage and data rate of the SMW's root file system.

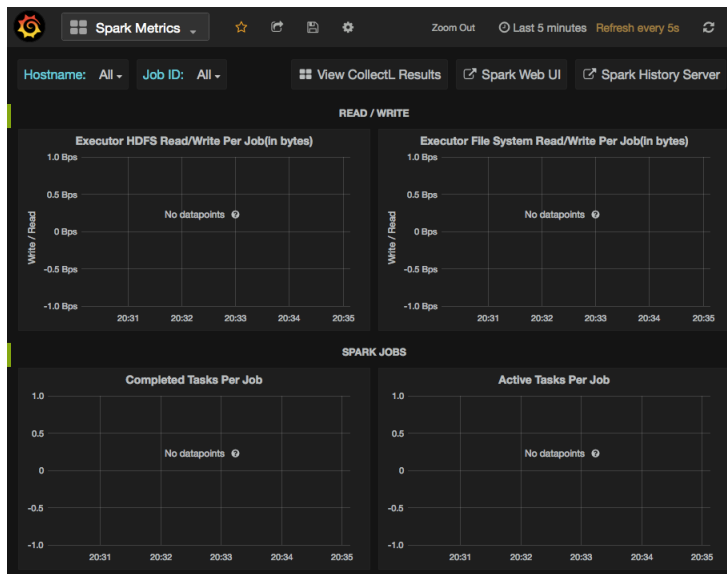
- **Root File System Hard Drive (/dev/sda) Reads/Writes Bytes/Sec** - Displays information about the usage of memory on the root file system of the SMW
- **Root File System Percent Used** - Displays the percentage for used SMW root file system space.
- **NETWORK DATA RATES**
 - **Operational Network Traffic Bytes/sec** - Displays the operational network's data rate.
 - **Management Network Traffic Bytes/sec** - Displays the management network's data rate.
- **NETWORK PACKET DROPS/SEC AND ERRORS/SEC**
 - **Operational Network Dropped and Errors Per Sec** - Displays the number of dropped packets and errors per second for the operational network.
 - **Management Network Dropped and Errors Per Sec** - Displays the number of dropped packets and errors per second for the management network.

Figure 46. SMW Metrics Dashboard



- **Spark Metrics** - Displays graphs representing statistical data related to Spark jobs. This dashboard also contains links for viewing the **Spark Web UI** and **Spark History Server**.

Figure 47. Grafana Spark Metrics Dashboard



Graphs displayed on this dashboard are grouped into the following sections:

- **READ/WRITE:** Displays statistics related to the file system statistics of a Spark executor. Results in the graphs of this section are displayed per node for a particular Spark Job. The Y-axis displays the number in bytes, whereas the X-axis displays the start/stop time of the task for a particular Spark Job.

This section contains the following graphs:

- **Executor HDFS Read/Write Per Job (in bytes):** Reading and writing from HDFS.
- **Executor File System Read/Write Per Job (in bytes):** Reading and writing from a File System.
- **SPARK JOBS:** Displays statistics related to the list of executors per node for a particular Spark Job. The Y-axis displays the number of tasks and X-axis displays the start/stop time of the task for a particular Spark Job.

This section contains the following graphs:

- **Completed Tasks Per Job:** The approximate total number of tasks that have completed execution.
- **Active Tasks Per Job:** The approximate number of threads that are actively executing tasks.
- **Current Pool Size Per Job:** The current number of threads in the pool.
- **Max Pool Size Per Job:** The maximum allowed number of threads that have ever simultaneously been in the pool.
- **DAG Scheduler** - Displays statistics related to Spark's Directed Acyclic Graphs.

This section contains the following graphs:

- **DAG Schedule Stages** - This graph displays the following types of DAG stages:
 - **Waiting Stages:** Stages with parents to be computed.
 - **Running Stages:** Stages currently being run.
 - **Failed Stages:** Stages that failed due to fetch failures (as reported by `CompletionEvents` for `FetchFailed` end reasons) and are going to be resubmitted.

- **DAG Scheduler Jobs** - This graph displays the following types of DAG scheduler jobs:
 - All Jobs - The number of all jobs
 - Active Jobs - The number of active jobs
- **DAG Scheduler Message Processing Time** - This graph displays the processing time of the DAG Scheduler.
- **JVM Memory Usage** - The memory usage dashboards represent a snapshot of used memory.
 - **JVM Memory Usage - init**: Represents the initial amount of memory (in bytes) that the Java virtual machine requests from the operating system for memory management during start up. The Java virtual machine may request additional memory from the operating system and may also release memory to the system over time. The value of `init` may be undefined.
 - **JVM Memory Usage - Used**: Represents the amount of memory currently used (in bytes).
 - **JVM Memory Usage - Committed**: Represents the amount of memory (in bytes) that is guaranteed to be available for use by the Java virtual machine. The amount of committed memory may change over time (increase or decrease). The Java virtual machine may release memory to the system and committed could be less than `init`. Committed will always be greater than or equal to used.
 - **JVM Memory Usage - Max**: Represents the maximum amount of memory (in bytes) that can be used for memory management. Its value may be undefined. The maximum amount of memory may change over time if defined. The amount of used and committed memory will always be less than or equal to max if max is defined. A memory allocation may fail if it attempts to increase the used memory such that `used > committed` even if `used <= max` is still true, for example, when the system is low on virtual memory.
 - **JVM Heap Usage**: Represents the maximum amount of memory used by the JVM heap space
 - **JVM Non-Heap Usage**: Represents the maximum amount of memory used by the JVM non-heap space

20.2 Performance Metrics Collected on Urika-GX

Table 10. CPU Metrics

Metric	Description
<code>cputotals.user</code>	Percentage of node usage utilized in user mode
<code>cputotals.nice</code>	Percentage of CPU time spent executing a process with a “nice” value
<code>cputotals.sys</code>	Percentage of CPU memory used in system mode
<code>cputotals.wait</code>	Percentage of CPU time spent in wait state
<code>cputotals.idle</code>	Percentage of CPU time spent in idle state
<code>cputotals.irq</code>	Percentage of CPU time spent processing interrupts
<code>cputotals.soft</code>	Percentage of CPU time spent processing soft interrupts
<code>cputotals.steal</code>	Percentage of CPU time spent running virtualized (always 0)

Metric	Description
ctxint.ctx	Number of context switches
ctxint.int	Number of interrupts
ctxint.proc	Number of process creations/sec
ctxint.runq	Number of processes in the Run queue
cpuload.avg1	Average CPU load over the last minute
cpuload.avg5	Average CPU load over the last 5 minutes
cpuload.avg15	Average CPU load over the last 15 minutes

Table 11. Disk Metrics

Metric	Description
disktotals.reads	Combined number of reads for all hard drives and SSD on this node
disktotals.readkbs	Combined number of KB/sec read for all hard drives and SSD on this node
disktotals.writes	Combined number of writes for all hard drives and SSD on this node
disktotals.writekbs	Combined number of KB/sec written for all hard drives and SSD on this node
diskinfo.reads.sda	Number of memory reads on system hard drive
diskinfo.readkbs.sda	KB/seconds read on system hard drive
diskinfo.writes.sda	Number of memory writes on system hard drive
diskinfo.writekbs.sda	KB/seconds written on system hard drive
diskinfo.rqst.sda	Number of IO requests (readkbs + writekbs)/(reads + writes) on system hard drive
diskinfo.qlen.sda	Average number of IO requests queued on system hard drive
diskinfo.wait.sda	Average time in msec for a request has been waiting in the queue on system hard drive
diskinfo.util.sda	Percentage of CPU time during which I/O requests were issued on system hard drive
diskinfo.time.sda	Average time in msec for a request to be serviced by the system hard drive
diskinfo.reads.sdb	Number of memory reads on second hard drive
diskinfo.readkbs.sdb	KB/seconds read on second hard drive
diskinfo.writes.sdb	Number of memory writes on second hard drive
diskinfo.writekbs.sdb	KB/seconds written on second hard drive
diskinfo.rqst.sdb	Number of IO requests (readkbs + writekbs)/(reads + writes) on second hard drive

