Elastic Workload Manager User Guide

Version: 1.1

Copyright

Table of Contents

Revision History

Date	Description of Change
11/29/2016	Version 1.1.

1 Preface

This guide explains how to perform different tasks with the Jisto Elastic Workload Manager.

1.1 Audience

This guide is intended for Jisto software users that are testing and running the Elastic Workload Manager. This guide does not assume any prior knowledge of the Elastic Workload Manager, but it does assume that the reader understands basic virtualization and containerization concepts.

1.2 Software Requirements

This document makes reference to the Elastic Workload Manager Web UI and CLI. To access the UI, you must have web access to the system where the Jisto Server is installed. You must use one of the following web browsers to access the Elastic Workload Manager Web UI:

- Google Chrome version 53 or higher
- Internet Explorer 11
- Microsoft Edge Release 37 and 38
- Mozilla Firefox 48 and 49
- Google Chromium 53 and 54

To access the Elastic Workload Manager Command Line Interface (CLI), you need to be able to create a terminal window that connects to the Linux system where the Jisto Server is installed.

1.3 Document Conventions

In order to highlight certain kinds of information, this document uses the following conventions:

Table 1 – Document Conventions

Convention	Purpose	Example
`Courier`	Designates text that should be entered exactly as written in a file or field.	The file must contain the `iterations` parameter.
`Courier Italics`	Designates a variable value that you need to enter in a file or in a field.	The *`host_share_drive`* is the path of the share drive on your client Linux system.
`Courier Bold`	Designates the name of a file in text.	Invoke the **`run.sh`** command in order to verify that the exported files work as expected.
Myriad Pro Italic	Designates a concept or term that is being defined in text.	The *Jisto Server* is the Linux system where the Elastic Workload Manager is installed.

Convention	Purpose	Example
Myriad Pro	Designates a tab, menu option or other element in a user interface.	To do this, click the **Live Monitoring** tab.

2 Basic Concepts and Uses

2.1 Concepts

Before using the Elastic Workload Manager, there are a few concepts that you should understand. Since these concepts are directly related to the operations that occur on the different systems, it may be helpful to look at Figure 1 before they are discussed.

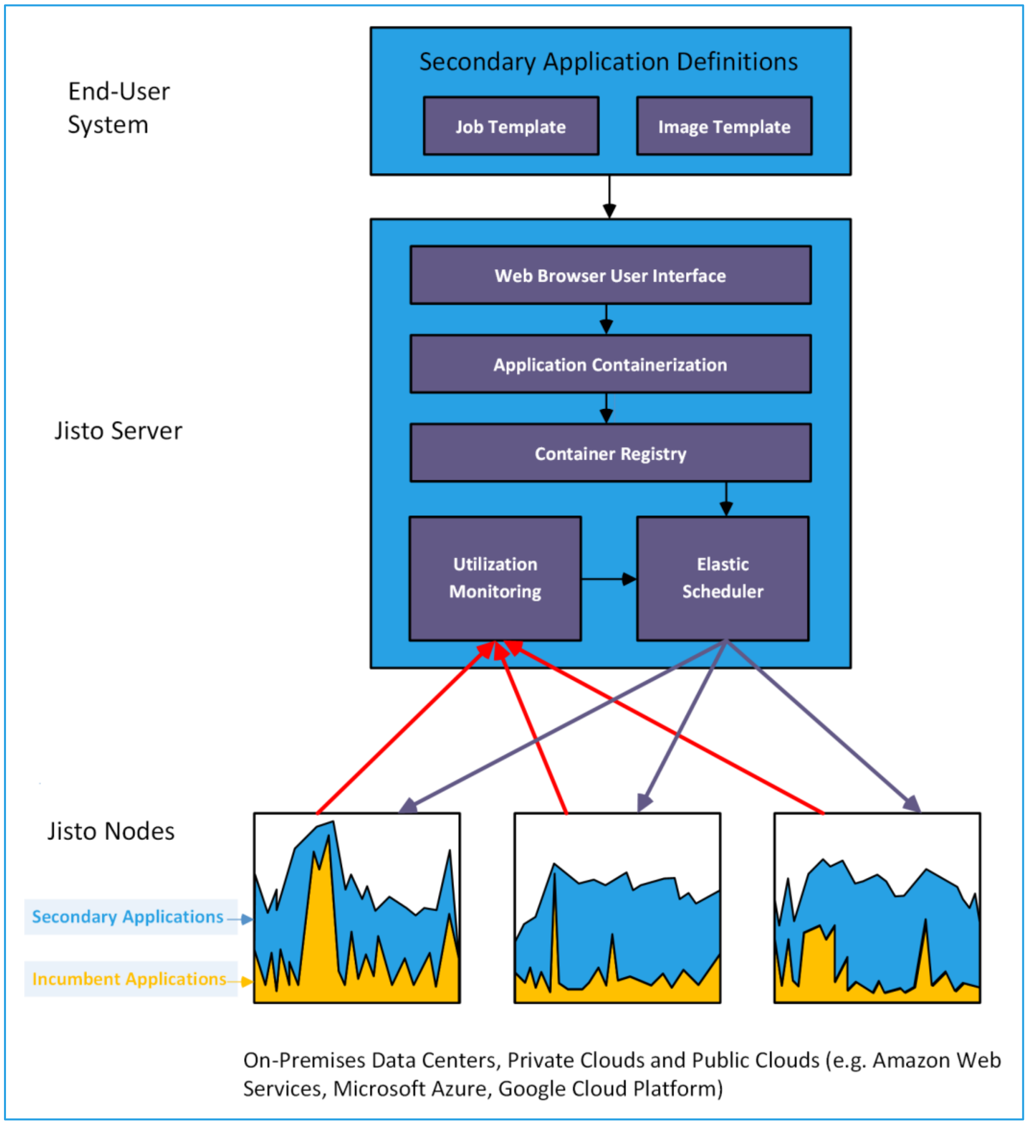

End-User System
Secondary Application Definitions
Job Template
Image Template
Jisto Server
Web Browser User Interface
Application Containerization
Container Registry
Utilization Monitoring
Elastic Scheduler
Jisto Nodes
Secondary Applications
Incumbent Applications
On-Premises Data Centers, Private Clouds and Public Clouds (e.g. Amazon Web Services, Microsoft Azure, Google Cloud Platform)

Figure 1 – Jisto System Components

2.1.1 Secondary Application Definition Concepts

A *secondary application definition*, as its name indicates, consists of files that define an application that will be transferred for processing to a *Jisto node*. The definition consists of two files – an *image template* and a *job template*. The creation of these two files takes place outside of the Elastic Workload Manager on the end-user's Linux system.

An *image template* is a set of application-related files. In addition to the actual application file, an image template may contain build and run scripts, RPM files, sub-directories and other kinds of files. All the files that are part of an image template must be in a tar.gz file that is uploaded via the web browser user interface to the Elastic Workload Manager.

A *job template* specifies the number of times or iterations that the associated secondary application must run. It may also optionally contain parameters that specify the ports and shared drives that the application can access on the Jisto node. As with the image template, the job template must be in a tar.gz file before you can upload it via the web browser user interface to the Elastic Workload Manager.

NOTE: Before creating an image or job template for a secondary application, please contact Jisto at support@jisto.com for more information.

2.1.2 Jisto Server Concepts

The *Jisto Server* is the system where the Elastic Workload Manager is installed. This program consists of the following components:

- Web Browser User Interface – Enables a user to upload image templates and job templates and to monitor the resource usage on Jisto nodes.
- Application Containerization – Creates and containerizes the application image that the user specifies in an image template.
- Container Registry – Stores the containers on the Jisto server.
- Elastic Scheduler – Determines the best Jisto nodes for the deployment of containerized applications and deploys them accordingly. (See arrows from the Jisto server to the Jisto nodes in Figure 1.)
- Utilization Monitoring - Monitors the utilization data of every connected Jisto Node whether it is in an on-site data center or a private or public cloud. This is facilitated by the Jisto Agents that are installed on all Jisto Nodes. (See arrows from the Jisto nodes to the Jisto server in Figure 1.)

As noted above, the Application Containerization component creates an application image and containerizes it. A *containerized application* or simply *container* is an instance of an application that is stored on the Jisto server in the Container Registry. When a job specifies several iterations or *tasks*, the Elastic Scheduler may subsequently deploy several instances of the same container to different Jisto nodes.

2.1.3 Jisto Node Concepts

A *Jisto node* is a system on which the Jisto Server deploys *containers*. A *container* is a user-space instance in which an application can run on the Jisto node. A container has its own file system but can use the Jisto node's network ports and shared drives if so configured in the job template.

A *secondary application* is the application that runs in a container. As its name implies, this application is *seconded* or transferred to the Jisto node temporarily for processing. While the secondary application is running, it uses only those system resources that Jisto allocates. This means that the amount of CPU, memory and disk available to a secondary application may vary significantly during execution, in response to the resource needs of any incumbent application.

An *incumbent application* is any application that runs independently of Jisto on the Jisto node. Incumbent applications have a higher priority for resource utilization than secondary applications deployed in containers and their Quality of Service (QoS) is maintained. It might be helpful to think of incumbent applications as native to the system, while secondary applications are deployed only temporarily and managed by the Elastic Workload Manager.

The *Jisto agent* is software that is installed on every Jisto node. This agent monitors CPU, memory, disk and network usage. It uses this data to control the amount of system resources that are available to the secondary applications – those deployed in containers - in response to decreased or increased utilization by incumbent applications. The resource elasticity that is provided by the agent in this way results in more efficient utilization of system processing power and memory.

2.2 Uses of the Elastic Workload Manager

As explained earlier, the Elastic Workload Manager allows you to perform the following operations:

- Create jobs that specify how applications will be run
- Monitor the utilization of Jisto nodes where applications are running

The following sections walk you through these actions and where appropriate explain what happens 'behind the scenes' to make these operations possible.

2.2.1 Job Creation and Container Deployment

In order to create a job that results in container deployment, an administrator first accesses the Jisto server via the web browser UI. A series of events lead to the deployment of a container on a Jisto node. While some of these events are initiated by the end user, others occur automatically on the basis of system utilization and other criteria. In order to appreciate what the Jisto server does, you should familiarize yourself with the events that occur from image template upload to container deployment as shown in Figure 2 below.

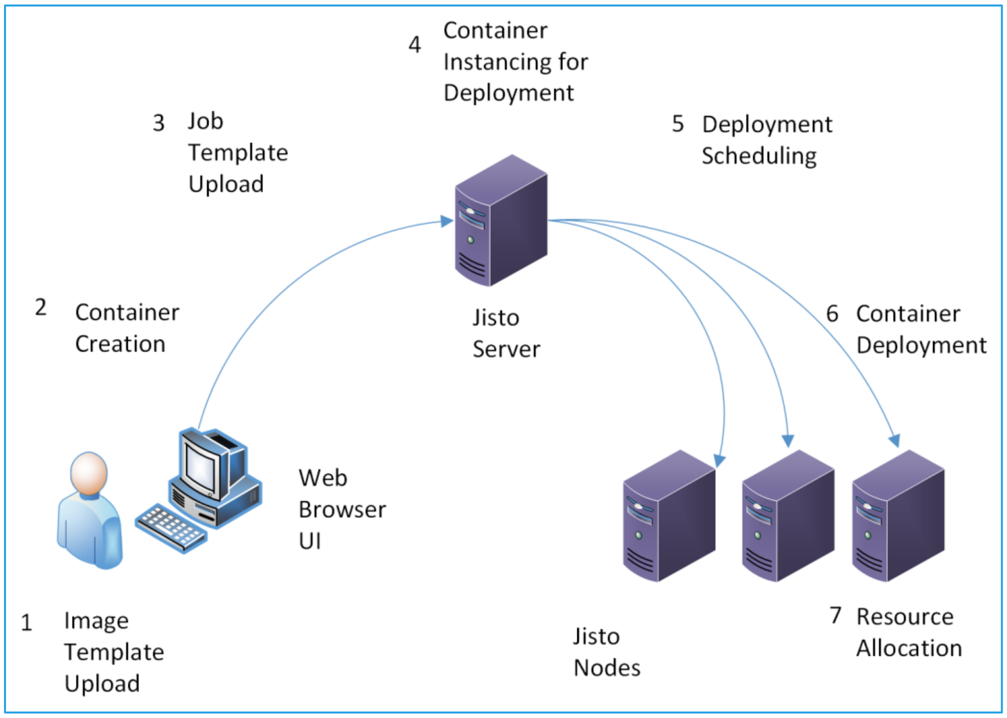


Figure 2 – Events in Container Deployment

The events illustrated above are as follows:

1. The user uploads an image template that contains application and related files through the web browser UI.
2. The Jisto server creates a container based on the uploaded image template.
3. The user uploads a job template that specifies the iterations or tasks that the container should run.
4. The Jisto Server creates a container instance for each iteration that the container should run, as specified in the job template.
5. The Jisto Server determines where to deploy the containers.
6. The Jisto Server deploys the containers to the Jisto nodes.
7. On each Jisto node, a Jisto agent allocates resources for the container - now referred to as a secondary application - in real time, in response to changes in resource utilization by incumbent applications.

