



JFrog

MISSION CONTROL

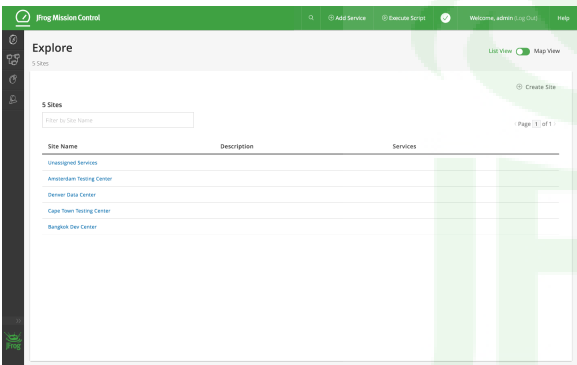
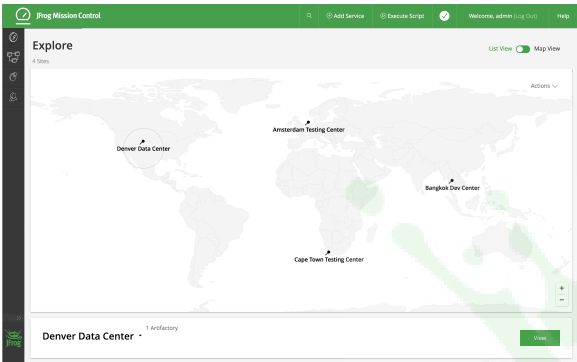
Version 2.0 User Guide

Welcome to JFrog Mission Control

Overview

JFrog Mission Control is designed to be your single access point for managing multiple services of Artifactory and Xray. It lets you view all services under your administrative control whether they are installed on your own site, or at geographically remote sites around the world. You can see the connections between them, perform operations on several instances at once, or drill down to view and configure a single instance at a time. The main modules of Mission Control are:

- **Explore:** View all sites being managed by Mission Control on a map or as a list.
- **Services:** View and manage all services under your control.
- **Graphs:** Get data about your Artifactory instances and their repositories managed by Mission Control.
- **Admin:** Manage configuration scripts, [Git integration](#), licenses, disaster recovery, users, and more.



Page Contents

- [Overview](#)
- [Support Matrix](#)
- [PDF Download](#)

Quick Links

Mission Control
REST API

Release

Read More

- [Installing Mission Control](#)
- [Upgrading Mission Control](#)
- [Running with Docker](#)
- [Configuring Mission Control](#)
- [Configuration Scripts](#)
- [Exploring Sites](#)
- [Managing Services](#)
- [Managing Licenses](#)
- [Notifications](#)
- [Graphs](#)
- [JMX MBeans](#)
- [System Monitoring](#)
- [Mission Control REST API](#)
- [JFrog CLI](#)
- [Disaster Recovery](#)
- [System Backup and Rapid Recovery](#)
- [Troubleshooting](#)
- [Release Notes](#)

Support Matrix

Mission Control provides full support for Artifactory instances running on an Enterprise (or SaaS Enterprise) license, and partial support for Artifactory instances running on an OSS or Pro license as described in the table below:

	Xray	OSS	Pro	Enterprise	AOL Dedicated Server
Explore	✓	✓	✓	✓	✓
Service view	✓	✓	✓	✓	✓
Scripting Artifactory configuration updates	✓	✗	✗	✓	✓
REST API	✓	✗	✗	✓	✓
Graphs	✗	✗	✗	✓	✓
Disaster Recovery	✗	✗	✗	✓	✓
Notification and policies	✗	✗	✗	✓	✓
License deployment	✗	✗	✗	✓	✓
License bucket	✗	✗	✗	✓	✓

Installing Mission Control

Overview

This page helps you get started using Mission Control. Assuming you comply with the [System Requirements](#) specified below, after going through the instructions on this page, you should have your instance of Mission Control configured with at least one Artifactory instance which you can monitor and configure through your Mission Control.

Mission Control supports managing [services](#) of Artifactory from version 4.5.



Running with Docker?

This page provides installation and upgrade instructions for running Mission Control as a Debian or Centos distribution.

If you are running Mission Control as a Docker container, please refer to [Running with Docker](#).

License

Mission Control itself is provided for free and has no license requirements. However, while you may view any Artifactory service from Mission Control, you may only configure and see relationships between services that are activated with an **Artifactory Enterprise license**.



Viewing Artifactory OSS, Artifactory Pro and Artifactory SaaS

You may add services of Artifactory OSS, Artifactory Pro to Mission Control, but for these instances, you can only view basic information. You cannot perform any actions on these services through Mission Control.

As a shared managed service, Artifactory SaaS is maintained for you by JFrog and can not be managed by Mission Control.

However, if you are running a Artifactory SaaS as a **dedicated server**, this can be fully managed by Mission Control.