Metric	Description
diskinfo.qlen.sdb	Average number of IO requests queued on second hard drive
diskinfo.wait.sdb	Average time in Milliseconds for a request has been waiting in the queue on second hard drive
diskinfo.time.sdb	Average time in msec for a request to be serviced by the second hard drive
diskinfo.util.sdb	Percentage of CPU time during which I/O requests were issued on second hard drive
diskinfo.reads.nvme0n1	Number of memory reads on SSD
diskinfo.readkbs.nvme0n1	KB/seconds read on SSD
diskinfo.writes.nvme0n1	Number of memory writes on SSD
diskinfo.writekbs.nvme0n1	KB/seconds written on SSD
diskinfo.rqst.nvme0n1	Number of IO requests (readkbs + writekbs)/(reads + writes) on SSD
diskinfo.qlen.nvme0n1	Average number of IO requests queued on SSD
diskinfo.wait.nvme0n1	Average time in msec for a request has been waiting in the queue on SSD
diskinfo.util.nvme0n1	Percentage of CPU time during which I/O requests were issued on SSD
diskinfo.time.nvme0n1	Average time in msec for a request to be serviced by the SSD

Table 12. Memory Metrics

Metric	Description
meminfo.tot	Total node memory
meminfo.free	Unallocated node memory
meminfo.shared	Unused memory
meminfo.buf	Memory used for system buffers
meminfo.cached	Memory used for caching data between the kernel and disk, direct I/O does not use the cache
meminfo.used	Amount of used physical memory not including kernel memory
meminfo.slab	Amount of memory used for slab allocations in the kernel
meminfo.map	Amount of memory mapped by processes
meminfo.hugetot	Amount memory allocated through huge pages
meminfo.hugefree	Amount of available memory via huge pages
meminfo.hugersvd	Amount of memory reserved via huge pages
Swap memory metrics	
swapinfo.total	Total swap memory

Metric	Description
swapinfo.free	Total available swap memory
swapinfo.used	Amount of swap memory used
swapinfo.in	Kilobytes of swapped memory coming in per second
swapinfo.out	Kilobytes of swapped memory going out per second
Page memory metrics	
pageinfo.fault	Page faults/sec resolved by not going to disk
pageinfo.majfault	These page faults are resolved by going to disk
pageinfo.in	Total number of pages read by block devices
pageinfo.out	Total number of pages written by block devices

Table 13. Socket Metrics

Metric	Description
sockinfo.used	Total number if socket allocated which can include additional types such as domain
sockinfo.tcp	Total TCP sockets currently in use
sockinfo.orphan	Number of TCP orphaned connections
sockinfo.alloc	TCP sockets allocated
sockinfo.mem	Number of pages allocated by TCP sockets
sockinfo.udp	Total UDP sockets currently in use
sockinfo.tw	Number of connections in TIME_WAIT
sockinfo.raw	Number of RAW connections in use
sockinfo.frag	Number of fragment connections
sockinfo.fragm	Memory in bytes

Table 14. Lustre Metrics

Metric	Description
lusctl.reads	Number of reads by Lustre clients
lusctl.readkbs	Number of reads in KB by Lustre clients
lusctl.writes	Number of writes by Lustre clients
lusctl.writekbs	Number of writes in KB by Lustre clients
lusctl.numfs	Number of Lustre file systems

20.3 Default Log Settings

The following table lists the default log levels of various Urika-GX analytic components. If a restart of the service is needed, please first stop services using the `urika-stop` command, change the log level, and then restart services using the `urika-start` command.

Table 15. Default Log Levels

Component	Default Log Level	Restarting service required after changing log level?
Spark	Default log levels are controlled by the <code>/opt/cray/spark/default/conf/log4j.properties</code> file. Default Spark settings are used when the system is installed, but can be customized by creating a new <code>log4j.properties</code> file. A template for this can be found in the <code>log4j.properties.template</code> file.	No
Hadoop	Default log levels are controlled by the <code>log4j.properties</code> file. Default Hadoop settings are used when the system is installed, but can be customized by editing the <code>log4j.properties</code> file.	Yes
Mesos	INFO	Yes
Marathon	INFO. Log levels can be modified by editing the <code>log4j.properties</code> file.	Yes
Grafana	INFO. Log properties can be modified by editing the <code>/etc/grafana/grafana.ini</code> file.	Yes
Jupyter Notebook	Log levels are controlled by the <code>Application.log_level</code> configuration parameter in <code>/etc/jupyterhub/jupyterhub_config.py</code> . It is set to 30 by default.	Yes
Cray Graph Engine (CGE)	INFO. The <code>log-reconfigure --log-level number</code> command can be used to modify the log level. Use the drop down on the CGE UI to set the log level for the specific action being performed, i.e. <code>query</code> , <code>update</code> or <code>checkpoint</code> . Use the drop down on the Edit Server Configuration page to set the log level. Changing the log level in this manner persists until CGE is shut down.	No. Restarting CGE reverts the log level to INFO
Flex scripts: <ul style="list-style-type: none"> <code>urika-yam-status</code> <code>urika-yam-flexup</code> <code>urika-yam-flexdown</code> 	INFO. Changing the log level for these scripts is not supported.	NA

Component	Default Log Level	Restarting service required after changing log level?
• <code>urika-yam-flexdown-all</code>		
Spark Thrift server	INFO.	Yes
HiverServer2	INFO.	Yes
Tenant proxy logs	DEBUG	No
Urika-GX security manager logs	INFO	No

20.4 Tunable Hadoop and Spark Configuration Parameters

This section lists the supported list of Hadoop and Spark parameters on the system.

Hadoop

Before tuning any Hadoop configuration parameters, services should be stopped via the `urika-stop` command. After the parameters have been changed, services should be started using the `urika-start` command.

- **MapReduce Configuration Parameters** - Common configuration parameters (along with default values on Urika-GX) that can be tuned in the `mapred-site.xml` file include:
 - `mapreduce.am.max-attempts` (defaults to 2)
 - `mapreduce.job.counters.max` (defaults to 130)
 - `mapreduce.job.reduce.slowstart.completedmaps` (defaults to 0.05)
 - `mapreduce.map.java.opts` (defaults to `-Xmx8192m`)
 - `mapreduce.map.log.level` (defaults to INFO)
 - `mapreduce.map.memory.mb` (defaults to 10240)
 - `mapreduce.map.output.compress` (defaults to false)
 - `mapreduce.map.sort.spill.percent` (defaults to 0.7)
 - `mapreduce.output.fileoutputformat.compress` (defaults to false)
 - `mapreduce.output.fileoutputformat.compress.type` (defaults to BLOCK)
 - `mapreduce.reduce.input.buffer.percent` (defaults to 0.0)
 - `mapreduce.reduce.java.opts` (defaults to `-Xmx8192m`)
 - `mapreduce.reduce.log.level` (defaults to INFO)
 - `mapreduce.reduce.shuffle.fetch.retry.enabled` (defaults to 1)
 - `mapreduce.reduce.shuffle.fetch.retry.interval-ms` (defaults to 1000)
 - `mapreduce.reduce.shuffle.fetch.retry.timeout-ms` (defaults to 30000)