2.2.2 Data Flow During Monitoring

After a Jisto agent is installed on a Jisto node, it begins to collect data on CPU and memory usage by incumbent applications and by the system as a whole. In addition, the agent gathers data on the network and disk usage. This data is transmitted to the Jisto Server but also used by the agent.

This data is needed by the agent in order to increase or decrease the system resources that are allocated to the secondary applications. It is needed by the Elastic Scheduler on the Jisto Server to help determine where to deploy containers. Finally, it is needed by the Web Browser

User Interface of the Jisto Server to provide live monitoring of resource usage on Jisto nodes.

The diagram below shows the data flow from Jisto agent to live monitoring.

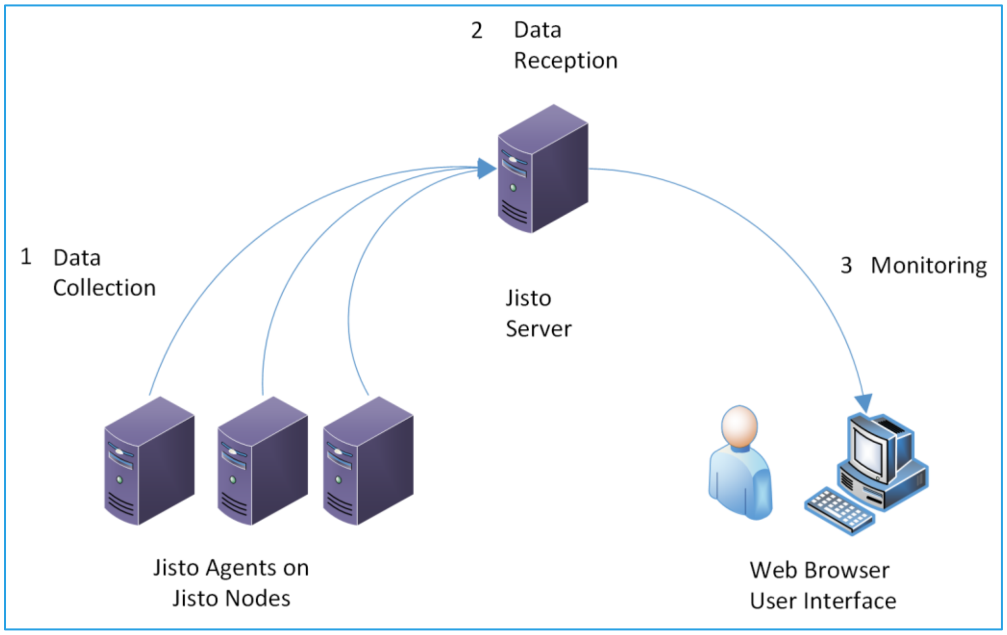


Figure 3 – Data Flow in Usage Monitoring

The data flow illustrated above is composed of the following stages:

1. The Jisto agents that are installed on the Jisto nodes read data on CPU and memory usage by incumbent applications and by the system as a whole.
2. The data that was collected by the different Jisto agents is sent to the Jisto server.
3. The user monitors activity on the Jisto nodes via the Web Browser UI.

3 Using the Elastic Workload Manager

The Elastic Workload Manager is accessed via a web browser user interface. From this UI, you can perform certain functions. Most importantly, you can define jobs that result in the deployment of applications to Jisto nodes for processing. You can subsequently monitor both the resources that are used by these secondary applications as well as those used by incumbent applications that are running on the same Jisto nodes. The Elastic Workload Manager lets you view this resource utilization in terms of criteria such as CPU, memory, network and disk usage.

The following sections describe how to perform these different tasks.

3.1 Loading Image Templates

3.1.1 Preparing the Image Template

Before you can upload an image template, you need to have a tar.gz file that contains the files that are required to build the image. The tar.gz file must contain 3 other files that the Elastic Workload Manager requires in order to run the application in a container. These files are described in the following table.

Table 2 – Contents of the Image Template

File	Purpose
`config.json`	Defines the build file, the run file, the directory in the tar where the application is installed.
`build.sh`	Defines the bash commands that install the application.
`run.sh`	Defines how the application will be run. Typically, this file will specify under what conditions the application will be invoked.

After these files have been defined and tested on a Linux system, create a tar.gz file that contains them. You might want to unzip the tar.gz and invoke the **run.sh** command in order to verify that the exported files work as expected. When you are satisfied that the tar has been created properly, you are ready to upload the image with the Elastic Workload Manager UI.

NOTE: Before creating an image template, please contact Jisto at support@jisto.com for more information.

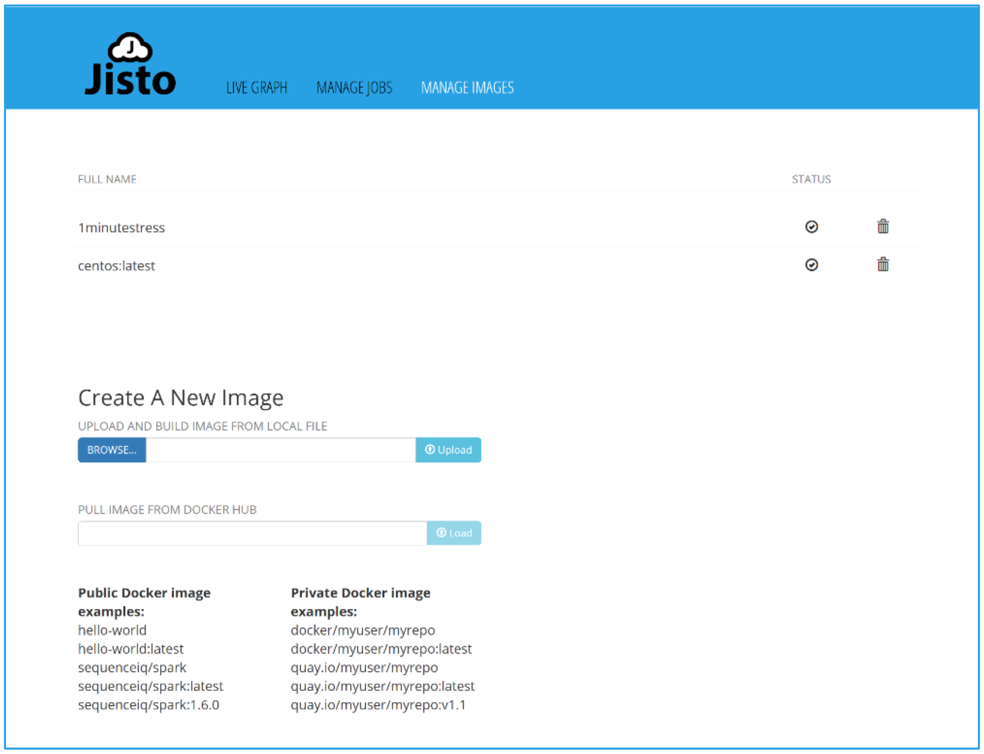
3.1.2 Using the Web Browser User Interface to Upload an Image

After you have a tar.gz containing the image, you need to upload it. Use the following procedure to do this:

1. Open the web browser and navigate to the Elastic Workload Manager UI. The address should be of the following format, where *XXX.XXX.XXX.XXX* is the IP address and where, if the default HTTP port 80 is not used, *port* is the port of the Jisto Server:

   ```
   http://XXX.XXX.XXX.XXX:port
   ```

2. The first screen that appears will be **Live Monitoring**. Click **Manage Images** to display the tab where you can upload an application image. The tab will be highlighted and displayed, as shown below:



FULL NAME	STATUS
1minutestress	✓
centos:latest	✓

Create A New Image

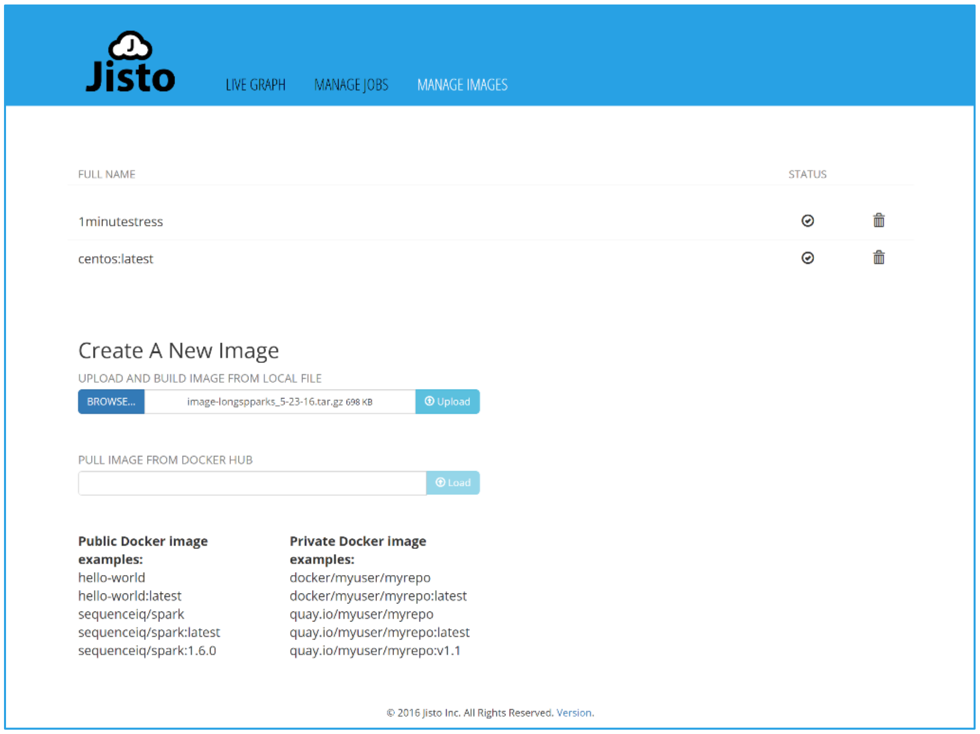
UPLOAD AND BUILD IMAGE FROM LOCAL FILE

PULL IMAGE FROM DOCKER HUB

Public Docker image examples:	Private Docker image examples:
hello-world	docker/myuser/myrepo
hello-world:latest	docker/myuser/myrepo:latest
sequenceiq/spark	quay.io/myuser/myrepo
sequenceiq/spark:latest	quay.io/myuser/myrepo:latest
sequenceiq/spark:1.6.0	quay.io/myuser/myrepo:v1.1

Figure 4 – Images Tab of Elastic Workload Manager

3. Click the **Browse** button. The system opens a dialog box that lets you search for and select the tar.gz file that contains the application image that you want to upload.
4. After selecting the tar.gz file, it will be listed in the **Create A New Image** field as shown below:


Jisto
LIVE GRAPH
MANAGE JOBS
MANAGE IMAGES

FULL NAME	STATUS
1minutestress	✓
centos:latest	✓

Create A New Image

UPLOAD AND BUILD IMAGE FROM LOCAL FILE

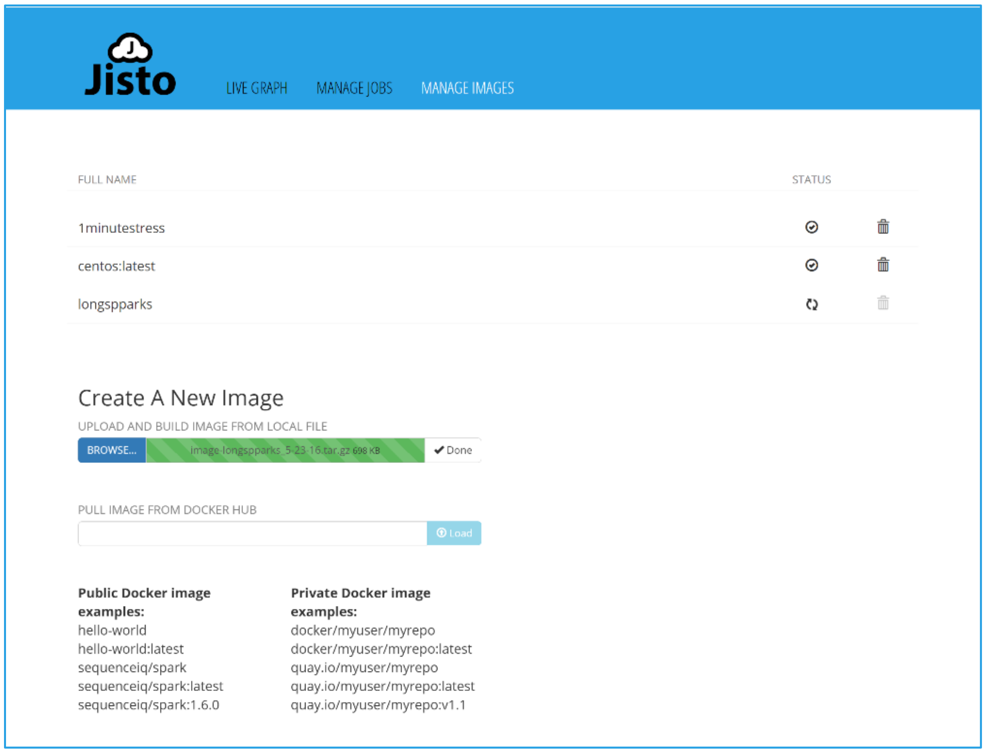
BROWSE...
image-longspparks_5-23-16.tar.gz 698 KB
Upload
PULL IMAGE FROM DOCKER HUB
Load

Public Docker image examples:	Private Docker image examples:
hello-world	docker/myuser/myrepo
hello-world:latest	docker/myuser/myrepo:latest
sequenceiq/spark	quay.io/myuser/myrepo
sequenceiq/spark:latest	quay.io/myuser/myrepo:latest
sequenceiq/spark:1.6.0	quay.io/myuser/myrepo:v1.1

© 2016 Jisto Inc. All Rights Reserved. Version.

Figure 5 – Images Tab with Image Identified

5. Click the **Upload** button. The Elastic Workload Manager begins to upload the image that you have specified. Under the string **FULL NAME**, the name of the image appears. During the upload, an icon made of 2 spinning circular arrows will appear to the right of the image name, as shown below:



FULL NAME	STATUS	
1minutestress	✓	🗑
centos:latest	✓	🗑
longspparks	⟳	🗑

Create A New Image

UPLOAD AND BUILD IMAGE FROM LOCAL FILE

PULL IMAGE FROM DOCKER HUB

Public Docker image examples:	Private Docker image examples:
hello-world	docker/myuser/myrepo
hello-world:latest	docker/myuser/myrepo:latest
sequenceiq/spark	quay.io/myuser/myrepo
sequenceiq/spark:latest	quay.io/myuser/myrepo:latest
sequenceiq/spark:1.6.0	quay.io/myuser/myrepo:v1.1

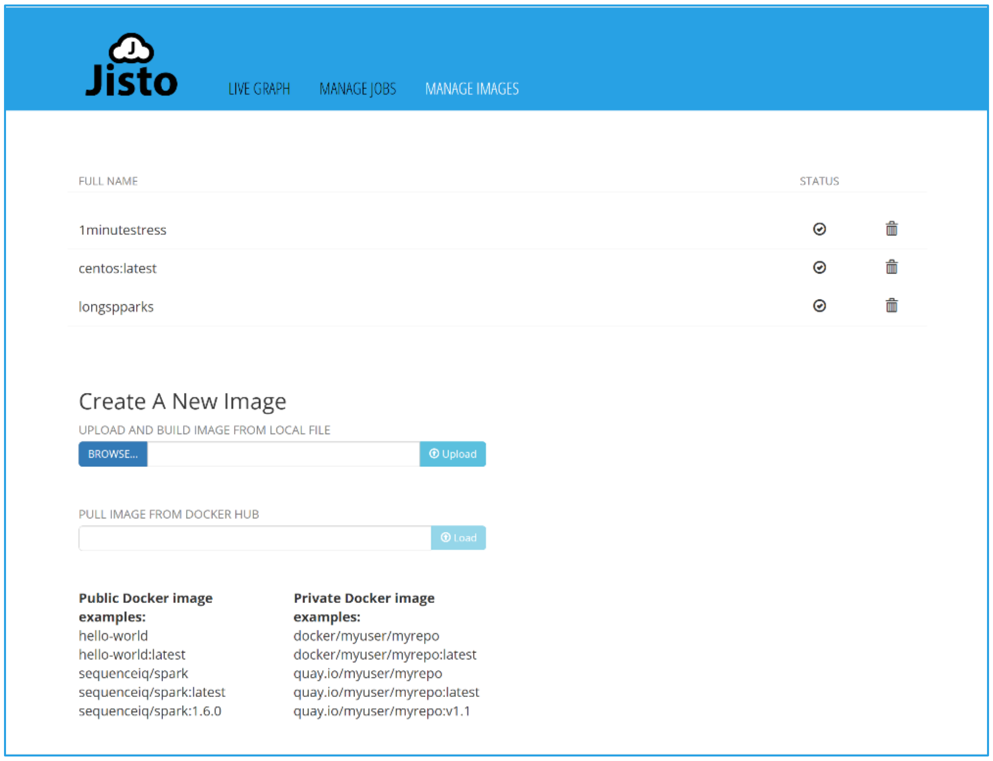
Figure 6 – Image Uploading

6. When the image has been successfully uploaded, the circular arrow icon will disappear and be replaced with the ✓ icon. Click the **Done** button to clear the **Create A New Image** field. You are then free to upload another image.

3.1.3 Deleting an Image

You may need to delete an image that has been uploaded. To delete an image, use the following procedure:

1. Open the web browser and navigate to the Elastic Workload Manager UI.
2. Click **Manage Images** to display the tab where you can upload an application image. The tab will be highlighted and display the images that had been earlier uploaded, as shown below:



FULL NAME	STATUS
1minutestress	✓
centos:latest	✓
longspparks	✓

Create A New Image

UPLOAD AND BUILD IMAGE FROM LOCAL FILE

PULL IMAGE FROM DOCKER HUB

Public Docker image examples:	Private Docker image examples:
hello-world	docker/myuser/myrepo
hello-world:latest	docker/myuser/myrepo:latest
sequenceiq/spark	quay.io/myuser/myrepo
sequenceiq/spark:latest	quay.io/myuser/myrepo:latest
sequenceiq/spark:1.6.0	quay.io/myuser/myrepo:v1.1

Figure 7 – Images Tab with Image To Be Deleted

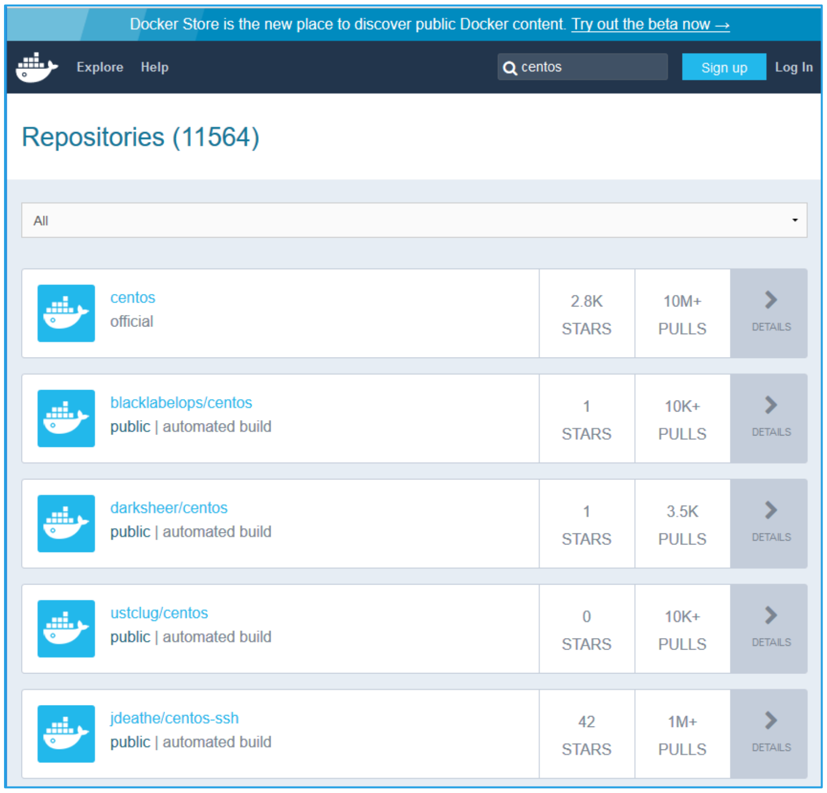
3. Click the 🗑 delete icon to the right of the image name. The system will remove the image from the list on the tab.

3.1.4 Loading Docker Images

The Elastic Workload Manager lets you upload and deploy Docker Images from Docker Hub, a popular Docker image library where thousands of containerized applications are stored. By using the **Select Docker image name** field, you can specify a Docker Hub image and load it into the Container Registry.

To load a Docker image, use the following procedure:

1. Open the web browser and navigate to the Docker Hub web site https://hub.docker.com.
2. Search for the image that you want to deploy for processing. If you are searching for a CentOS image, for example, you can enter the string `centos` in the search box and press Return. The Docker Hub web site would then look as shown below:

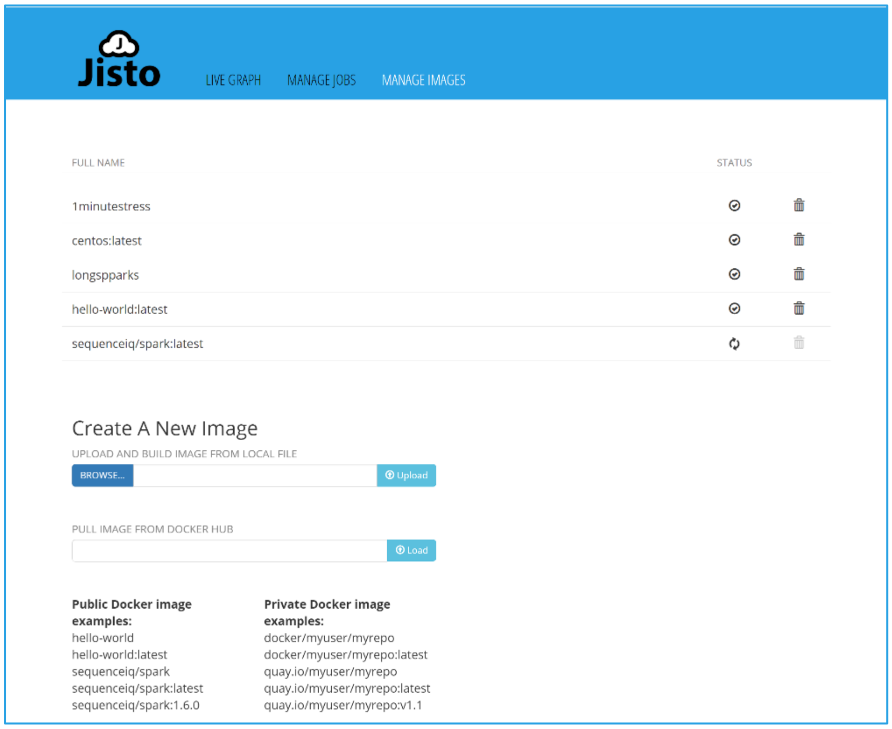

Docker Store is the new place to discover public Docker content. Try out the beta now →