System Requirements

Hardware

JFrog Mission Control requires the following hardware:

- Processor: 6 cores
- RAM Memory: 8 GB
- Storage: 100 GB

Platforms

JFrog Mission Control supports any non-Windows platform that can run Docker v1.11 and above. In addition, it has been tested and verified to run as a non-Docker installation on the following 64-bit flavors of Linux:

- Debian 8.x
- Centos 7.x
- Ubuntu 16.x
- Red Hat 7.x

Page Contents

- [Overview](#)
- [License](#)
- [System Requirements](#)
 - [Hardware](#)
 - [Platforms](#)
 - [Java requirements](#)
 - [PHP requirements](#)
 - [SELinux](#)
 - [Browsers](#)
- [Download](#)
- [Installation](#)
 - [Docker Installation](#)
 - [Debian Installation](#)
 - [System library requirements](#)
 - [Installation Instructions](#)
 - [CentOS Installation](#)
 - [System library requirements](#)
 - [Installation Instructions](#)
 - [Ubuntu Installation](#)
 - [System library requirements](#)
 - [Installation Instructions](#)
 - [Red Hat Installation](#)
 - [System library requirements](#)
 - [Installation Instructions](#)
- [Installation File Structure](#)
- [Mission Control File structure](#)
 - [Home directory](#)
- [Microservices and Ports](#)
 - [Docker Installation](#)
 - [Linux Installation](#)
- [Data](#)
- [Status of Installation](#)
- [Uninstalling](#)
- [Secure Access With SSL](#)
- [Accessing Mission Control](#)

Read More

- [Managing Third-Party Components](#)
- [Using External Databases](#)
- [Elasticsearch Usage Guide](#)
- [Migrating Data From InfluxDB to Elasticsearch](#)

Java requirements

You must run Mission Control with **JDK 8**.



You can download the latest JDK from the [Oracle Java SE Download Site](#).



JAVA_HOME and JRE_HOME

Make sure your JAVA_HOME environment variable correctly points to your JDK 8 installation.

If you also have JRE_HOME defined in your system, this will take precedence over JAVA_HOME and therefore you need to, either point JRE_HOME to your JDK 8 installation, or remove the JRE_HOME definition.

PHP requirements

Mission Control uses PHP to support its scripting functionality and requires **PHP 5.6** on Debian and **PHP 5.4** on Centos.

SELinux

Centos flavors may have issues starting MongoDB because of SELinux restrictions. Please refer to the [MongoDB documentation](#) for instructions on configuring SELinux.

Browsers

Mission Control has been tested with the latest versions (known at the time of release) of Google Chrome, Firefox and Safari.

Download

The latest version of Mission Control is freely available for download from the [Mission Control Download Page](#).

Installation



No spaces

Mission Control offers a variety of options for installation on different platforms.

In all cases, make sure that the full path to the installation folder does not contain any spaces.

Docker Installation

Mission Control is available for download as a Docker image to be run as a container. For full details, please refer to [Running with Docker](#).

Debian Installation

JFrog Mission Control currently supports Debian 8.x.

Do not install Mission Control on a higher version of Debian as it has not been validated to work.

System library requirements

Mission control needs the following libraries to be present as run-time dependencies. Please ensure these are available before you begin installation.

- libcurl3
- libltdl7
- php5-fpm
- net-tools

When you install without these dependencies, the installer displays an error indicating that a number of dependencies are missing, prompting you to install them along with the install command.

Installation Instructions

Once you have [downloaded](#) Mission Control, installing it is very straightforward:

1. Extract the contents of the compressed file

Installing Mission Control

```
tar -xvf jfmc-debian-<version>.tar.gz
```

2. Run the installer

Installing Mission Control

```
cd jfmc-debian-<version>
./installJFMC-debian.sh
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using it's own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

3. The log file for the installation will be in a file installJFMC-debian.<timestamp>.log.
4. A control file is created as part of the installation. Start Mission control using this file

Starting Mission Control

```
For v2.0.0 to v2.1.0,
/opt/jfrog/jfmc/scripts/jfmc.sh start

For v2.1.1,
/opt/jfrog/mission-control/scripts/jfmc.sh start
```

CentOS Installation

JFrog Mission Control currently supports CentOS 7.x.

Do not install Mission Control on a higher version of CentOS as it has not been validated to work.

System library requirements

Mission control needs the following libraries to be present as run-time dependencies. Please ensure these are available before you begin installation.

- openssl
- php-fpm
- net-tools

Installation Instructions

Once you have [downloaded](#) Mission Control, installing it is very straightforward:

1. Extract the contents of the compressed file

Installing Mission Control

```
tar -xvf jfmc-centos-<version>.tar.gz
```

2. Run the installer

Installing Mission Control

```
cd jfmc-centos-<version>
./installJFMC-centos.sh
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using it's own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

3. A control file is created as part of the installation. Start Mission control using this file

Starting Mission Control

```
For v2.0.0 to v2.1.0,
/opt/jfrog/jfmc/scripts/jfmc.sh start
```

```
For v2.1.1,
/opt/jfrog/mission-control/scripts/jfmc.sh start
```

Ubuntu Installation

JFrog Mission Control currently supports Ubuntu 16.x.

Do not install Mission Control on a higher version of Ubuntu as it has not been validated to work.