- `mapreduce.reduce.shuffle.input.buffer.percent` (defaults to 0.7)
- `mapreduce.reduce.shuffle.merge.percent` (defaults to 0.66)
- `mapreduce.reduce.shuffle.parallelcopies` (defaults to 30)
- `mapreduce.task.io.sort.factor` (defaults to 100)
- `mapreduce.task.io.sort.mb` (defaults to 1792)
- `mapreduce.task.timeout` (defaults to 300000)
- `yarn.app.mapreduce.am.log.level` (defaults to INFO)
- `yarn.app.mapreduce.am.resource.mb` (defaults to 10240)
- **YARN Configuration Parameters** - Common configuration parameters (along with default values on Urika-GX) that can be tuned in the `yarn-site.xml` file include:
 - `yarn.nodemanager.container-monitor.interval-ms` (defaults to 3000)
 - `yarn.resourcemanager.am.max-attempts` (defaults to 2)
 - `yarn.nodemanager.health-checker.script.timeout-ms` (defaults to 60000)
 - `yarn.scheduler.maximum-allocation-mb` (defaults to 225280)
 - `yarn.nodemanager.vmem-pmem-ratio` (defaults to < 2.1)
 - `yarn.nodemanager.delete.debug-delay-sec` (defaults to 0)
 - `yarn.nodemanager.health-checker.interval-ms` (defaults to 135000)
 - `yarn.nodemanager.resource.memory-mb` (defaults to 225280)
 - `yarn.nodemanager.resource.percentage-physical-cpu-limit` (defaults to 80)
 - `yarn.nodemanager.log.retain-second` (defaults to 604800)
 - `yarn.resourcemanager.zk-num-retries` (defaults to 1000)
 - `yarn.resourcemanager.zk-retry-interval-ms` (defaults to 1000)
 - `yarn.resourcemanager.zk-timeout-ms` (defaults to 10000)
 - `yarn.scheduler.minimum-allocation-mb` (defaults to 10240)
 - `yarn.scheduler.maximum-allocation-vcores` (defaults to 19)
 - `yarn.scheduler.minimum-allocation-vcores` (defaults to 1)
 - `yarn.log-aggregation.retain-seconds` (defaults to 2592000)
 - `yarn.nodemanager.disk-health-checker.min-healthy-disks` (defaults to 0.25)

Spark

Cray Spark configuration defaults are chosen based on internal verification and benchmarking to provide strong performance across a range of workloads. These defaults may not be ideal for all workloads and thus, there may be a need to change configuration settings. This can be done on a per-job basis to optimize performance of individual workloads.

Users can change Spark configuration parameters when they launch jobs using the `--conf` command line flag.

Common Spark configuration parameters (along with default values on Urika-GX) that can be tuned in the `spark-default.conf` file include:

- `spark.cores.max` (defaults to 32 cores). For more information about changing this default, see [Modify the Default Number of Maximum Spark Cores](#) on page 31
- `spark.executor.memory` (defaults to 96g)
- `spark.shuffle.compress` (defaults to false)
- `spark.locality.wait` (defaults to 1)
- `spark.eventLog.enabled` (defaults to true)
- `spark.driver.memory` (defaults to 16g)
- `spark.driver.maxResultSize` (defaults to 1g)
- `spark.serializer` (defaults to `org.apache.spark.serializer.JavaSerializer`)
- `spark.storage.memoryFraction` (defaults to 0.6)
- `spark.shuffle.memoryFraction` (defaults to 0.2)
- `spark.speculation` (defaults to false)
- `spark.speculation.multiplier` (defaults to 1.5)
- `spark.speculation.quantile` (defaults to 0.75)
- `spark.task.maxFailures` (defaults to 4)
- `spark.app.name` (default value: none)

Refer to online documentation for a description of these parameters.

20.5 Node Types

Urika-GX contains the following types of nodes:

- **1 sub-rack system:**
 - I/O nodes: `nid00007` and `nid00015`
 - Login nodes: `nid00006` and `nid00014` (login node1) login node 1 and 2 respectively.
 - Service nodes: `nid00000`, `nid00001` and `nid00002`
 - Compute nodes: all the remaining nodes.
- **2 sub-rack system:**
 - I/O nodes: `nid00015` and `nid00031`
 - Login nodes: `nid00014` (login node1) and `nid00030` (login node2)
 - Service nodes: `nid00000`, `nid00001` and `nid00002`
 - Compute nodes: all the remaining nodes.
- **3 sub-rack system:**
 - I/O nodes: `nid00031` and `nid00047`
 - Login nodes: `nid00030` (login node1) and `nid00046` (login node2)
 - Service nodes: `nid00000`, `nid00016` and `nid00032`
 - Compute nodes: all the remaining nodes.

20.6 Service to Node Mapping

The list of services available for use depends on the security mode the system is running under. For more information, refer to [Urika-GX Service Modes](#) on page 7.

Table 16. Urika-GX Service to Node Mapping (2 Sub-rack System)

Node ID(s)	Service(s) Running on Node /Role of Node
nid00000	<ul style="list-style-type: none"> • ZooKeeper • ncmd • Mesos Master • Marathon • Primary HDFS NameNode • Hadoop Application Timeline Server • Collectl • nrpe • kubelet
nid000[01-07, 09-13, 17-29]	<ul style="list-style-type: none"> • Collectl • Mesos Slave • Data Node • YARN Node Manager (if running) • nrpe • kubelet
nid00008	<ul style="list-style-type: none"> • ZooKeeper • Secondary HDFS NameNode • Mesos Master • Oozie • Hive Server2 • Spark Thrift Server • Hive Metastore • WebHCat • Postgres database • Marathon • YARN Resource Manager • Collectl • nrpe

Node ID(s)	Service(s) Running on Node /Role of Node
	<ul style="list-style-type: none"> • kubelet
nid00014 (Login node 1)	<ul style="list-style-type: none"> • HUE • HAProxy • Collectl • Urika-GX Applications Interface UI • Cray Application Management UI • Jupyter Notebook • Service for flexing a YARN cluster • Documentation and Learning Resources UI • nrpe • kubelet • Kubernetes Controller
nid00015, nid00031 (I/O nodes)	These are nodes that run Lustre clients and nrpe
nid00016	<ul style="list-style-type: none"> • ZooKeeper • Mesos Master • Marathon • Hadoop Job History Server • Spark History Server • Collectl • nrpe • kubelet
nid00030 (Login node 2)	<ul style="list-style-type: none"> • HUE • HA Proxy • Collectl • Service for flexing a YARN cluster • Grafana • InfluxDB • nrpe • kubelet

Table 17. Urika-GX Service to Node Mapping (3 Sub-rack System)

Node ID(s)	Service(s) Running on Node /Role of Node
nid00000	<ul style="list-style-type: none"> • ZooKeeper