Repositories (11564)

centos official	2.8K STARS	10M+ PULLS	DETAILS
blacklabelops/centos public | automated build	1 STARS	10K+ PULLS	DETAILS
darksheer/centos public | automated build	1 STARS	3.5K PULLS	DETAILS
ustclug/centos public | automated build	0 STARS	10K+ PULLS	DETAILS
jdeathe/centos-ssh public | automated build	42 STARS	1M+ PULLS	DETAILS

Figure 8 – Docker Hub Web Site After Search

3. Find the application that you want to load and deploy and note its name. **You should know that only official Docker-approved applications can be loaded into the Elastic Workload Manager**. The `centos` image shown in Figure 8, for example, is an official image. As such it is identified by its name alone and by the string `official` that appears beneath it.
4. Enter the name of the application image in the **Name** field.
5. Click the **Load** button under the **NAME** field. The Elastic Workload Manager begins to upload the Docker image to the Container Registry. Under the string **FULL NAME**, the name of the Docker image appears. During the upload, an icon made of 2 spinning circular arrows will appear to the right of the Docker image name, as shown below:



FULL NAME	STATUS
1minutestress	✓
centos:latest	✓
longspparks	✓
hello-world:latest	✓
sequenceiq/spark:latest	⟳

Create A New Image

UPLOAD AND BUILD IMAGE FROM LOCAL FILE

PULL IMAGE FROM DOCKER HUB

Public Docker image examples:	Private Docker image examples:
hello-world	docker/myuser/myrepo
hello-world:latest	docker/myuser/myrepo:latest
sequenceiq/spark	quay.io/myuser/myrepo
sequenceiq/spark:latest	quay.io/myuser/myrepo:latest
sequenceiq/spark:1.6.0	quay.io/myuser/myrepo:v1.1

Figure 9 – Docker Image During Load

6. After the Docker image has been successfully loaded, the circular arrow icon will disappear and the name remains. You are then free to load another Docker image or deploy an image as explained in section 3.2 Creating a Job.

3.2 Creating a Job

3.2.1 Preparing the Job Source File

Before you can create a job, you need to have a tar.gz file that contains the job source file. The job source file defines how many times the containerized application should be run. It is a short JSON file that must be named `config.json` and must at a minimum contain the `iterations` parameter.[1] It may contain other, optional parameters as well.

3.2.1.1 Required Parameter

The syntax for the `iterations` parameter is shown below:

```
{
        "iterations": "x"
}
```

where the *x* value following "`iterations`" defines the number of times that the containerized

[1] The `config.json` file that is part of the image template is a separate file from the `config.json` file that is the job source file. These are 2 separate files that have different purposes.

application will be run, as in the following example:

```
        {
                "iterations": "20"
        }
```

3.2.1.2 Optional jobName Parameter

The syntax for the `jobName` parameter is shown below:

```
        {
                "jobName": "x"
        }
```

where the *`x`* value following "`jobName`" specifies the name of the job, as in the following example:

```
        {
                "jobName": "Hello World!"
        }
```

As its name implies, the `jobName` parameter specifies the name of the job. This job name appears on the **Manage Jobs** page under the **Name** column. If a job name is not specified with this parameter, Jisto assigns the name of the associated image as the job name.

3.2.1.3 Options Parameter

There are additional parameters that can be specified in a job source file. These parameters are embedded in the `options` parameter. Currently there are five optional parameters, shown below:

- `net`
- `privileged`
- `volumes`
- `tcpports`
- `udpports`
- `env`

The `net` parameter defines the network used by the containerized application. It has the following syntax:

```
        "net": "network"
```

where `network` specifies the network stack and interfaces that the container will use. At present, the only supported value is `host` which specifies that the containerized application will share the Jisto node's network. Any other value will configure the network to be bridged, meaning that it will have its own network internal to the Jisto node. In this case, it is possible to expose the container ports on the Jisto node by using the `tcpports` and `udpports` parameters described below.

CAUTION: Do not use the `tcpports` and `udpports` parameters when `net` is set to `host`.

The `privileged` parameter specifies if the container will have capabilities on the Jisto node that go beyond what would normally be available to it. These capabilities include kernel features and device access. The parameter has the following syntax:

```
"privileged": "boolean"
```

where *`boolean`* is either `true` or `false`. A setting of `true` specifies that the container has extra capabilities. It should be used only if the containerized application needs these enhanced system resources.

The `volumes` parameter lets you specify volumes on the host Jisto node that the container can access. This parameter has the following syntax:

```
"volumes":{

       "host_volume ": "container_volume"

         }
```

The *`host_volume`* is the path of the volume on the Jisto node to be shared with the container. The *`container_volume`* is the name of the volume used within the container's file system that will make available the contents of *`host_volume`*. Both these values are strings that must be enclosed in double quotes.

The `tcpports` parameter lets you forward TCP ports from the container to the Jisto node to the container. It has the following syntax:

```
"tcpports":{

       "host_TCP_port": "container_TCP_port"

         }
```

where *`host_TCP_port`* is the port number on the Jisto node that will expose the container port and *`container_TCP_port`* is the container port number to be exposed. Both these values are integers that must be enclosed in double quotes. This parameter is incompatible with the `“net”:”host”` setting described above.

In a similar way, the `udpports` parameter lets you forward UDP ports from the Jisto node to the container. It has the following syntax:

```
"udpports":{

       "host_UDP_port": "container_UDP_port"

         }
```

where `host_UDP_port` is the UDP port number on the Jisto node that will expose the container port and `container_UDP_port` is the container UDP port number to be exposed. Both these values are integers that must be enclosed in double quotes. This parameter is incompatible with the `"net": "host"` setting described above.

The `env` parameter lets you define a setting for an environment variable to be used in the container. It has the following syntax:

```
"env": "env_variable=value"
```

where *`env_variable`* and is the environment variables and *`value`* is its setting. The `env` parameter can be used to define more than one environment variable. When used in this way, each variable value pair must be delimited by a semi colon, as shown in the following example:

```
"env": "SLEEP_TIME=300;baz=quux"
```

As noted earlier, the optional parameters must be embedded in the `options` parameter. This means, in essence, that optional parameters are inserted within `"options":{… }` in the job source file. The following code sample shows how these optional parameters would look in the JSON file, although as noted earlier the `"net": "host"` setting is incompatible with the `udpports` and `tcpports` parameters:

```
{
    "iterations" : "1",
    "options":{
        "net": "host",
        "privileged": "true",
        "volumes":{
          "/tmp":"/hosttmp",
          "/var":"/hostvar"
        },
        "tcpports":{
          "8000":"8000"
        },
        "udpports":{
          "8000":"8000"
        }
        "env": "SLEEP_TIME=300;baz=quux"
    }
}
```

You can create the job source file with a text editor such as Notepad. When you have a job source file, load it into a tar.gz file.

NOTE: For help creating a job source file, please contact Jisto at support@jisto.com for more information.

3.2.2 Using the UI to Create a Job

When you have a tar.gz file that contains the job template and after you have uploaded one or more image templates, you can create a job. To do this, use the following procedure:

1. Click **Manage Jobs** at the top of the screen. The **Manage Jobs** tab is displayed as shown below:

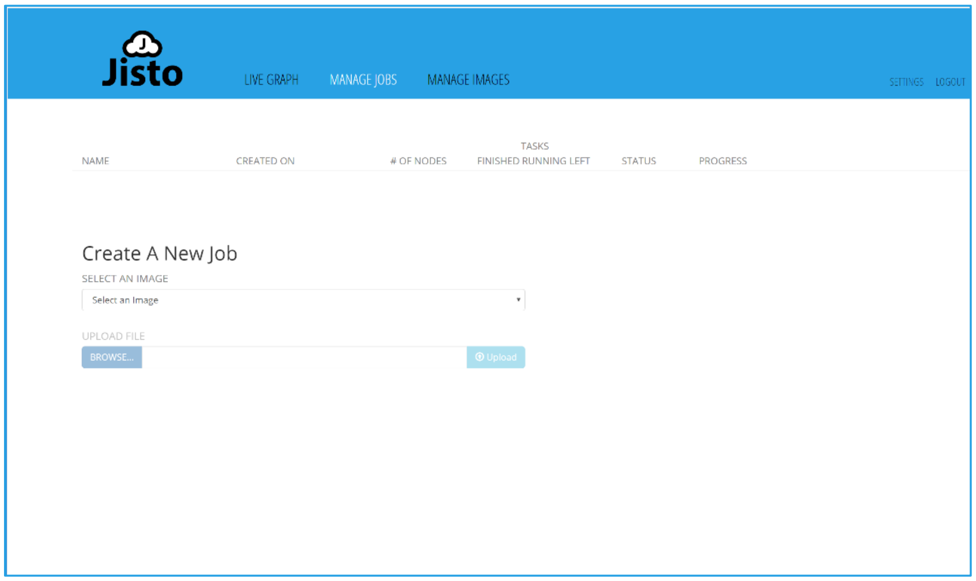
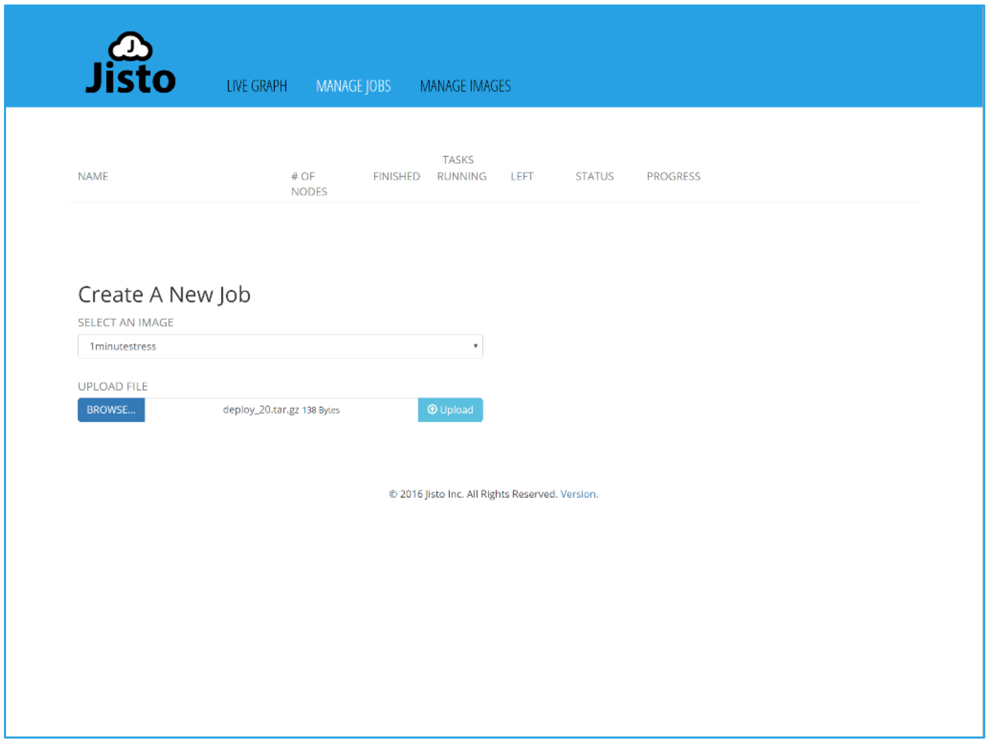