System library requirements

Mission control needs the following libraries to be present as run-time dependencies. Please ensure these are available before you begin installation.

- libcurl3
- libltdl7
- php5.6-fpm
- net-tools

In order to install "php5.6-fpm" please run the below commands:

```
apt-get install python-software-properties
add-apt-repository ppa:ondrej/php
apt-get update
apt-get install -y php5.6-fpm
```

Installation Instructions

Once you have [downloaded](#) Mission Control, installing it is very straightforward:

1. Extract the contents of the compressed file

Installing Mission Control

```
tar -xvf jfmc-ubuntu-<version>.tar.gz
```

2. Run the installer

Installing Mission Control

```
cd jfmc-ubuntu-<version>  
./installJFMC-ubuntu.sh
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using it's own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

3. The log file for the installation will be in a file `installJFMC-ubuntu.<timestamp>.log`.
4. A control file is created as part of the installation. Start Mission control using this file

Starting Mission Control

```
For v2.0.0 to v2.1.0,  
/opt/jfrog/jfmc/scripts/jfmc.sh start
```

```
For v2.1.1,  
/opt/jfrog/mission-control/scripts/jfmc.sh start
```



If you are having any errors with PostgreSQL installation, then please check the following log files "bitrock_installer.log" and "install-postgresql.log" under /tmp/ folder

Red Hat Installation

JFrog Mission Control currently supports Red Hat 7.x.

Do not install Mission Control on a higher version of Red Hat as it has not been validated to work.

System library requirements

Mission control needs the following libraries to be present as run-time dependencies. Please ensure these are available before you begin installation.

- openssl
- php-fpm
- net-tools

Installation Instructions

Once you have [downloaded](#) Mission Control, installing it is very straightforward:

1. Extract the contents of the compressed file

Installing Mission Control

```
tar -xvf jfmc-redhat-<version>.tar.gz
```

2. Run the installer

Installing Mission Control

```
cd jfmc-redhat-<version>  
./installJFMC-redhat.sh
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using it's own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

3. A control file is created as part of the installation. Start Mission control using this file

Starting Mission Control

```
For v2.0.0 to v2.1.0,  
/opt/jfrog/jfmc/scripts/jfmc.sh start
```

```
For v2.1.1,  
/opt/jfrog/mission-control/scripts/jfmc.sh start
```

Installation File Structure

After downloading and extracting the installer, the following file structure is created under the installation folder

<i>install-<flavor>.sh</i>	The installation script for specific Linux flavor (Debian/Centos)
<i>config</i>	A folder containing files necessary to configure 3rd party services like MongoDB, PHP-FPM, etc.
<i>packages</i>	A folder containing the actual packages to install (deb or rpm files)
<i>seed_data</i>	A folder containing scripts necessary to seed users/data into 3rd party services
<i>migration</i>	A folder containing scripts necessary to migrate from earlier versions of Mission control
<i>version.sh</i>	A file containing the version of the JFMC

Mission Control File structure

Home directory

The Mission Control home (/opt/jfrog/) directory will contain files necessary to start and stop all the micro-services associated with JFrog Mission Control

<i>mission-control/bin</i>	Mission Control service files
<i>mission-control/lib</i>	The Mission Control runtime JAR file
<i>mission-control/scripts</i>	Control scripts for Mission Control

Microservices and Ports

Docker Installation

Mission Control's Docker installation only needs port 8080 to be exposed on the host to function.

If the port is occupied, Mission control will throw an error on start.

To change the port that the Docker installation uses, update the **server.port** property to another value (between 0 and 65535) in the *\$JFMC_HOME/jfmc/etc/mission-control.properties* file.

Linux Installation

Mission Control runs a number of microservices with specific port allocations as described in the table below.

If a port is already in-use, the installer will display a warning, and in some cases, prompt for a different port.

Microservice	Port	Purpose	Service name in Debian and Centos Installs
Mission Control Server	8080	Core Mission Control service	mission-control.service
Scheduler	8085	Manages scheduling for different internal Mission Control tasks	jfi-scheduler.service
Executor	8087	Executes tasks to collect data from services	jfi-executor.service
Graphs Core	8090	Graphs core functions	jfi-core.service
	8089	Graphs core functions over SSL	
Elasticsearch	9200	Data service used for time-series data to generate graphs	elasticsearch.service
	9300	Transport client port for bulk inserts	
Mongo	27017	Data service used for storing non-time series data	mongod.service
Postgres	5432	Data service used by the scheduler microservice	postgresql-9.6.service
influx	8088	Maintained to manage migration from version 1.x to version 2.x <i>Deprecated post 2.1.1</i>	influxdb.service

Data

The Mission Control data folder (usually, */var/opt/jfrog/mission-control*) will contain files created and used by each of the micro-services.



Log location

Logs are written into data folder (usually, */var/opt/jfrog/mission-control/logs*) and **log rotation is not enabled**. It is recommended to turn on log rotation.

Status of Installation

Use the following command to check the installation status. Once installation is complete, the same command will provide the status of the services as well.