Node ID(s)	Service(s) Running on Node /Role of Node
	<ul style="list-style-type: none"> • ncmd • Mesos Master • Marathon • Primary HDFS NameNode • Hadoop Application Timeline Server • Collectl • nrpe • kubelet
nid00001-nid00015, nid00017-nid00029, nid00033-nid00045	<ul style="list-style-type: none"> • Collectl • Mesos Slave • Data Node • YARN Node Manager (if running) • nrpe • kubelet
nid00016	<ul style="list-style-type: none"> • ZooKeeper • Mesos Master • Marathon • Hadoop Job History Server • Spark History Server • Collectl • nrpe • kubelet
nid00030 (Login node 1)	<ul style="list-style-type: none"> • HUE • HA Proxy • Collectl • Urika-GX Applications Interface UI • Jupyter Notebook • Service for flexing a YARN cluster • Documentation and Learning Resources UI • nrpe • kubelet • Kubernetes Controller
nid00031, nid00047 (I/O nodes)	<p>These are nodes that run Lustre clients</p> <ul style="list-style-type: none"> • kubelet

Node ID(s)	Service(s) Running on Node /Role of Node
nid00032	<ul style="list-style-type: none"> • ZooKeeper • Secondary NameNode • Mesos Master • Oozie • Hive Server2 • Hive Metastore • WebHcat • Postgres database • Marathon • YARN Resource Manager • Collectl • Spark Thrift Server • nrpe • kubelet
nid00046 (Login node 2)	<ul style="list-style-type: none"> • HUE • HA Proxy • Collectl • Grafana • InfluxDB • Service for flexing a YARN cluster • nrpe • kubelet

For additional information, use the `urika-inventory` command as root on the SMW to view the list of services running on node, as shown in the following example:

```
# urika-inventory
```

For more information, see the `urika-inventory` man page.

20.7 Port Assignments

Table 18. Services Running on the System Management Workstation (SMW)

Service Name	Default Port
SSH	22

Table 19. Services Running on the I/O Nodes

Service Name	Default Port
SSH	22

Table 20. Services Running on the Compute Nodes

Service Name	Default Port
ssh	22
YARN Node Managers	8040, 8042, 45454, and 13562
Mesos slaves on all compute nodes	5051
DataNode Web UI to access the status, logs and other information	50075
DataNode use for data transfers	50010
DataNode used for metadata operations.	8010

Table 21. Services Accessible via the Login Nodes via the Hostname

Service	Default Port
Mesos Master UI	5050. This UI is user-visible.
Spark History Server's web UI	18080. This UI is user-visible.
HDFS NameNode UI for viewing health information	50070. This UI is user-visible.
Secondary NameNode web UI	50090. This UI is user-visible.
Web UI for Hadoop Application Timeline Server	8188. This UI is user-visible.
YARN Resource Manager web UI	8088. This UI is user-visible.
Marathon web UI	8080. This UI is user-visible.
Hive Server2	SSL - 29207 Non-SSL - 10000
Hive Metastore	9083.
Hive WebHCat	50111.
Oozie server	11000. The Oozie dashboard UI runs on this port and is user-visible.
Hadoop Job History Server	19888 on nid00016. This is a user-visible web UI.

Service	Default Port
HUE server	8888 on login1 and login2. The web UI for the HUE dashboard runs on this port and is user-visible.
CGE <code>cge-launch</code> command	3750. See S-3010, " <i>Cray® Graph Engine Users Guide</i> " for more information about the <code>cge-launch</code> command or see the <code>cge-launch</code> man page.
CGE Web UI and SPARQL endpoints	3756
Spark Web UI	4040. This port is valid only when a Spark job is running. If the port is already in use, the port number's value is incremented until an open port is found. Spark Web UI runs on whichever login node (1 or 2) that the user executes <code>spark-submit/spark-shell/spark-sql/pyspark</code> on. This UI is user-visible.
InfluxDB	8086 on login2. InfluxDB runs on nid00046 on three sub-rack, and on nid00030 on a two sub-rack system.
InfluxDB port for listening for <code>collectl</code> daemons on compute nodes	2003. InfluxDB runs on login node 2 on the Urika-GX system.
InfluxDB cluster communication	8084
Grafana	3000 on login2 (login node 2). The Grafana UI is a user-visible.
Web UI for Jupyter Notebook	7800. Jupyter Notebook internally uses HTTP proxy, which listens to ports 7881 and 7882
Urika-GX Applications Interface	80 on login1 (login node 1).
Urika-GX Application Management	80 on login1 (login node 1).
Spark SQL Thrift Server	SSL - 29208 Non-SSL - 10015
Kubernetes Master	6443 on login1 (login node 1).

Additional Ports and Services

Table 22. Additional Services and Ports They Run on

Service	Port
Apache ZooKeeper	2181
Kafka (not configured by default)	9092
Flume (not configured by default)	41414
Port for SSH	22

20.8 Major Software Components Versions

Table 23. Software Component Versions

Component	Version
Ant	1.9.2
Cray Graph Engine	3.2UP03
collectl	3.7.4
Cobbler	2.6
Docker	1.12.3
emacs editor	24.3.1
Environment modules (software for switching versions of other packages)	3.2.10
gcc (C/C++ compiler)	4.8.5
glibc (GNU C libraries)	2.17
Git (version control tool)	1.8.3.1
Grafana	3.0.1
HAProxy	1.5.18
InfluxDB	0.12.2
Java runtime execution environment	OpenJDK 1.8
Java development environment	OpenJDK 1.8
Jupyter Hub	0.6.1
Jupyter	4.2.0
Jupyter Notebook	4.2.3
kdump	3.10.0
Kubernetes	1.9.2
Lustre server (on I/O nodes) and Lustre client (on all nodes) software	2.7.1
Marathon	1.1.1
Maven	3.3.9
Apache Mesos	1.1.0
Nagios	4.3.4
mrunit	2.7

Component	Version
Python language interpreter	2.7.5, 3.4.3, and Anaconda Python 3.5.2 (via Anaconda distribution version 4.1.1)
R language interpreter	3.4.3
SBT	0.13.9
Scala compiler	2.11.8
Apache Spark	Cray customized version of the Spark on Kubernetes project, which is based on Spark 2.2.0.
vi editor	VIM 7.4
Operating System	
Operating system on system nodes and SMW	CentOS 7.4
Hadoop and Hadoop ecosystem components	
In the following list, items marked with a * are installed but not configured on the Urika-GX system.	
Apache Hadoop	2.7.3
HUE	3.10.0
HDFS	2.7.3
Hortonworks Data Platform (HDP)	2.6.1.0-129
*Flume	1.5.2
Hive	1.2.1
*Kafka	0.9.0
*Mahout	0.9.0
Oozie	4.2.0
*Sqoop	1.4.6
*Pig	0.16.0
ZooKeeper	3.4.6

gdb (Debugger for C/C++ programs) is not installed on the system by default, but can be installed by administrators via YUM if required, as shown in the following example:

```
# yum install gdb
```

For additional information, execute the `urika-rev` and `urika-inventory` commands as root from the SMW, as shown in the following examples:

- `# urika-rev`
- `# urika-inventory`

21 Troubleshooting

21.1 Diagnose and Troubleshoot Orphaned Mesos Tasks

Prerequisites

This procedure requires root access and the system to be running in the default service mode. Execute the `urika-state` command to ensure that Mesos is running and that the system is operating in the default service mode.

About this task

The metrics displayed in Mesos UI can also be retrieved using CURL calls. Cray-developed scripts (for flexing up a YARN sub-cluster) and `mrunch` use these curl calls in as they interoperate with Mesos for resource brokering. If the metrics displayed by Mesos UI and the metrics that the curl calls return different results Mesos may not work correctly and all the Mesos frameworks will be affected. As such, the aforementioned Cray-developed scripts and `mrunch` will not be able to retrieve the needed resources. This behavior can be identified when:

- there is a disconnect between the CURL calls and the Mesos UI. Specifically, there will be an indication of orphaned Mesos tasks if the CURL call returns a higher number of CPUs used than that returned by the UI. Cray-developed scripts for flexing YARN sub-clusters use curl calls, and hence do not allow flexing up if there are not enough resources reported.
- there are orphaned Mesos tasks, as indicated in the Mesos Master and Mesos Slave logs at `/var/log/mesos`. Mesos Master will reject task status updates because it will not recognize the framework those tasks are being sent from.