Figure 10 – Jobs Tab

2. In the **SELECT AN IMAGE** field, use the drop-down list to select the image.
3. In the **UPLOAD FILE** field, click the **BROWSE** button. The system opens a dialog box that lets you search for and select the job template. After you select the job template, its name will appear on the **Jobs** tab as shown below:



NAME	# OF NODES	FINISHED	TASKS RUNNING	LEFT	STATUS	PROGRESS

Create A New Job

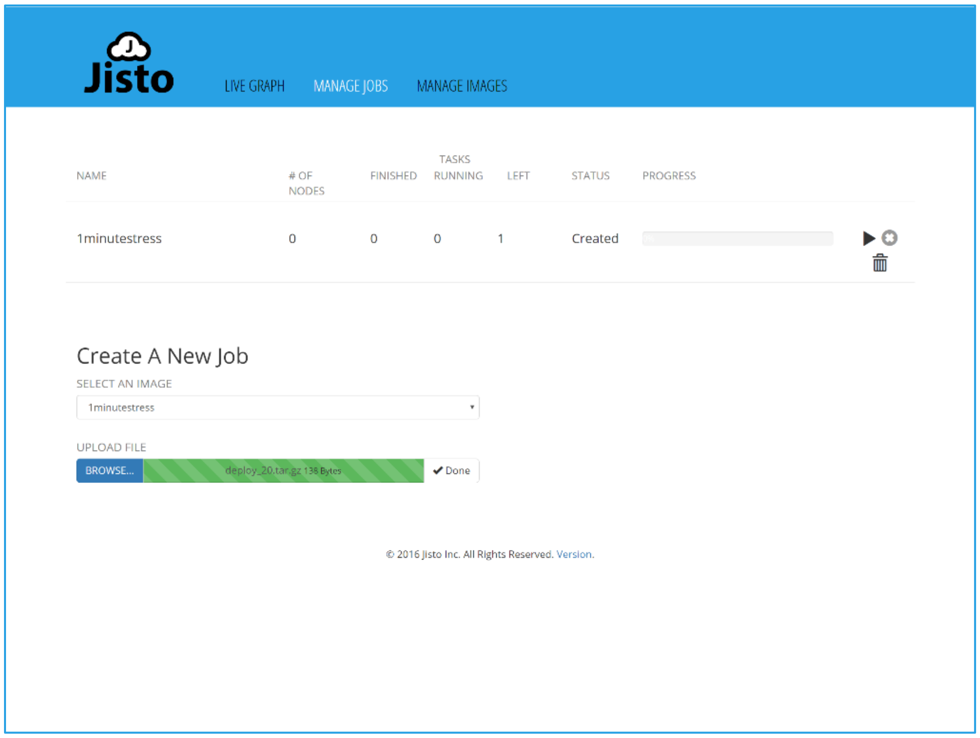
SELECT AN IMAGE: 1minutestress

UPLOAD FILE: deploy_20.tar.gz 138 Bytes

© 2016 Jisto Inc. All Rights Reserved. Version.

Figure 11 – Jobs Tab

4. Click the **Upload** button. The new job will be listed on the **Jobs** tab, as shown below:



NAME	# OF NODES	FINISHED	TASKS RUNNING	LEFT	STATUS	PROGRESS
1minutestress	0	0	0	1	Created	

Create A New Job

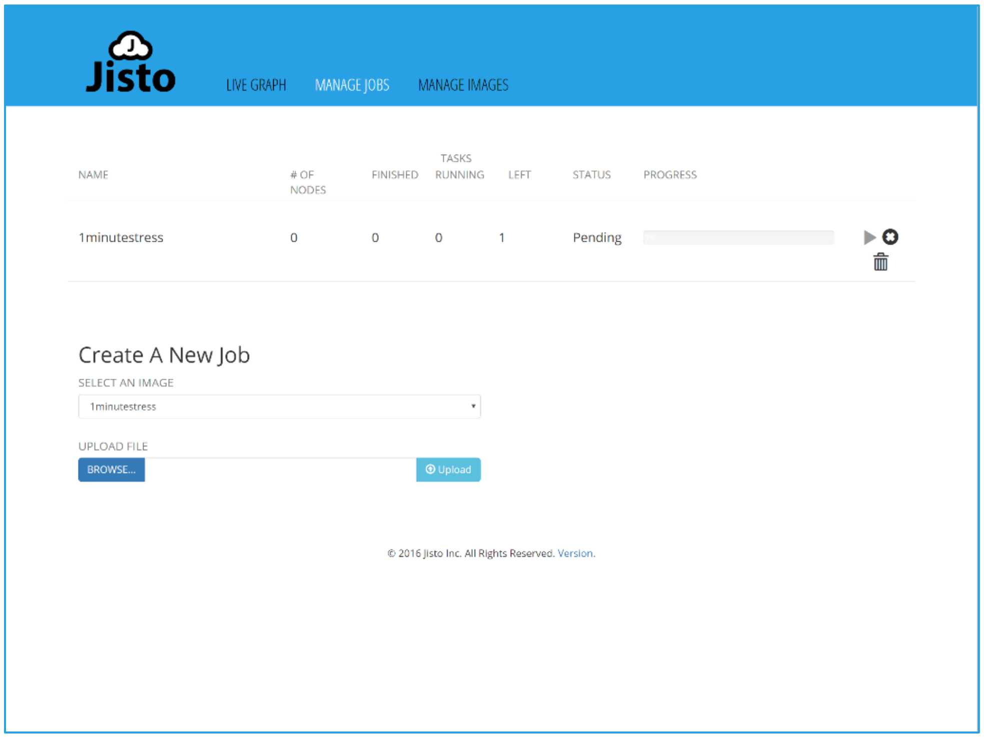
SELECT AN IMAGE: 1minutestress

UPLOAD FILE: BROWSE... deploy_20.tar.gz 138 Bytes ✔ Done

© 2016 Jisto Inc. All Rights Reserved. Version.

Figure 12 – Newly-Created Job

5. Click the **Done** button to clear the **Select job source file** field. You are then free to upload another job template.
6. Click the ▶ play button to start deployment of the job to Jisto nodes. The progress status of the job will change to Pending, as shown below:



NAME	# OF NODES	FINISHED	TASKS RUNNING	LEFT	STATUS	PROGRESS
1minutestress	0	0	0	1	Pending	

Create A New Job

SELECT AN IMAGE

1minutestress

UPLOAD FILE

© 2016 Jisto Inc. All Rights Reserved. Version.

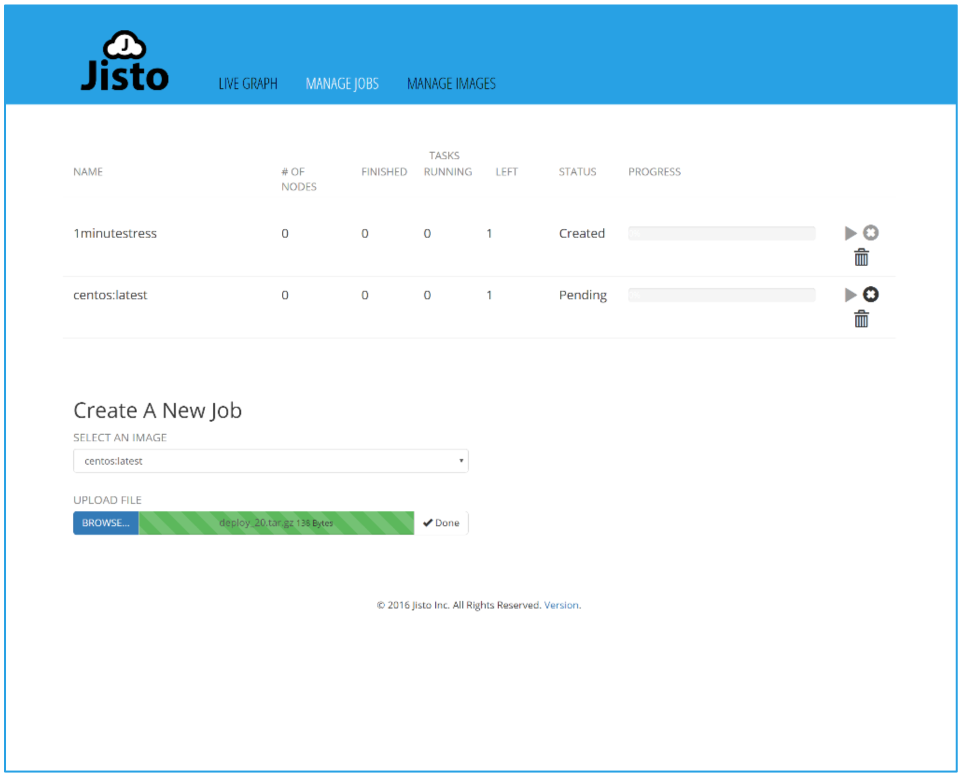
Figure 13 – Job Pending

After the job has concluded processing, the progress status will change to Finished.

3.2.3 Deleting a Job

From time to time you may need to delete a job that has been uploaded. This could occur, for example, after you have run the application a sufficient number of times. To delete a job, use the following procedure:

1. Open the web browser and navigate to the Elastic Workload Manager UI.
2. Click **Jobs** to display the tab where you can upload a job source file. The tab will be highlighted and display the jobs that had been earlier uploaded, as shown below:



NAME	# OF NODES	FINISHED	TASKS RUNNING	LEFT	STATUS	PROGRESS
1minutestress	0	0	0	1	Created	
centos:latest	0	0	0	1	Pending	

Create A New Job

SELECT AN IMAGE: centos:latest

UPLOAD FILE: deploy_20.tar.gz 138 Bytes

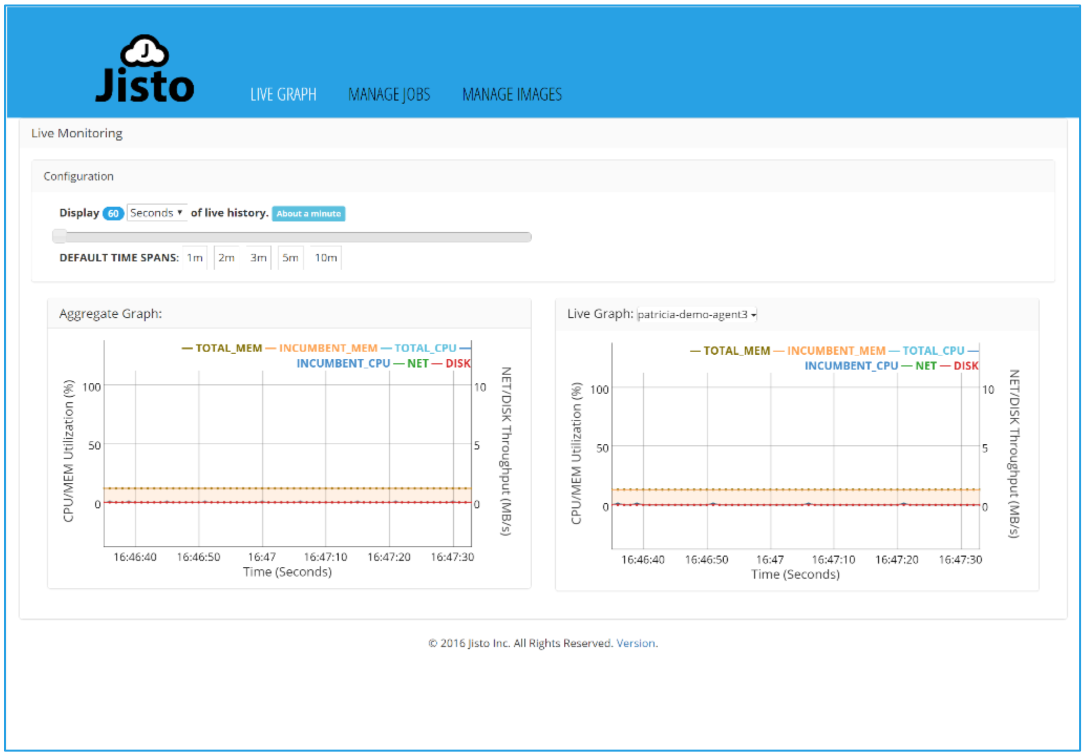
© 2016 Jisto Inc. All Rights Reserved. Version.