Mission Control services status

For v2.0.0 to v2.1.0,
`/opt/jfrog/jfmc/scripts/jfmc.sh status`

For v2.1.1,
`/opt/jfrog/mission-control/scripts/jfmc.sh status`

Uninstalling

Use the control file to initiate uninstalling JFrog Mission Control as follows:

Please note that this will not uninstall third-party components installed as part of installation. They will have to be uninstalled manually.

Removing the Mission Control services

For v2.0.0 to v2.1.0,
`/opt/jfrog/jfmc/scripts/jfmc.sh removeServices`

For v2.1.1,
`/opt/jfrog/mission-control/scripts/jfmc.sh removeServices`

Secure Access With SSL

JFrog Mission Control supports secure access with SSL. The following example shows how to enable access with SSL using a JKS keystore:

1. Stop JFrog Mission Control
2. Consult your Certificate Authority and generate a certificate for your instance of Mission Control
3. Modify your `$MC_HOME/etc/mission-control.properties` file as follows:

- a. Comment the line specifying 8080 as the server port (`server.port=8080`) and uncomment the line specifying 8443 as the server port. When done you should have:

```
# server.port=8080 This line is commented
server.port=8443
```

- b. Set the path to your keystore in the `server.ssl.key.store` property. For example:

```
server.ssl.key-store=path/to/keystore.jks
```

- c. Uncomment and set the keystore password property:

```
server.ssl.key-store-password=<Keystore password>
```

- d. Uncomment and set the keystore type property:

```
server.ssl.key-store-type=JKS
```

- e. Save the changes to your `mission-control.properties` file.

4. Start JFrog Mission Control

Once you have completed this configuration, you can access JFrog Mission Control through the server port specified in the `mission-control.properties` file.

For example, using the above configuration, you could access Mission Control via SSL using the following URL:

```
https://<mission-control-server-ip>:8443
```

Accessing Mission Control

Mission Control can be accessed through your browser using the following URL:

```
http://SERVER_DOMAIN:<server port>
```

The default port used by Mission Control is 8080, so a default installation would be accessed at <http://localhost:8080>.



Using a different port

To change the port, in *\$MC_HOME/etc/mission-control.properties*, set `server.port=<port number>`.



Managing Third-Party Components

Overview

Mission Control works with a number of third-party services including various databases. These include:

- **MongoDB** - used for internal Mission Control operations
- **PostgreSQL** - used for internal scheduler and other services
- **Elasticsearch** - used to store historical data

The following sections show how to manage the configuration for these components.

Page Contents

- [Overview](#)
- [Changing Database Credentials](#)
 - [MongoDB](#)
 - [PostgreSQL](#)
 - [Elasticsearch](#)

Changing Database Credentials

Mission Control works with various databases which come pre-configured with default credentials for access by Mission Control. The following sections show how to change these default credentials.

MongoDB

Mission control creates and uses three MongoDB databases for its operation as described in the following table:

Database name	user	password	role
insight_CUSTOM_	jfrog_insight	password	dbOwner
insight_team	jfrog_insight	password	dbOwner
mission_platform	mission_platform	password	dbOwner

To change the default credentials for any of these databas, please use the following steps.

1. [Change the MongoDB password](#)
2. [Change the graph_core service password](#)
3. [Update the Mission Control properties file with the new credentials](#)

Changing the Password in MongoDB

```
# Access MongoDB as the each user above
$ mongo --port 27017 -u "jfrog_insight" -p "password" --authenticationDatabase "insight_CUSTOM_"

# Switch to the corresponding database above
$ use insight_CUSTOM_

# Update the credentials
$ db.updateUser("jfrog_insight",{pwd: "<new_password>"})

# Verify the update was successful by logging in with the new credentials
$ mongo --port 27017 -u "jfrog_insight" -p "<new_password>" --authenticationDatabase "insight_CUSTOM_"
```



Credentials must match

Note that the **insight_CUSTOM_** and **insight_team** databases must always use the same username and password.

Changing the Password in the Graph_Core Service

Update the new credentials into the Mission Control service using the following REST API endpoint

```
# On the host running the services run the following command
$ curl 'localhost:8088/api/settings' -d 'action=set&key=integrations.mongodb.url&value=mongodb:27017' -X POST
```

Update the Mission Control Properties File

On the host running the services run the following command:


```
$JFMC_HOME/etc/mission-control.properties, uncomment/set:  
spring.data.mongodb.username  
spring.data.mongodb.password
```

PostgreSQL

The default credentials for the internal PostgreSQL database used by Mission Control is:

username: quartzdb

password: insight

To change the default credentials used by Mission control to access its internal PostgreSQL database, you need to log into the database as the "quartzdb" user and change the password as follows:

1. [Change the password in PostgreSQL](#)
2. [Change the password in the scheduler service](#)

Changing the Password in PostgreSQL

```
# Access PostgreSQL as the quartzdb user adding the optional -W flag to invoke the password prompt  
$ psql -d quartzdb -U quartzdb -W  
  
# Securely change the password for user "quartzdb". Enter and then retype the password at the prompt.  
\password <new_password>  
  
# Verify the update was successful by logging in with the new credentials  
$ psql -d quartzdb -U quartzdb -W
```

Changing the Password in the Scheduler Service

```
# On the host running the services run the following command OR within the container (if you are using  
docker installation)  
$cd $JFI_HOME_SCHEDULER/_MASTER_/data/contexts/settings/  
  
$quartz.properties, uncomment/set:  
  
org.quartz.dataSource.quartzDataSource.user = username  
org.quartz.dataSource.quartzDataSource.password = password
```

Note: Make sure to change authentication settings in the `pg_hba.conf` to ensure the policies you need are set appropriately.