If this behavior is encountered, follow the instructions listed in this procedure:

Procedure

1. Log on to the System Management Workstation (SMW) as root
2. Clear the slave meta data on all the nodes with Mesos slave processes running

The following example can be used on a 3 sub-rack system:

```
# pdsh -w nid000[00-47] -x nid000[00,16,30,31,32,46,47] \
  'rm -vf /var/log/mesos/agent/meta/slaves/latest'
```

3. Stop the cluster

```
# urika-stop
```

4. Start the cluster

```
# urika-start
```

After following the aforementioned steps, the system should be restored to its original state. For additional information, contact Cray Support.

21.2 Analytic Applications Log File Locations

Log files for a given service are located on the node(s) the respective service is running on, which can be identified using the `urika-inventory` command. For more information, see the `urika-inventory` man page.

Table 24. Analytics Applications Log File Locations

Application/Script	Log File Location
Mesos	<code>/var/log/mesos</code>
Marathon	<code>/var/log/messages</code>
HA Proxy	<code>/var/log/haproxy.log</code>
Mesos frameworks: <ul style="list-style-type: none"> Marathon Spark 	<code>/var/log/mesos/agent/slaves/</code> . Within this directory, a framework's output is placed in files called <code>stdout</code> and <code>stderr</code> , in a directory of the form <code>slave-X/fw-Y/Z</code> , where <code>X</code> is the slave ID, <code>Y</code> is the framework ID, and multiple subdirectories <code>Z</code> are created for each attempt to run an executor for the framework. These files can also be accessed via the web UI of the slave daemon. The location of the Spark logs is determined by the cluster resource manager that it runs under, which is Mesos on Urika-GX.
Grafana	<code>/var/log/grafana/grafana.log</code>
InfluxDB	<code>/var/log/influxdb/influxd.log</code>
collectl	collectl does not produce any logging information. It uses logging as a mechanism for storing metrics. These metrics are exported to InfluxDB. If collectl fails at service start time, the cause can be identified by executing the <code>collectl</code> command on the command line and observing what gets printed. It will not complain if the InfluxDB socket is not available.
Hadoop	<p>The following daemon logs appear on the node they are running on:</p> <ul style="list-style-type: none"> <code>/var/log/hadoop/hdfs/hadoop-hdfs-namenode-<i>nid</i>.log</code> <code>/var/log/hadoop/hdfs/hadoop-hdfs-datanode-<i>nid</i>.log</code> <code>/var/log/hadoop/yarn/yarn-yarn-nodemanager-<i>nid</i>.log</code> <code>/var/log/hadoop/yarn/yarn-yarn-resourcemanager-<i>nid</i>.log</code> <p>In the above locations, <i>nid</i> is used as an example for the node name.</p> <p>Application specific logs reside in HDFS at <code>/app-logs</code></p>
Spark	<ul style="list-style-type: none"> Spark event logs (used by the Spark History server) reside at: <code>hdfs://user/spark/applicationHistory</code>

Application/Script	Log File Location
	<ul style="list-style-type: none"> Spark executor logs (useful to debug Spark applications) reside with the other Mesos framework logs on the individual compute nodes (see above) at: <code>/var/log/mesos/agent/slaves/</code>
Jupyter Notebook	<code>/var/log/jupyterhub.log</code>
Flex scripts: <ul style="list-style-type: none"> <code>urika-yam-status</code> <code>urika-yam-flexdown</code> <code>urika-yam-flexdown-all</code> <code>urika-yam-flexup</code> 	<code>/var/log/urika-yam.log</code>
ZooKeeper	<code>/var/log/zookeeper</code>
Hive Metastore	<code>/var/log/hive</code>
HiveServer2	<code>/var/log/hive</code>
HUE	<code>/var/log/hue</code>
Spark Thrift Server	<code>/var/log/spark</code>

Spark Audit Logs

A per-user Spark audit log that details start and stop of applications is located at `/var/log/spark/k8s/username.log` with entries of the following form:

```
Tue Apr 03 07:54:05 CDT 2018 username spark-test-1522760043061-driver START \
Application Started with 1 driver plus 5.0 executors using 6.0 cores and 496.0GB memory
Tue Apr 03 07:54:38 CDT 2018 username spark-test-1522760043061-driver STOP \
Application Stopped
Tue Apr  3 08:15:51 CDT 2018 username username-shell-159738-5d5b87c8b-82td6 \
START spark-shell Shell Started with 1 driver using 16.0 cores and 60GB memory
Tue Apr  3 08:16:36 CDT 2018 username username-shell-159738-5d5b87c8b-82td6 \
STOP spark-shell Shell Stopped
Wed Apr 04 04:28:45 CDT 2018 username spark-test-1522834122688-driver START \
Application Started with 1 driver plus 7.0 executors using 8.0 cores and 688.0GB memory
Wed Apr 04 04:29:38 CDT 2018 username spark-test-1522834122688-driver STOP \
Application Stopped
Wed Apr 04 04:30:09 CDT 2018 username spark-test-1522834207396-driver START \
Application Started with 1 driver plus 2.0 executors using 3.0 cores and 208.0GB memory
Wed Apr 04 04:30:42 CDT 2018 username spark-test-1522834207396-driver STOP \
Application Stopped
Wed Apr 04 04:32:05 CDT 2018 username spark-test-1522834323513-driver START \
Application Started with 1 driver plus 2.0 executors using 17.0 cores and 208.0GB memory
Wed Apr 04 04:32:28 CDT 2018 username spark-test-1522834323513-driver STOP \
Application Stopped
```

These log files will be located on whatever node an application is submitted from, typically `login1`, though maybe elsewhere depending how the system enables users to access the system.

This log has the general format `date username driver-pod-name action message`, where:

- `driver-pod-name` is the Kubernetes pod name that is the driver for the application that can be used to link this information to more detailed information from Kubernetes.
- `action` is either `START` or `STOP`
- `message` contains informational content, such as resources requested by the users application.



CAUTION: In the event of a failed or cancelled job (i.e. user executes a Ctrl+C/kill on the job) there may be no corresponding `STOP` event registered for a job reported as started.

21.3 Clean Up Log Data

As jobs are executed on the system, a number of logs are generated, which need to be cleaned up, otherwise they may consume unnecessary space. Log data is useful for debugging issues, but if it is certain that this data is no longer needed, it can be deleted.