Figure 14 – Jobs Tab with Job to Be Deleted

3. Click the 🗑 delete button to the right of the job name. The system will remove the job from the list on the tab.

3.3 Monitoring Jobs on Jisto Nodes

After the jobs have been deployed, you can use the **Live Graph** tab to view the effects that they have on the Jisto nodes. To do this, click **Live Graph**. Your screen may appear as shown below:


Jisto
LIVE GRAPH
MANAGE JOBS
MANAGE IMAGES
Live Monitoring
Configuration
DEFAULT TIME SPANS: 1m 2m 3m 5m 10m
Aggregate Graph:
Live Graph:
— TOTAL_MEM — INCUMBENT_MEM — TOTAL_CPU — INCUMBENT_CPU — NET — DISK
CPU/MEM Utilization (%)
NET/DISK Throughput (MB/s)
Time (Seconds)
© 2016 Jisto Inc. All Rights Reserved. Version.

Display 60 Seconds of live history. About a minute

Live Graph: patricia-demo-agent3

Figure 15 – Live Graph Tab

This tab lets you view the utilization of each Jisto node. On the right side of the tab, you can use the drop down list in the **Live** field to select a node to view. In Figure 15, for example, the node kevin-demo-agent3 is currently selected. Underneath this field, the percentage of memory and CPU that are used by incumbent applications are plotted against total CPU and memory. The total amount of network traffic and disk usage on the node are also presented. On the left side of the window, you can view the average CPU and memory utilization of all the nodes, plotted over time. You can also view the average network and disk usage of all the nodes.

3.3.1 Color Coding Schema

On the **Live Monitoring** tab, data is color-coded whether it appears in graphs or in the additional information window. The following table explains the color coding schema for data:

Table 3 – Color Codes for Live Monitoring

INCUMBENT_CPU	Dark blue identifies resources used by the incumbent applications on the Jisto nodes.
TOTAL_CPU	Light blue identifies resources used by all the applications on the Jisto nodes.
MEM	On the graph showing the average usage across the Jisto nodes, light brown is used to show memory usage by incumbent applications.

MEM	Dark brown is used to indicate total memory usage on the Jisto nodes.
NET	Green is used to indicate network usage on the Jisto nodes.
DISK	Red identifies disk usage on the Jisto nodes.

3.3.2 Display Options

If you want to change the time duration for the usage data that you are observing, you can use the **Display** option. This option lets you specify the observation period by means of a drop down list and a horizontal scroll bar. The drop down list lets you specify the units of time – **Seconds**, **Minutes** or **Hours**. The horizontal scroll bar lets you specify the number of these units. Both are shown below:

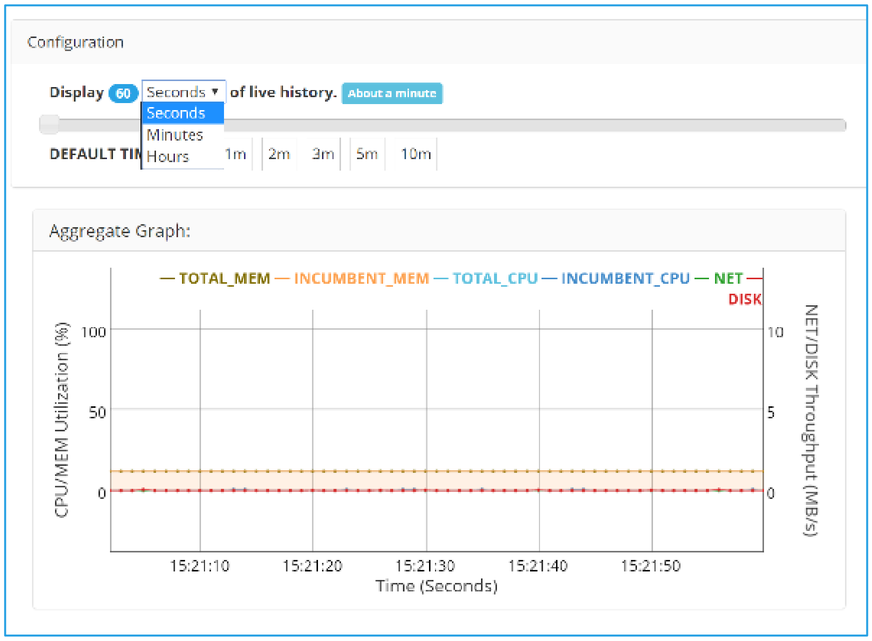


Figure 16 – Display Option Menu

If you want to change the period of time under review, use the drop down list to select the unit of time. Then move the scroll bar to select the number of unit. To display utilization over approximately the last 60 minutes, for example, you would select Minutes from the drop down list and then move the scroll bar to select 60, as shown below:

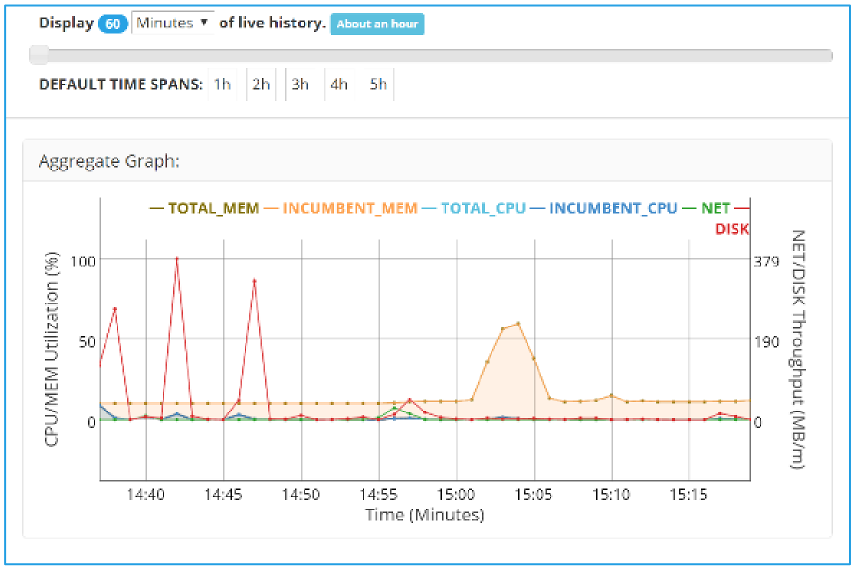


Figure 17 – Display Option Selected

4 Using the Command Line Interface

Depending on the needs of your organization, you may want to use the Command Line Interface (CLI) of the Elastic Workload Manager. The CLI commands let you perform most of the operations that the UI supports. You can, for example, upload application images and create jobs.

Using the CLI, you can also create tasks and associate them with images and jobs explicitly. A task can be thought of as a command that associates an application image with a job. This means that with the CLI, you can define a job that deploys many different application images and not just one single image. This provides greater flexibility and automation in job definition than is currently available in the Web UI.

While tasks give you greater freedom when creating a job, this feature requires that you create a task for every job you create, even a job that will run just one image. This means, unfortunately, that more steps are required to create a job with only one image when using the CLI than when using the Web UI.

Because of these different features, there is a slightly different workflow in the CLI than the Web UI. Before using the CLI commands, you should familiarize yourself with CLI workflow, shown below.

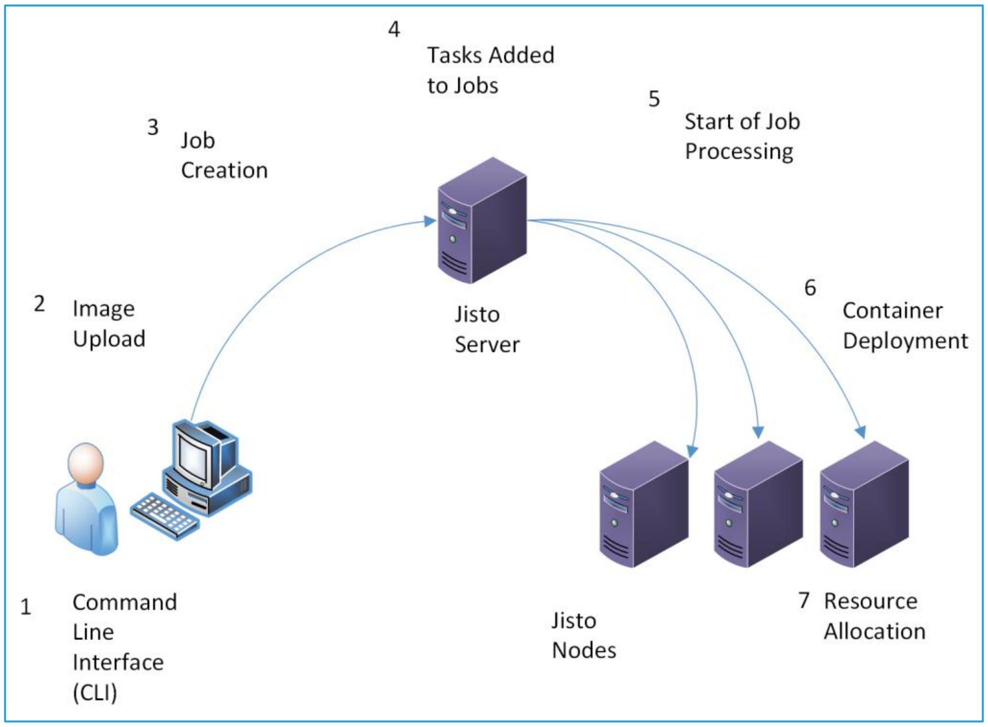


Figure 18 – Workflow Using the CLI

Note that you can also create scripts that incorporate CLI commands and subsequently run them either manually or automatically. Creating and running scripts can save time when you need to upload large numbers of images or create multiple tasks and jobs.

The following sections introduce you to the CLI command syntax and describe how to perform the essential procedures of uploading an image, creating a job and adding tasks that associate images to jobs with the CLI.

4.1 Before Using the Command Line Interface

Before using `jistocli`, you must first open the gRPC port 16384 on the Jisto Server. To do this, login with an account that has root privileges on the Jisto Server and type the following command or the equivalent:

```
iptables -I INPUT -p tcp --dport 16384 -j ACCEPT
```

Then set the path by typing the following command:

```
export PATH=$PATH:/opt/jisto
```

4.2 CLI Commands

The CLI consists of one command with the following syntax, where *`XXX.XXX.XXX.XXX`* is an IP address:

```
jistocli [--version]
         [--help]
```

```
[--web XXX.XXX.XXX.XXX -u user_id image upload image_name]
[--web XXX.XXX.XXX.XXX -u user_id image delete image_name]
[--scheduler XXX.XXX.XXX.XXX:16384 job create job_name]
[--scheduler XXX.XXX.XXX.XXX:16384 job run JobUUID]
[--scheduler XXX.XXX.XXX.XXX:16384 job delete JobUUID]
[--scheduler XXX.XXX.XXX.XXX:16384 task create JobUUID
image_name task_name env 'variable=value']
```

The `jistocli` command can be used in the 8 different ways enclosed in brackets above.

CAUTION: The brackets that enclose each way that you can use `jistocli` are not part of the command. **Do not enter the brackets on the command line**.