Elasticsearch

After Mission Control is fully installed, the Elasticsearch binaries and data files can be found under:

<JFMC_HOME/elasticsearch>.

Mission Control currently uses Elasticsearch without x-pack, therefore, authentication is not needed to access the history database.

Using External Databases

Overview

JFrog Mission Control uses several databases for different features of its operation.

- **Elasticsearch v5.5.2** stores analytics data used to create [Graphs](#).
- **PostgreSQL v9.6** is used for continuous data import and scheduling
- **MongoDB v3.2.6** is used for configuration management

Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **Elasticsearch**, **PostgreSQL** or **MongoDB** databases if you have these already installed and in use in your organization.

It is up to you to choose which, if any of these databases to externalize when you install Mission Control.

During the installation process, the Mission Control installation script will first ask if you want to perform a standard installation:

```
Perform a standard Installation? [Y/n]:
```

If you respond "Y", the installation process will automatically run to completion and install internal databases for Mission Control to use.

If you respond "n", then for each of the three databases, Mission Control will ask whether you want to use the internal database or an external one you are already using in your organization.



You take full responsibility for your own databases

If you choose to have Mission Control use any of your own databases for its operation, you take full responsibility for the maintenance, monitoring, backup and correct functioning of these databases.

If you do externalize any of the databases, Mission Control will also ask if you want it to seed the database or prefer to do it manually.

Page Contents

- [Overview](#)
- [Externalizing Elasticsearch](#)
 - [Manually Seeding Elasticsearch](#)
- [Externalizing MongoDB](#)
 - [Manually Seeding MongoDB](#)
- [Externalizing PostgreSQL](#)
 - [Manually Seeding PostgreSQL](#)
- [Externalizing Databases on an Existing Installation](#)
- [Changing Externalized Databases](#)

Externalizing Elasticsearch

To externalize the Elasticsearch database, respond to the prompts as described below:

Prompt	Response
Install Elasticsearch? [Y/n]:n	"n"
Please enter the Elasticsearch URL [http://docker.for.mac.localhost:9200]:	Provide the URL to your Elasticsearch database or accept the default if that is correct.
Does this Elasticsearch instance need credentials?	"Y" if your Elasticsearch database requires credentials, or "n" if it allows anonymous access.
Please enter the User ID: Please enter the Password:	Provide your Elasticsearch user ID and password if required. If you want the installer to seed your database automatically, make sure this user has privileges to create templates, aliases and indices.

Attempt to seed Elasticsearch?
[y/N]:

- If you respond with "y", the installer will attempt to perform the following actions to seed the database automatically:
 - create the necessary templates and indices
 - copy all files required to seed the Elasticsearch database to a separate folder to enable manual seeding at a later time should that be necessary

If the automatic seeding operation fails, the installer will display a prompt asking if you want to **retry**, **abort** or **skip**.

- If you respond with "N", the installer will copy all files required to seed the Elasticsearch database to a separate folder to enable manual seeding

Manually Seeding Elasticsearch



Seed the database before starting JFrog Mission Control

If you choose to seed your database manually, make sure you do so BEFORE starting up JFrog Mission Control .

To use an external Elasticsearch database, Mission Control requires connectivity to the database and the presence of certain templates and aliases. These are created by the **createIndices.sh** script provided in the installation package using the following process:

- Make the **createIndices.sh** file executable (`chmod +x createIndices.sh`)
- Create the following environment variables (with appropriate values)
 - ELASTIC_SEARCH_URL
 - ELASTIC_SEARCH_USERNAME (optional)
 - ELASTIC_SEARCH_PASSWORD (optional)
- Execute the file (`./createIndices.sh`)

Externalizing MongoDB

To externalize the MongoDB database, respond to the prompts as described below:

Prompt	Response
Install MongoDB? [Y/n]:n	"n"
Please enter the MongoDB Host [docker.for.mac.localhost]:	Enter the host that MongoDB is available on. (Usually, 'localhost'). NOTE: Do NOT include a protocol.
Please enter the MongoDB Port [27017]:	Enter the port that Mission Control can use to access MongoDB. (Usually, 27017)