- **Mesos logs** - Mesos logs are stored under `var/log/mesos`, whereas the Mesos framework logs are stored under `/var/log/mesos/agent/slaves`. These logs need to be deleted manually.
- **Marathon logs** - Marathon logs are stored under `var/log/message` and need to be deleted manually.
- **HA Proxy logs** - HA Proxy logs are stored under `var/log/message` and need to be deleted manually.
- **Jupyter logs** - Jupyter log file are located at `var/log/jupyterhub/jupyterhub.log` and need to be deleted manually.
- **Grafana and InfluxDB logs** - Grafana logs are stored under `var/log/grafana`, whereas InfluxDB logs are stored under `var/log/influxdb`. Influxdb log files are compressed. Both Grafana and InfluxDB use the `logrotate` utility to keep log files from using too much space. Log files are rolled daily by default, but if space is critical, logs can be deleted manually.
- **Spark logs** - Shuffle data files on the SSDs is automatically deleted on Urika-GX. Spark logs need to be deleted manually and are located at the following locations:
 - Spark event logs - Located at `hdfs://user/spark/applicationHistory`
 - Spark executor logs - Located on individual compute nodes at `/var/log/mesos/agent/slaves/`
- **Hadoop logs** - Hadoop log files are located in the following locations and need to be deleted manually:
 - Core Hadoop - Log files are generated under the following locations:
 - `var/log/hadoop/hdfs`
 - `var/log/hadoop/yarn`
 - `var/log/hadoop/mapreduce`
 - ZooKeeper - ZooKeeper logs are generated under `var/log/zookeeper`
 - Hive (metastore and hive server2) - These logs are generated under `var/log/hive`
 - Hive Webhcat - These logs are generated under `var/log/webhcat`
 - Oozie - Oozie logs are stored under `/var/log/oozie`
 - HUE - HUE logs are generated under `/var/log/hue`
- **Flex scripts (urika-yam-status, urika-yam-flexup, urika-yam-flexdown, urika-yam-flexdown-all)** - These scripts generate log files under `/var/log/urika-yam.log` and need to be deleted manually.
- **mrunch** - `mrunch` does not generate logs.
- **Cray Graph Engine (CGE) logs** - The path where CGE log files are located is specified via the `-l` parameter of the `cge-launch` command. Use the `cge-cli log-reconfigure` command to change the location after CGE is started with `cge-launch`. CGE logs need to be deleted manually. Users can also use `--log-level`

argument to CGE CLI commands to set the log level on a per request basis. In addition, the `cge.server.DefaultLogLevel` parameter in the `cge.properties` file can be used to set the log level to the desired default.

21.4 Ensure Long Running Spark Jobs Finish Executing

Prerequisites

This procedure requires root privileges and needs to be carried out on SMW node.

About this task

Delegation tokens are acquired from the HDFS NameNode when a job starts executing. These tokens need to be renewed after they have expired. The default expiration period/lifetime of delegation tokens is 1 day. Spark jobs that interact with HDFS can successfully complete if the execution time is less than the delegation token's lifetime. The lifetimes of the tokens can be modified by changing certain properties in the `hdfs-site.xml` file, as described in this procedure.

Procedure

1. Log on to the SMW as root.
2. Stop the HDFS service.

```
# urika-stop -s hdfs
```

3. Edit the `hdfs-site.xml` file to set the following values greater than or equal to the Spark Job execution time.
 - `dfs.namenode.delegation.key.update-interval`
 - `dfs.namenode.delegation.token.max-lifetime`
 - `dfs.namenode.delegation.token.renew-interval`
4. Copy the modified `hdfs-site.xml` file to all the nodes.
5. Restart the HDFS service.

```
# urika-start -s hdfs
```

21.5 Troubleshoot Common Analytic and System Management Issues

The following table contains a list of some common error messages and their description. Please note that this is not an exhaustive list. Online documentation and logs should be referenced for additional debugging/troubleshooting. For a list of Cray Graph Engine error messages and troubleshooting information, please refer to the *Cray® Graph Engine User Guide*.

Table 25. System Management Error Messages

Error Message	Description	Notes/Resolution
ERROR: unauthorized command 'cat' requested by client	This message is returned when a restricted user, logged in to a tenant VM, attempts to execute a command that is not part of set of the white listed commands.	Only white listed commands can be executed by restricted users who are logged into tenant VMs. For more information, refer to the 'Urika®-GX System Administration Guide'.
Error message: ERROR: tag(s) not found in playbook: non_existent_service. possible values: collectl, grafana, hdfs, hdfs_dn, hdfs_nn, hdfs_sn, hdp_app_timeline, hdp_hist, hive, hive2, hive_met, hive_web, hue, influxdb, jupyterhub, marathon, mesos, nodemanager, oozie, spark_hist, yarn, yarn_rm, zookeeper	Description: User has specified a service that does not exist to the urika-stop or urika-start command.	Resolution: Use the correct name of the services by selecting one of the options listed in the error message.
Error message: You are not authorized to log into this system -- to obtain access please contact your system administrator su: Permission denied	This message may be returned by the urika-state command on a freshly deployed system if the hdfs user has not yet been created.	Wait until the main_run/main_update playbooks have finished creating the hdfs user

Table 26. Spark Error Messages

Error Message	Description	Resolution
INFO mesos.CoarseMesosSchedulerBackend: Blacklisting Mesos slave 20151120-121737-1560611850-5050-20795-S0 due to too many failures; is Spark installed on it? INFO mesos.CoarseMesosSchedulerBackend: Mesos task 30 is now TASK_FAILED	There may be something preventing a Mesos slave from starting the Spark executor. Common causes include: <ul style="list-style-type: none"> The SSD is too full The user does not have permission to write to Spark temporary files under /var/spark/tmp/ 	Refer to Spark logs.
ERROR mesos.MesosCoarseGrainedSchedulerBackend: Mesos error: Master refused authentication Exiting due to error from cluster scheduler: Master refused authentication	This message appears when Spark is not able to authenticate with Mesos.	The user may not have Mesos credentials, in which case the user would need to be added via the usm-sync-users script. In other cases, this error may be a result of the user's credentials being in a bad state, in which case the

Error Message	Description	Resolution
		admin would need to run the <code>usm-recreate-secret</code> script to create a new secret for the user. For more information, refer to the 'Urika®-GX System Administration Guide'.
Lost executor # on <i>host</i>	Something has caused the Spark executor to die. One of the reasons may be that there is not enough memory allocated to the executors.	<p>Increase the memory allocated to executors via one of the following parameters:</p> <ul style="list-style-type: none"> <code>--executor-memory</code> <code>spark.executor.memory</code> configuration <p>Refer to Spark logs for additional information.</p>

Table 27. Flex Scripts Error Messages

Error Message	Description	Resolution
The number of nodes requested to flex up is greater than the total number of resources available. Please enter a valid number of nodes	The user is attempting to flex up more nodes than are available which using the <code>urika-yam-flexup</code> command.	Enter a lower number of nodes for the flex up request.
No time out specified by user through commandline argument, setting the timeout from <code>/etc/urika-yam.conf</code> file. in <code>/etc/urika-yam.conf</code> val: 15 minutes	The user has not specified a timeout while using the <code>urika-yam-flexup</code> command.	This error message can safely be ignored if it is required to use the default timeout value, which is 15 minutes. Otherwise, please specify the desired value when using the <code>urika-yam-flexup</code> command.
ID names can only contain alphanumeric, dot '.' and dash '-' characters. '@' not allowed in jhoole@#\$. Usage: <code>urika-yam-flexup --nodes #nodes --identifier name --timeout timeoutInMinutes</code>	The user has specified an incorrect identifier/application name when using the <code>urika-yam-flexup</code> command.	Reenter the command with the correct identifier.
Minimum timeout is 5 minutes. A timeout less than the minimum timeout cannot be requested, with an exception of zero timeout. Please note that you can request a zero timeout (set value of timeout to 0) by which you do not call timeout, you chose to flex down the nodes manually using <code>urika-yam-flexdown</code> . Please submit a new flex up request with valid timeout.	Incorrect minimum timeout was specified.	Submit a new flex up request with valid timeout (Request for timeout greater than minimum timeout).