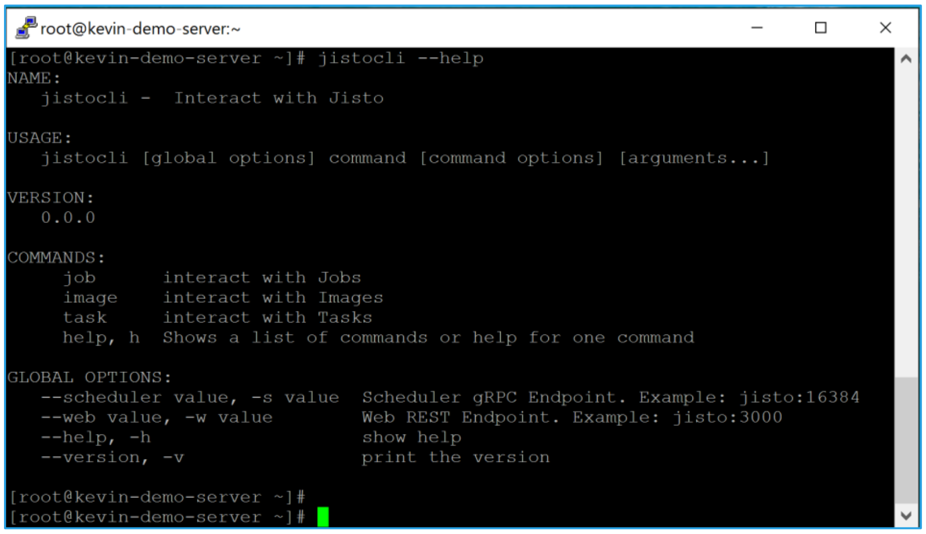
The following sections present the `--help` and `--version` global options before describing the `image`, `job`, and `task` commands and the `upload`, `delete`, `create`, `run` and `env` options that you can use with the `jistocli` command.

4.3 Displaying Version Number and Help

The two options `--version` and `--help` display information that can be of interest to you. When you type `jistocli --version`, for example, the installer displays the version number of the Jisto CLI, as shown below:

```
jistocli version 0.0.0
```

The `--help` option displays the built-in documentation associated with the installer. Information on all the `jistocli` options is displayed, as shown in the screen shot below.



```
[root@kevin-demo-server ~]# jistocli --help
NAME:
   jistocli -  Interact with Jisto

USAGE:
   jistocli [global options] command [command options] [arguments...]

VERSION:
   0.0.0

COMMANDS:
     job      interact with Jobs
     image    interact with Images
     task     interact with Tasks
     help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --scheduler value, -s value  Scheduler gRPC Endpoint. Example: jisto:16384
   --web value, -w value        Web REST Endpoint. Example: jisto:3000
   --help, -h                   show help
   --version, -v                print the version

[root@kevin-demo-server ~]#
[root@kevin-demo-server ~]#
```

Figure 19 – Help on the `jistocli` Command

4.4 Uploading an Image

The `image` command lets you upload an image into the Elastic Workload Manager. If you are uploading or deleting an image using the `jistocli` on a system other than the Jisto Server, you must use the `--web` *`XXX.XXX.XXX.XXX`* option to specify the Web UI port, where *`XXX.XXX.XXX.XXX`* is the IP address or hostname of the Jisto Server.

In addition, you must use the `-u` global option to specify a user ID with sufficient privileges to upload an image. The system will prompt you for the password after launching the command.

To upload an image, for example, enter a command that uses the `--web` and `-u` options as shown below:

```
jistocli --web XXX.XXX.XXX.XXX -u admin image upload image.tar.gz
```

The CLI prompts for the password of the user *`admin`*:

```
2016/10/24 14:23:21 Enter password for: admin
```

After you enter a password, the CLI then displays an informational message, as shown below:

```
Image {image} uploaded. Note that it may not be available for
immediate use.
```

Note of the name of the image. You will need the name of the image to delete it or to associate it to a task.

4.5 Creating a Job

The `job` command lets you create, run and delete a job from the command line. When you create a job, you must specify the gRPC port of the Jisto Server Elastic Scheduler for it, as shown in the following example:

```
jistocli --scheduler XXX.XXX.XXX.XXX:16384 job create MyJob
```

The CLI then displays the UUID of the job, as shown in the example below:

```
2016/08/04 12:18:32 Created Job. Name: { MyJob } UUID: { 1685d125-
5a5f-11e6-92b6-005056061e5b }
```

In this example, the UUID of the job that was created is `1685d125-5a5f-11e6-92b6-005056061e5b`. **Each time that you create a job, take note of the job's UUID.** You will need the UUID to run or delete the job, and to create tasks associated with it.

CAUTION: Please note that after creating a job, you will not be able to run it until you have defined at least one task for it.

4.6 Creating a Task for a Job

The `task` command lets you associate an image with a job. It has the following syntax:

```
jistocli -scheduler XXX.XXX.XXX.XXX:16384 task create JobUUID
ImageName env 'variable=value'
```

where `XXX.XXX.XXX.XXX` is the IP address or hostname of the server and `JobUUID` is the UUID is the job's UUID and `ImageName` is the name of the image and `variable=value` is the environment variable value pair that the `env` option defines.

You can create a task by using these arguments, as in the following example:

```
jistocli --scheduler XXX.XXX.XXX.XXX:16384 task create 1685d125-
5a5f-11e6-92b6-005056061e5b stress env 'SLEEP_TIME=300'
```

To add several iterations of the same image to a job, make repeated invocations of the same command. To specify that an image be run multiple times, for example, invoke the above command as many times as required.

You can use the `env` option to define a value for an environment variable to be used by the task when processing. The `env` option can be used to define more than one environment variable. When used in this way, each variable value pair must be delimited by a semi colon, as shown in the following example:

```
'SLEEP_TIME=300;baz=quux'
```

You can also use the `task` command to associate several different images with just one job. To do this, specify the same `JobUUID` and different `ImageName` values when invoking the task command with the create option. You can add as many tasks to a job as you want until the job is completely defined.

Note that you can associate an image with a task after it has been uploaded BUT before it has been built. Recall from the section Uploading an Image that an image is NOT properly built for Jisto immediately after upload.

4.7 Running a Job

After creating a job and an image, and then creating a task to associate them together, you can use the `job` command with the `run` option to start the job from the command line. Use this command only after you have verified that images that are associated with the job have been built properly. (There is currently no way to do this using `jistocli`, but you can verify this through the web interface.)

When you run a job, you must specify the gRPC port of the Jisto Server Elastic Scheduler for it, as shown in the following example:

```
jistocli --scheduler XXX.XXX.XXX.XXX:16384 job run 1685d125-5a5f-
11e6-92b6-005056061e5b
```

The CLI then displays a message, as shown below:

```
2016/08/04 17:27:12 Job Run operation success. UUID: { 1685d125-
5a5f-11e6-92b6-005056061e5b }
```

4.8 Deleting Images and Jobs

To delete a job or an image, you need to use the delete command option with the job or image command. To delete a job, you would use a command like the following:

```
jistocli --scheduler XXX.XXX.XXX.XXX:16384 job delete 1685d125-5a5f-
11e6-92b6-005056061e5b
```

The CLI then displays a message, as shown below:

```
2016/08/04 17:36:54 Deleted Job. UUID: { 1685d125-5a5f-11e6-92b6-
005056061e5b }
```

To delete an image, you would use a command like the following:

```
jistocli --web XXX.XXX.XXX.XXX -u admin image delete stress
```

The CLI prompts for the password of the user `admin`:

```
2016/10/24 14:23:21 Enter password for: admin
```

After you enter a password, CLI then displays a message, as shown below:

```
Image {stress} Deleted.
```

NOTE: At present you cannot delete tasks from within a job. If you make an error in adding a task to a job, you must build a new job with the correct tasks.

5 Using the Jisto RESTful API

Jisto provides a RESTful API that you can use to create applications that perform all the operations that are available in the CLI. This API lets you

- Upload an image
- Load an image from the Docker registry
- Create a job
- Create a task and associate it with an image and a job
- Run jobs
- Delete jobs
- Delete images

The following sections introduce you to the Jisto RESTful API and describe how services that let perform the essential procedures of uploading an image, creating a job and adding tasks that associate images to jobs.

5.1 Uploading an Image

The service that you can use to upload an image has the attributes described below.

Summary	upload uploads an image.
URL	Upload
Method	POST
URL Params	NONE
Form Data	`contentType` – Must be set to `images`. `files[]` – Must specify the image file. Does **NOT** specify an array. At present, the image specified by `files[]` must be a tar.gz file that contains all files and directories containing the image.
Multipart Form Data	`------WebKitFormBoundarywUiXDmsMlAbmucEy Content-` `Disposition: form-data; name="contentType" images ------ WebKitFormBoundarywUiXDmsMlAbmucEy Content-` `Disposition: form-data; name="files[]"; filename="1minuteofstress.tar.gz" Content-Type: application/gzip ------` `WebKitFormBoundarywUiXDmsMlAbmucEy--`
Success Response	200
Sample Call	`$ curl -X POST --header 'Content-Type: multipart/form-data' --header 'Accept: application/json' --header 'Authorization: Basic cm9vdDpqaXN0bw==' -F "contentType=images;files[]=@1minutestress.tar.gz" 'http://172.18.0.2/upload'`
Authorization Header Notes	An `upload` call requires user authentication and authorization. This means that a call must pass a valid user ID and password for it to be successful. This information is passed via an `Authorization` header, as shown in the Sample Call. The `Authorization` header takes a value of the form `Basic` *`xxx`* where *`xxx`* is a Base64 encoded user_id:password pair. You must encode a valid pair in Base64 and then specify it in an `Authorization` header. See the https://www.base64encode.org/ web site for information on encoding a user_id:password pair.

5.2 Pulling an Image from the Docker Hub Registry

The service that you can use to pull an image from Docker Hub has the attributes described below.

Summary	pullImage pulls an image from the Docker Hub Registry.
URL	/pullImage
Method	POST
URL Params	*`imageName`* – The name of the image stored in Docker Hub. Specifying a Docker Hub image requires the insertion of a colon in the

	`imageName` string. To insert a colon in a URL, use the characters `%3A`. A colon will be substituted for these characters in subsequent processing.
Body Params	NONE
Success Response	200
Sample Call	`$ curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Authorization: Basic cm9vdDpqaXN0bw==' 'http://172.18.0.2/pullImage/hello-world%3Alatest'`
Authorization Header Notes	A `pullImage` call requires user authentication and authorization. This means that a call must pass a valid user ID and password for it to be successful. This information is passed via an `Authorization` header, as shown in the Sample Call. The `Authorization` header takes a value of the form `Basic xxx` where `xxx` is a Base64 encoded user_id:password pair. You must encode a valid pair in Base64 and then specify it in an `Authorization` header. See the web site https://www.base64encode.org/ for information on encoding a user_id:password pair.

5.3 Deleting an Image

The service that you can use to delete an image has the attributes described below.

Summary	deleteImage deletes an image that have been previously uploaded to the Jisto Server or pulled from the Docker Hub Registry.
URL	/deleteImage
Method	POST
URL Params	`imageName` – The name of the image to be deleted. To insert a colon in a URL, use the characters `%3A`. A colon will be substituted for these characters in subsequent processing.
Body Params	NONE
Success Response	Code 200
Sample Call	`curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Authorization: Basic cm9vdDpqaXN0bw==' 'http://172.18.0.2/deleteImage/hello-world%3Alatest'`

Authorization Header Notes	A `deleteImage` call requires user authentication and authorization. This means that a call must pass a valid user ID and password for it to be successful. This information is passed via an `Authorization` header, as shown in the Sample Call. The `Authorization` header takes a value of the form `Basic` *`xxx`* where *`xxx`* is a Base64 encoded user_id:password pair. You must encode a valid pair in Base64 and then specify it in an `Authorization` header. See the web site https://www.base64encode.org/ for information on encoding a user_id:password pair.

5.4 Creating a Job

The service that can be invoked to create a job on the Jisto server has the attributes described below.

Summary	job returns a Job UUID, given a valid definition.
URL	Job
Method	POST
URL Params	NONE
Body Params	`"Name": "`*`JobName`*`"` where *`JobName`* is, of course, the name of the job.
Success Response	Code 200 UUID – Job UUID.
Sample Call	`$ curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{"Name": "MikeTestJob"}' 'http://172.18.0.2:9878/v1/job'`
Notes	

5.5 Creating a Task

The service that can be invoked to create a task on the Jisto server has the attributes described below.