<div>  Docker installation does not try to automatically seed the database </div> <p>In a Docker installation, the installer does not attempt to seed the external MongoDB database. Instead, it copies the files needed for manual seeding as described in Manually Seeding MongoDB below and the automatic installation process ends here.</p> <p>Attempt to seed MongoDB? [y/N]:</p>	<p>To have the installer seed your MongoDB automatically, respond with "y"</p> <div>  MongoDB must be on the same machine for automatic seeding </div> <p>The installer can only seed a MongoDB database automatically if it is on the same machine as the installer.</p> <p>If your MongoDB is installed on another machine, respond with "N".</p> <p>If you respond with "N", the installer copies the files you will need to manually seed the database and ends the installation process.</p> <p>If you respond with "y" the installer continue and prompt you with the following questions.</p>
<p>Is this a fresh installation with no users? (If you choose 'y', the installer will create an admin user):</p>	<p>If you respond "y", the installer will prompt you for an admin user and password</p>
<p>Please enter the MongoDB admin user ID:</p> <p>Please enter the MongoDB admin user password:</p>	<p>Provide an admin user ID and password</p> <p>The installer will attempt to perform the following actions to seed the MongoDB automatically:</p> <ul style="list-style-type: none"> • create the necessary databases and users • copy all files required to seed the MongoDB database to a separate folder to enable manual seeding at a later time should that be necessary <p>If the automatic seeding operation fails, the installer will display a prompt asking if you want to retry, abort or skip.</p>

Manually Seeding MongoDB



Seed the database before starting JFrog Mission Control

If you choose to seed your database manually, make sure you do so BEFORE starting up JFrog Mission Control .



MongoDB is used to store metadata about Mission Control's microservices, so the script attempts to create the necessary databases and users. If you are familiar with MongoDB or do not have access to the Mongo instance, you can review the `createMongoUsers.js` file and create these yourself using the appropriate database client. If not and if you have access to the instance where MongoDB is installed, follow the instructions below:

- Copy the files `createMongoUsers.sh` and `createMongoUsers.js` to the system where MongoDB is running.
- Make the shell file executable (`chmod +x createMongoUsers.sh`)
- Execute the file `./createMongoUsers.sh` and follow the prompts on screen.

Externalizing PostgreSQL

To externalize the PostgreSQL database, respond to the prompts as described below:

Prompt	Response
--------	----------

Install Postgres? [Y/n]:n	"n"
Please enter the Postgres Host [docker.for.mac.localhost]:	Enter the host that Postgres is available on. NOTE: Do NOT include a protocol.
Please enter the Postgres Port [5432]:	Enter the port that Mission Control can use to access Postgres. (Usually, 5432)
<div>  Docker installation does not try to automatically seed the database In a Docker installation, the installer does not attempt to seed the external Postgres database. Instead, it copies the files needed for manual seeding as described in Manually Seeding Postgres below and the automatic installation process ends here. </div> <p>Attempt to seed Postgres? [y/N]:</p>	<p>To have the installer seed your Postgres automatically, respond with "y"</p> <div>  Postgres DB must be on the same machine for automatic seeding The installer can only seed a Postgres database automatically if it is on the same machine as the installer. If your Postgres database is installed on another machine, respond with "N". </div> <p>If you respond with "N", the installer copies the files you will need to manually seed the database and ends the installation process.</p> <p>If you respond with "y" the installer continue and prompt you with the following questions.</p>
Please enter the path where Postgres executable (psql) is available:	Provide the path to your psql executable
Please enter the ID of the root user:	Provide a root user ID and password
Please enter the password of the root user:	<p>The installer will attempt to perform the following actions to seed the Postgres automatically:</p> <ul style="list-style-type: none"> • create the necessary databases and users • copy all files required to seed the Postgres database to a separate folder to enable manual seeding at a later time should that be necessary <p>If the automatic seeding operation fails, the installer will display a prompt asking if you want to retry, abort or skip.</p>

Manually Seeding PostgreSQL



Seed the database before starting JFrog Mission Control

If you choose to seed your database manually, make sure you do so BEFORE starting up JFrog Mission Control .

Postgres is used to store jobs by Mission Control's scheduling service, so the script attempts to create a database (default value: **quartzdb**), a user and password (default values: **quartzdb:password**).

The script then attempts to create several tables and indices.

If you are familiar with Postgres or the externalized instance is managed by another user, you can use any database client to create the necessary database and users and then create the tables in the file **quartz_postgres.sql**.

If not, and if you have access to the instance where Postgres is installed, follow the instructions below:

- Copy the files `createPostgresUsers.sh` and `quartz_postgres.sql` to the system where Postgres is running.
 - Make the shell file executable (`chmod +x createPostgresUsers.sh`)
 - Execute the file `./createPostgresUsers.sh` and follow the prompts on screen
-

Externalizing Databases on an Existing Installation

You can externalize the databases Mission Control uses on an existing installation at any time by simply running the installer again.

When prompted with **Perform a standard upgrade?** [Y/n], select **"N"**.

Now just continue with the process in the same way you would during a new installation as described in [Using External Databases](#) above.

Changing Externalized Databases

Mission Control offers you the flexibility to decide when to externalize its databases, and even to switch externalized databases in case you have more than one instance of any particular database installed in your system.

When you do externalize a database, the installer creates a file in the installation log folder (`$JFMC_DATA/installer`) to indicate if the database has been seeded. If, when changing the externalized database, you want the installer to try seeding the new database, make sure to delete the corresponding file before running the installer.