Error Message	Description	Resolution
Currently only "x" nodes are available in the cluster. Please wait till the number of nodes you require are available Or submit a new flex up request with nodes less than "x"	This error is seen when the number of nodes requested to flex up is not available.	Either wait till the number of nodes required are available Or submit a new flex up request with nodes less than "x".
Invalid app name. Your app name can consist of a series of names separated by slashes. Each name must be at least 1 character. The name may only contain digits (0-9), dashes (-), dots (.), and lowercase letters (a-z). The name may not begin or end with a dash.	This error is seen when the identifier provided by user for the flex up request is invalid.	Follow the rules mentioned there and re-submit a new flex up request.
Total number of resources not set in the /etc/urika-yam.conf file, please re-check the configuration file	In /etc/urika-yam.conf file, the number of resources is set by default. The total number of resources may not have been set.	Re-check the status of mesos cluster.
Hostname is not set in the /etc/urika-yam.conf file, please re-check the configuration file.	In /etc/urika-yam.conf file, the parameter hostname is set by default. The value set may not be correct or may not have been set.	Ensure that this parameter is set, and the value is the same as default value.
Mesos port is not set in the /etc/urika-yam.conf file, please re-check the configuration file	In /etc/urika-yam.conf file, the parameter marathon_port is set by default. This parameter may not have been set or value set may not be set to the same as the default value.	Ensure that this parameter is set, and the value is the same as default value.
Marathon port is not set in the /etc/urika-yam.conf file, please re-check the configuration file.	In /etc/urika-yam.conf file, the parameter marathon_port is set by default. This parameter may not have been set or value set may not be set to the same as the default value..	It should be ensured that this parameter is set, and the value is the same as default value.
The number of nodes you requested to flex up is greater than the total number of resources available. Please enter a valid number of nodes	This error is seen when the number of nodes requested to flex up is more than the total number of nodes available in the cluster	Submit a new flex up request with nodes less than or equal to the number of nodes available in the cluster.
App '\$marathon_app_name' does not exist. Please re-check the identifier corresponding nodes you flex up, that you would like to flex down	The identifier provided for flex down does not exist.	Re-check the usage: if operating as the root user, please provide the complete name as seen in urika-yam-status or as a non-root user, ensure to provide the same identifier used at the time of flex up. In addition, check if /var/log/urika-yam.log

Error Message	Description	Resolution
		reflects any log messages where timeout criteria has been matched and there was a flex down of the app already.
Looks like there is some problem with flex up. Please try <code>urika-yam-status</code> or look at the logs to find the problem	The job failed to launch.	Review logs (stored at <code>/var/log/urika-yam.log</code> on login nodes) or execute the <code>urika-yam-status</code> command to identify if there is any problem. Please check if there are any issues related to Mesos and/or Marathon. If the Mesos and/or Marathon web UI cannot be accessed, contact the administrator, who should verify that the Mesos and Marathon daemons are up and running. If any of these daemons are not running for some reason, report the logs to Cray Support and restart the Mesos cluster using the <code>urika-start</code> command. For more information, see the <code>urika-start</code> man page.
Could not find the script <code>urika-yam-start-nodemanager</code> in <code>hdfs</code> . Looks like there is an error with your <code>urika-yam</code> installation Please contact your sysadmin	The <code>urika-yam-start-nodemanager</code> script is a component of the Cray developed scripts for flexing up a YARN cluster and is installed as part of the installation of these flex scripts.	If this issue is encountered, the administrator should verify that: <ul style="list-style-type: none"> • HDFS is in a healthy state • Marathon and Mesos services are up and running. The status of the aforementioned services can be checked using the <code>urika-state</code> command. For more information, see the <code>urika-state</code> man page. Contact support for additional information about resolving this issue.
INFO: can not flexup YAM in secure mode	This message indicates that the user is attempting to execute the <code>urika-yam-flexup</code> script while the system is in the secure service mode, instead of in the default mode.	The <code>urika-yam-flexup</code> script can only be used in the secure service mode. For more information, refer to the Urika-GX System Administration Guide.
INFO: can not flexdown YAM in secure mode	This message indicates that the user is attempting to execute the <code>urika-yam-flexdown</code> script while the system is in the secure service mode, instead of in the default mode.	The <code>urika-yam-flexdown</code> script can only be used in the secure service mode. For more information, refer to the Urika-GX System Administration Guide.

Error Message	Description	Resolution
INFO: can not flexdown-all YAM in secure mode	This message indicates that the user is attempting to execute the <code>urika-yam-flexdown-all</code> script while the system is in the secure service mode, instead of in the default mode.	The <code>urika-yam-flexdown-all</code> script can only be used in the secure service mode. For more information, refer to the Urika-GX System Administration Guide.
INFO: can not get YAM status in secure mode	This message indicates that the user is attempting to execute the <code>urika-yam-status</code> script while the system is in the secure service mode, instead of in the default mode.	The <code>urika-yam-status</code> script can only be used in the secure service mode. For more information, refer to the Urika-GX System Administration Guide.

Table 28. Marathon/Mesos/`mr` Error Messages

Error Message	Description	Resolution
<p>ERROR:mr: Force Terminated job /mr/ 2016-193-12-13-03.174056 Cancelled due to Timeout</p> <p>Examples:</p> <ul style="list-style-type: none"> • <code>error("mr: --immediate timed out while waiting")</code> • <code>error("mr: Timed out waiting for mrund : %s" % appID)</code> • <code>error("mr: Force Terminated job %s Cancelled due to Timeout" %</code> 	<p>These errors indicate timeout and resource contention issues, such as the job timed out, the machine is busy, too many users running too many jobs, a user waiting for their job to start, but previous jobs have not freed up nodes, etc. Additionally, if a user set a job timeout's to 1 hour, and the job lasted longer than 1 hour, the user would get a <code>Job Cancelled timeout error</code>.</p>	<p>Ensure that there are enough resources available and that the timeout interval is set correctly.</p>
<p>HWERR[r0s1c0n3][64]:0x4b14:The SSID received an unexpected response:Info1=0x1910000000000003:Info2=0x7</p>	<p>Mesos is not able to talk to the Zookeeper cluster and is attempting to shut itself down.</p>	<p>Restart Mesos using the <code>urika-start</code> command.</p>
<p>WARN component.AbstractLifeCycle: FAILED SelectChannelConnector@0.0.0.0:4040: java.net.BindException: Address already in use</p>	<p>User is attempting to execute a job on a port that is already in use.</p>	<p>This message can be safely ignored.</p>
<p>ERROR:Unexpected 'frameworks' data from Mesos</p> <ul style="list-style-type: none"> • Examples: <ul style="list-style-type: none"> ◦ <code>error("Mesos Response: %s" % ret)</code> 	<p>These errors occur when <code>mr</code> is not able to connect/communicate with Mesos and/or Marathon.</p>	<p>Refer to online Mesos/Marathon documentation.</p>

Error Message	Description	Resolution
<ul style="list-style-type: none"> ○ error("Unexpected 'frameworks' data from Mesos") ○ error("mrun: Getting mrund state threw exception - %s" % ○ error("getting marathon controller state threw exception - %s" % ○ error("Unexpected 'apps' data from Marathon") ○ error("mrun: Launching mrund threw exception - %s" % (str(e))) ○ error("mrun: unexpected 'app' data from Marathon: exception - %s" % (str(e))) ○ error("mrun: startMrund failed") ○ error("mrun: Exception received while waiting for " 		
<ul style="list-style-type: none"> ● error("mrun: select(): Exception %s" % str(e)) ● error("mrun: error socket") ● error("mrund: ● error %r:%s died\n" % (err,args[0])) ● error("mrund: select(): Exception %s\n" % str(e)) 	These errors may be encountered in situations where an admin physically unplugs an Ethernet cable while a CGE job was running, or a node died, etc.	Ensure that the Ethernet cable is plugged while jobs are running.
<ul style="list-style-type: none"> ● NCMD: Error leasing cookies MUNGE: ● Munge authentication failure [%s] (%s).\n 	These error only occur if the specific system services have failed.	Refer to log messages under <code>/var/log/messages</code> on the node the message was encountered on.
<p>ERROR:Not enough CPUs for exclusive access. Available: 0 Needed: 1</p> <p>Examples</p> <ul style="list-style-type: none"> ● parser.error("Only --mem_bind=local supported") ● parser.error("Only --cpu-freq=high supported") 	These errors are typically caused by user errors, typos and when not enough nodes are available to run a job.	Ensure that there are enough nodes available and there are no typos in the issues command.