Summary	task returns a Task UUID, given a valid definition.
URL	Task
Method	POST
URL Params	NONE
Body Params	`"Image" : "`*`ImageName`*`"` where *`ImageName`* is the name of the image. `"JobUUID" : "`*`JobUUID`*`"` where *`JobUUID`* is the identifier of the job.
Success	Code 200

Response	Task UUID
Sample Call	`$ curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' -d '{ "Image" : "1minutestress", "JobUUID": "2cf2b20a-90a8-11e6-bcf7-0242ac120002" }' 'http://172.18.0.2:9878/v1/task'`
Notes	

5.6 Running, Stopping, Cancelling or Deleting a Job

The service that can be invoked to run, stop, cancel or delete a job has the attributes described below.

Summary	UpdateJob is used to issue JobUpdate operations. It permits you to run, stop, cancel or delete a job.
URL	job/{JobUUID}/op/{JobOperation}
Method	POST

URL Params	`JobOperation` – Specified the operation to perform. Accepted and self-explanatory values are as follows:
	
	`RunJob`
	`StopJob`
	`CancelJob`
	`DeleteJob`
	
	`JobUUID` – Identifier of the job.

Body Params	NONE
Success Response	Code 200 UUID of updated job.
Sample Call	`curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' 'http://172.18.0.2:9878/v1/job/2cf2b20a-90a8-11e6-bcf7-0242ac120002/op/RunJob'`
Notes	

5.7 Retrieving a Job

The `job` service can be invoked to retrieve information on a job. It has the attributes described below.

Summary	GetJob is used to retrieve data on a job.
URL	job/{UUID}
Method	GET
URL Params	`UUID` – Identifies the job on which data is to be returned.
Body	NONE

Params	
Success Response	Code 200 UUID of job. Name of job. State of job.
Sample Call	`$ curl -X GET --header 'Accept: application/json' 'http://172.18.0.2:9878/v1/job/2cf2b20a-90a8-11e6-bcf7-0242ac120002'`
Notes	

6 Creating a Secondary Application With Chef

In addition to creating images through the Web UI, the CLI and the RESTful API, Jisto allows you to upload images that are created with the Chef 12 Development Kit. Jisto lets you subsequently run these Chef images as Jisto-managed secondary applications.

The following sections walk you through the creation of an image with Chef and its processing on a Jisto node.

6.1 About Chef

Chef is a systems framework that lets you deploy servers and applications to a physical or virtual machine or to a cloud location. Chef uses structures called cookbooks and recipes to define what exactly will be deployed. By packing the appropriate cookbooks and recipe, with a Jisto build script, run script and `config.JSON` file, you can create an image template that can be subsequently uploaded and deployed to one or more Jisto nodes.

Please note that this section presents only a subset of what can be done with Chef. On the https://www.chef.io/ web site you can find in depth information on creating cookbooks and recipes and on related topics.

6.2 Installing Chef

To create a chef cookbook that is compatible with Jisto, you first need to install the Chef Development Kit. Before doing this, choose as the Chef system, a machine that runs the same version of Linux as the one where you normally create Jisto images. If you are not sure of the version of Linux to use, use the `cat /etc/redhat-release` command to display the version of the operating system.

After doing this, use the following procedure to install Chef:

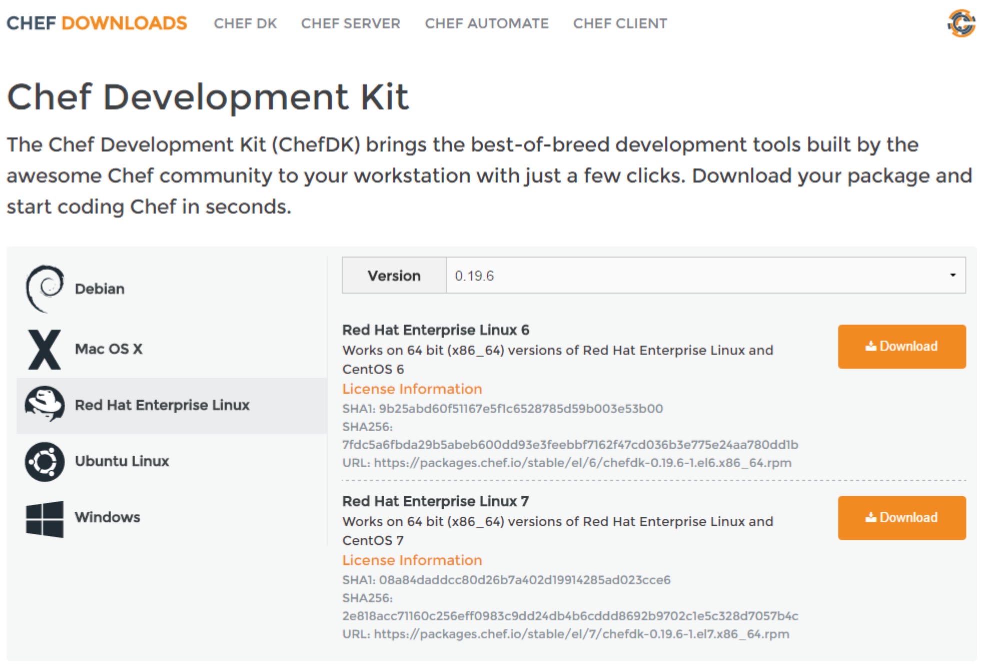
1. Browse to the following web site:

   `https://docs.chef.io/install_dk.html`

2. If you have not already familiarized yourself with Chef, please do so on this site. Then browse to the following site:

   `https://downloads.chef.io/chef-dk/`

3. The Chef web site lets you choose from development kits intended for different environments, as shown below:



Chef Development Kit

The Chef Development Kit (ChefDK) brings the best-of-breed development tools built by the awesome Chef community to your workstation with just a few clicks. Download your package and start coding Chef in seconds.

Version 0.19.6

Red Hat Enterprise Linux 6
Works on 64 bit (x86_64) versions of Red Hat Enterprise Linux and CentOS 6
License Information
SHA1: 9b25abd60f51167e5f1c6528785d59b003e53b00
SHA256:
7fdc5a6fbda29b5abeb600dd93e3feebbf7162f47cd036b3e775e24aa780dd1b
URL: https://packages.chef.io/stable/el/6/chefdk-0.19.6-1.el6.x86_64.rpm

Red Hat Enterprise Linux 7
Works on 64 bit (x86_64) versions of Red Hat Enterprise Linux and CentOS 7
License Information
SHA1: 08a84daddcc80d26b7a402d19914285ad023cce6
SHA256:
2e818acc71160c256eff0983c9dd24db4b6cddd8692b9702c1e5c328d7057b4c
URL: https://packages.chef.io/stable/el/7/chefdk-0.19.6-1.el7.x86_64.rpm

Figure 20 – Chef Download Page

4. If your Linux system is running Red Hat Enterprise 6 or 7 or CentOS 6 or 7, for example, select the Red Hat Enterprise Linux option. The page that displays lets you download a development kit that is appropriate for any of these operating systems.
5. Determine which option to select and then click the **Download** button. After the development kit has been downloaded, copy the rpm file to the Linux system where you will install the development kit if necessary.
6. Install the rpm file on the Linux system, using the following command:

```
sudo rpm -Uvh chefdk-0.19.6-1.el7.x86_64.rpm
```

6.3 Creating a Chef Cookbook, Recipe and Template

After installing Chef, you can create a cookbook as well as the certain items contained within it. To do this, use the following procedure:

1. On the Linux system where you have installed Chef, create a directory where you want to create the Jisto image, as for example with following command:

```
mkdir jisto-apache
```

2. Browse to the newly created directory:

```
cd jisto-apache
```

3. Create a sub-directory where the Chef cookbooks will be created, as follows:

```
mkdir cookbooks
```

4. Use Chef to create a cookbook for an application such as an Apache web server, for example, you can use the following command:

   ```
   chef generate cookbook cookbooks/apache
   ```

   This command will create a `recipes` directory under `cookbooks/apache`. In the `recipes` directory, Chef also creates the recipe file `default.rb`.

5. Edit the contents of the `default.rb` file appropriately to create a recipe. To create a recipe for an Apache web server, you would insert the following content:

   ```
   # This installs the httpd (Apache) web server.
   #
   package 'httpd'
   #
   # This enables and starts the http (Apache) service
   #
   service 'httpd' do
      action [:enable, :start]
   end
   #
   # This specifies the path of the default index.html
   # for the Jisto image that will incorporate the cookbook.
   #
   step
   template '/var/www/html/index.html' do
      source 'index.html.erb'
   end
   ```

6. Create a template for the default `index.html` that is referenced in the `default.rb` recipe file that was just modified. To do this, browse to the `jisto-apache` directory and enter the `chef generate` command to create the `index.html.erb` file, as shown below:

   ```
   chef generate template cookbooks/apache index.html
   ```

   Chef then displays the following set of messages:

   ```
    Recipe: code_generator::template
     * directory[cookbooks/apache/templates/default] action create
   (up to date)
     * template[cookbooks/apache/templates/index.html.erb] action
   create
       - create new file cookbooks/apache/templates/index.html.erb
       - update content in file
   cookbooks/apache/templates/index.html.erb from none to e3b0c4
       (diff output suppressed by config)hef generate template
   cookbooks/apache index.html
   ```

7. Edit the newly created `index.html.erb` file to specify the default web page to be displayed by the Apache web server. To specify a default web page, you could, for example, insert the following content:

```
<HTML><BODY>This is the Jisto web server!</BODY></HTML>
```

8. Edit the newly created `index.html.erb` file to specify the default web page to be displayed by the Apache web server. To specify a default web page, you could, for example, insert the following content:

```
<HTML><BODY>This is the Jisto web server!</BODY></HTML>
```

6.4 Creating a Jisto Image with an Embedded Cookbook

After creating a cookbook and modifying certain of its constituent files, you can create the Jisto image that will contain it. To do this, use the following procedure:

1. In the top level of the directory where you want to create the Jisto image, such as `jisto-apache` for example, create the `config.json` file. As explained in section **3.1.1 Preparing the Image Template**, the `config.json` file defines the image's name, build script, run script and current directory. For an image that contains a Chef cookbook, you need to specify where the chef client commands will be executed by means of the `installdestination` and `workingdir` parameters, as shown below:

```
{
"name": "jisto-apache",
"installsource": ".",
"installdestination": "/chef/",
"workingdir": "/chef",
"buildfile": "build.sh",
"runfile": "run.sh",
}
```

2. Create a build script in the current directory if necessary.

3. Create a run script in the current directory. The run script should start the application that is loaded into the Jisto image. In the case of a Chef cookbook, the run script needs to specify the `chef-client` command that launches a recipe. A run script that starts the apache recipe defined earlier would look like this:

```
#! /bin/bash
chef-client -z –runlist 'recipe[apache]'
tail -f /dev/null
```

4. Create the Jisto image file. Recall that an image file must be enclosed in a gz compressed tar file. This file should be created in the directory one level above the current directory, that is to say the directory where the build and run scripts are located. To create this kind of image file, run the following command in the current directory:

```
    tar zcf ../image_name.tar.gz *
```

where *image_name* is, of course, the name that you wish to assign to the image.

5. Create a directory where you can create the job source file for the Jisto image. Browse the to this directory and create the job source file for the Jisto image file, as explained in section **3.2.1 Preparing the Job Source File**. Verify that any of the ports that you specify for the image are available on Jisto nodes.

6. Package the job source file into a gz compressed tar file. To do this create the tar file in the directory just one level above the director where the job source file is located, as for example, with the following command:

   ```
       tar zcf ../job_source_file.tar.gz *
   ```

   where *job_source_file* is, of course, the name that you wish to assign to the job source file tar.

You can now upload the image and job source tar files into Jisto. When the image is processed by Jisto, it will invoke a Chef command that runs a Chef recipe, all within a container that is managed by Jisto.

7 Support

If you need help with any aspect of the Elastic Workload Manager UI or CLI, please contact Jisto support at support@jisto.com.