Elasticsearch Usage Guide

Overview

[Elasticsearch](#) is a highly scalable search and analytics engine. It is used to store and retrieve historical data for Artifactory services and their repositories, and provide it to Mission Control to display in its [Graphs module](#) as usage metrics.

Version

JFrog Mission Control currently uses **ElasticSearch version 5.6**.

Resources

- [Installation Guide](#)
- [Getting Started](#)
- [Snapshot and Restore](#)

Page Contents

- [Overview](#)
 - [Version](#)
 - [Resources](#)
- [How Elasticsearch is Packaged within Mission Control](#)
- [Installation Structure](#)
- [Ports](#)
- [Indexes and Aliases](#)
- [Index Cleanup](#)

How Elasticsearch is Packaged within Mission Control

Elasticsearch is packaged into the [Mission Control installation](#). The following describes the different variations for each distribution package type:

Distribution	Packaging
Docker	Elasticsearch for Docker is added as a Docker container to the Mission Control docker-compose project. This installs version 5.5.2.
Debian	Elasticsearch 5.6.2 is included in the Mission Control Debian project. Linux service files are added.
RPM	Elasticsearch 5.6.2 is included in the Mission Control RPM project. Linux service files are added.

Installation Structure

After Mission Control is fully installed, the Elasticsearch data files can be found in the following locations:

File	Location
Binaries	<JMFC_HOME>/elasticsearch/
Data files	Linux: <JMFC_HOME>/elasticsearch/ Docker: <JMFC_HOME>/elasticsearch/

Ports

Elasticsearch uses the following communication ports:

Service	Port
HTTP API	9200
Java Client	9300

Indexes and Aliases

There are two aliases to store and retrieve data:

Alias Name	Description
------------	-------------

active_insight_data	Used point to active index to push data.
search_insight_data	Used to search and retrieve data

On installation, these aliases and indices of format `active_insight_data_timestamp*` are created.

Index Cleanup

Mission Control is pre-configured to periodically cleanup indexes to keep data for a period of one year.



Migrating Data From InfluxDB to Elasticsearch

Overview

Mission Control 1.x used InfluxDB to store historical data on Artifactory's usage and storage. Starting from version 2.0, Mission Control will use a new Elasticsearch database to store this data.

After completing your [Mission Control upgrade to version 2.0](#), the migration of data from InfluxDB to Elasticsearch can be started.



Migration is Optional

If you're **not** interested in your historical data, there is no need to migrate it. Mission Control will start collecting data using Elasticsearch from version 2.0.



After the migration is complete the InfluxDB will continue to be available for use if you require it.



The migration script uses Gawk 3 and above. For more information, refer to the [Gawk user guide](#).

Page contents

- [Overview](#)
- [Getting Started](#)
 - [Script User Inputs](#)
 - [Migration Folder Structure](#)
- [Triggering the Migration](#)
 - [Docker Installation](#)
 - [Debian and RPM Installation](#)
- [Troubleshooting](#)
 - [Common errors and resolutions](#)

Getting Started

The following script execution process will take place once the migration is triggered:

1. **Historical data is exported from InfluxDB to CSV files.**
The script will try to sequentially export data from the *hour_data_policy*, *day_data_policy*, and *week_data_policy* and *year_data_policy* tables in the database. Any issues in data export will be reported during this step.
2. **The CSV files are converted and imported into Elasticsearch.**
This script is interactive and will ask for user inputs.

Script User Inputs

The migration script will prompt you to enter the following parameters:

Parameters	Data required
Provide a working directory for writing the logs folder <code><./influxmigration/></code> .	A working directory to store logs and other data files. Press <code>Enter</code> to use the default <code>./influxmigration/</code> folder.
Provide the baseurl for Elasticsearch <code><http://elasticsearch:9200/></code> .	The URL end point for Elasticsearch. For a Docker based installation, press <code>Enter</code> to use the default value (<code>http://elasticsearch:9200/</code>). For Debian and RPM installs, provide the path where Elasticsearch is installed. Usually this would be on the localhost (<code>http://localhost:9200/</code>).
Provide the username for authentication with Elasticsearch.	The Elasticsearch username for authentication. If no authentication was used in Elasticsearch, press <code>Enter</code> to bypass this step.
Provide the password for authentication with Elasticsearch.	The Elasticsearch password. This appears only if a username was provided in the previous step.
Provide the full path to the CSV location for conversion <code><./influxmigration/csv/></code> .	The path from which to pick the CSV files. Press <code>Enter</code> to use the default path or provide the correct path if you have changed the working directory in the previous step.

Once the script is complete, it will summarize the information provided and will prompt for a confirmation before the import action is triggered.



Troubleshooting

Check the troubleshooting section below for any errors with the execution.

Migration Folder Structure

Intermediate files and logs are stored in the following locations:

Files	Path
Migration logs	<jfmc_installation_folder>/influxmigration/logs
Exported CSV data	<jfmc_installation_folder>/influxmigration/csv
JSON chunks	<jfmc_installation_folder>/influxmigration/data

Triggering the Migration

Migrating historical data from InfluxDB to Elasticsearch can be triggered using the migration commands below.