Error Message	Description	Resolution
<ul style="list-style-type: none"> • <code>parser.error("Only --kill-on-bad-exit=1 supported")</code> • <code>parser.error("-n should equal (-N * --ntasks-per-node)")</code> • <code>parser.error("-N nodes must be >= 1")</code> • <code>parser.error("-n images must be >= -N nodes")</code> • <code>parser.error("No command specified to launch"); error("Not enough CPUs. ")</code> • <code>error("Not enough CPUs for exclusive access. ")</code> • <code>error("Not enough nodes. ")</code> • <code>parser.error("name [%s] must only contain 'a-z','0-9','-' and '.'")</code> • <code>parser.error("[%s] is not executable file" % args[0])</code> 		
ERROR: Zero read: Scaling DOWN not supported	This error message is returned when <code>mrunc</code> is not expecting one of the remote nodes to abruptly close its socket. When <code>mrunc</code> goes to read the socket, it detects the error, generates the message, and the application dies (or is killed).	<p>Marathon UI to scale down <code>mrunc</code> jobs is not supported.</p> <p>This message can occur in the following conditions:</p> <ul style="list-style-type: none"> • The user used the <code>--kill-on-scaledown</code> option of the <code>mrunc</code> command. • The first remote compute node did not exit successfully <p>Thus, this error message can only occur if the application is killed or if someone used the Marathon REST API in an attempt to scale the application down. No action needs to be taken in either case.</p>
<code>mrunc: error: Users may only cancel their own mrunc jobs</code>	This message comes when a non-root user tries to use <code>mrunc --cancel</code> to cancel any Marathon job that was not launched by that user using <code>mrunc</code> .	User <code>root</code> is allowed to use " <code>mrunc --cancel</code> " to kill any Marathon-started job. All other users should only kill the Marathon jobs they launched using the <code>mrunc</code> command.
INFO: Can not run this application in secure mode, cancelling app	This message indicates that the user is attempting to use <code>mrunc</code> while the system is in the secure service mode, instead of in the default mode.	<code>mrunc</code> can only be used in the default service mode. For more information, refer to the Urika-GX System Administration Guide.

Table 29. Hadoop Error Messages

Error Message	Description	Resolution
org.apache.hadoop.hdfs.server.\namenode.SafeModeException: Cannot create or delete a file. Name node is in safe mode.	During the start up, the NameNode goes into a safe mode to check for under replicated and corrupted blocks. A Safe mode for the NameNode is essentially a read-only mode for the HDFS cluster, where it does not allow any modifications to file system or blocks. Normally, the NameNode disables the safe mode automatically, however, if there are too many corrupted blocks, it may not be able to get out of the safe mode by itself.	Force the NameNode out of safe mode by running the following command as a HDFS user: <pre>\$ hdfs dfsadmin -safemode leave</pre>
Too many underreplicated blocks in the NameNode UI	Couple of dataNodes may be down. Please check the availability of all the dataNodes	If all the DataNodes are up and still there are under replicated blocks. Run the following 2 commands in order as a HDFS user: <pre>\$ hdfs fsck / grep 'Under replicated' awk -F': ' '{print \$1}' >> \ /tmp/under_replicated_files \$ for hdfsfile in `cat /tmp/ under_replicated_files`; \ do echo "Fixing \$hdfsfile : " ; \ hadoop fs -setrep 3 \$hdfsfile; \ done</pre>
Too many corrupt blocks in name node UI	The NameNode might not have access to at least one replication of the block.	Check if any of the DataNodes are down. If all the DataNodes are up and the files are no longer needed, execute the following command: <pre>\$ hdfs fsck / -delete</pre>
org.apache.hadoop.ipc.\RemoteException(java.io.IOException): \ File /tmp/test could only be replicated to \ 0 nodes instead of minReplication (=1).	HDFS space may have reached full capacity. Even though Urika-GX has a heterogeneous file system, the default storage type is DISK unless explicitly set to use SSD. The user might have filled up the default storage, which is why HDFS would not be able to write more data to DISK.	To identify the used capacity by storage type, use the following commands: For both DISK and SSD, calculate the sum of usage on all the DataNodes. For DISK: <pre>\$ df /mnt/hdd-2/hdfs/dd awk 'NR==2{print \$3}'</pre> For SSD: <pre>\$ df /mnt/ssd/hdfs/dd awk 'NR==2{print \$3}'</pre>
YARN job is not running. You can see the status of the job as ACCEPTED: waiting for AM container to be allocated, launched and register with RM.	The NodeManagers may not be running to launch the containers.	Check the number of available node managers by executing the following command: <pre>\$ yarn node -list</pre>

Additional Tips and Troubleshooting Information

- If JupyterHub processes owned by the user remain running after the user has logged out from Jupyter these processes can be manually killed using the Linux `kill` command.
- The system will return the message, "Service '*serviceName*' is not supported in the current security mode" if it is attempted to start a service that is not supported in the current service mode. Use the `urika-state` or `urika-service-mode` commands to check which service mode the system is running in. For more information, refer to [Urika-GX Service Modes](#) on page 7
- If for any reason, Marathon does not start after a system crash, as a result of the queue reaching full capacity, use the `urika-stop` command, followed by the `urika-start` command to resolve the issue.
- In Urika-GX's multi tenant environment, individual tenant members are restricted from overriding the global Hadoop configuration directory and from specifying a specific NameNode on the CLI. As such, certain arguments passed to HDFS commands on the CLI are ignored to ensure security of tenant data. If these arguments are passed to the CLI, the system will return a warning indicating that it detected an argument that is not allowed for restricted users and that the argument is being removed
- Use one of the following mechanisms if it is required to kill Spark jobs:
 - Kill the job using the Spark UI - Click on the text **(kill)** in the **Description** column of the **Stages** tab.
 - Kill the job using the Linux `kill` command.
 - Kill the job using the `Ctrl+C` keyboard keys.
- The system will return the following error if a user attempts to view help information for an unsupported Lustre `lfs` sub-command:

The *sub-command* command is either unknown or not supported for tenant users. For more information on tenant user rules try 'lfs help tenant-rules'.
- When modifying the number of CPUs or memory for a tenant VM, the system will return an error if it is attempted to allocate more than the acceptable value of CPU or memory to a tenant VM via the `ux-tenant-alter-vm` command. For more information, refer to the `ux-tenant-alter-vm` man page.
- In rare cases, switching from the secure to default mode may result in some Romana network policy information that is not translated into the appropriate IP table rules. This allows a recently created pod to ping a pod in a different Kubernetes name space. Contact Cray support if this problem is encountered.