During the migration process, you will need to provide the input parameters as described under [Script Input Parameters](#).

- InfluxDB and Elasticsearch need to be running. You can refer the the [InfluxDB JFrog User Guide](#) to get more information.

Docker Installation

Navigate to the Mission Control installation folder and execute the migrate command:

```
cd <jfmc_installation_folder>
mission-control migrateToElastic
```

Debian and RPM Installation

Navigate to the Mission Control installation folder and execute the migrate command:

Note: The Influx installation folder (**influxdb_installation_folder**) is generally at `/opt/jfrog/mission-control/influxdb/usr/bin`.

```
cd <jfmc_installation_folder>
cp migration/* <influxdb_installation_folder>
cd <influxdb_installation_folder>
./migrateInfluxToElastic.sh
```

Troubleshooting

Detailed run logs will be available in the configured logs folder. The log for the current execution is located here: `<installation_folder>/influxmigration/logs/log.out`.

If the script stops executing without any errors, check the log file for more details.

Common errors and resolutions

Cause	This happens in Debian/RPM installs where the script cannot find the InfluxDB executable used to export data to CSV format.
Resolution	Move the migration script to the InfluxDB installation folder before executing. Usually this folder is located here: <code>/opt/jfrog/mission-control/influxdb/usr/bin</code>

Cause	Missing InfluxDB data folder
Resolution	Issue the following command: <pre>curl -G "http://localhost:8086/query?pretty=true" --data-urlencode "q=show databases"</pre> <p>This should show you if you have the mission_control InfluxDB. If you don't, it likely that you have not copied the InfluxDB data directory.</p>

Cause	This happens when data exported from InfluxDB to CSV files has failed for some reason.
Resolution	Check previous errors if any and re-execute the script.

Cause	This happens because Gawk needs to be upgraded.
Resolution	Install Gawk version 3 and higher.

Cause	This happens when the script has detected an earlier run of the migration script. More details will be provided following this message in the standard out.
Resolution	In such cases, a repeated run of the script will create an extra index in Elasticsearch with the same time-series data. Although this use-case is undesirable, it will not affect the data integrity in the database. Even if there are multiple runs of migration, since the inserted data is of the same time-series, it will not affect the graphs in the UI



Upgrading Mission Control

Overview

The procedure to upgrade Mission Control depends on your installation type. We strongly recommend reading through this page before proceeding with your upgrade. Detailed upgrade instructions are provided below for the following installation types:

- Docker
- Standalone ZIP file
- CentOS
- Ubuntu
- Red Hat
- Debian

Page Contents

- [Overview](#)
- [Download](#)
 - [Docker Upgrade](#)
 - [Standalone Upgrade](#)
 - [Centos Upgrade](#)
 - [Ubuntu Upgrade](#)
 - [RedHat Upgrade](#)
 - [Debian Upgrade](#)
- [Migrating Your Data](#)

Download

You can download the latest version of Mission Control in all its formats from its [download page](#) on JFrog Bintray.



Removing InfluxDB

If you are upgrading from a previous version of Mission Control that used InfluxDB to store graph data, the upgrade script will suggest removing the InfluxDB installation.

Docker Upgrade

To upgrade Mission Control that is run as a Docker installation, please refer to [Running with Docker](#).

Standalone Upgrade



Upgrading from versions below 1.3 to versions 1.3 and above?

When upgrading from versions below 1.3 to version 1.3 and above, you first need to delete `$MC_HOME/data/storage/instanceConfigDescriptors` before you proceed with the upgrade.



Upgrading a standalone or ZIP installation from below version 1.6 to 2.x

From version 2.0, Mission Control does not support a standalone or ZIP installation. If you are running a standalone or ZIP installation of Mission Control that is below version 1.6, upgrading to version 2.x is a two step process:

1. First upgrade to version 1.6 as a Docker installation.
2. Then upgrade the Docker installation to the latest version

For details, please refer to [Upgrading Mission Control](#) under [Running with Docker](#).

Upgrading Mission Control is a simple process during which all of your instance data, repository data and configuration scripts remain intact.

Upgrading Mission Control involves the following basic steps:

1. [Backing up files](#)
2. [Installing the new version](#)
3. [Replacing backed-up files](#)
4. [Reconnecting to managed instances](#)

Backing Up Files

If you have modified either of `$MC_HOME/etc/mission-control.properties` or `$MC_HOME/etc/logback.xml`, save a copy of these files in a temporary location.

Installing the New Version

To install the new version of Mission Control, unzip the distribution archive in a temporary location and replace the following files and folders with the corresponding ones from the newly unzipped archive.

under \$MC_HOME :

1. \$MC_HOME/data
2. \$MC_HOME/etc
3. \$MC_HOME/logs

Replacing Backed-up Files

Replace the files you backed up in the upgraded installation.

Reconnecting to Services

To reconnect to Artifactory following an upgrade, you need to upload the Mission Control extensions to Artifactory as follows:

When [viewing the details of a service](#), from the **Actions** menu, click **Upload Extensions**.

Add Services to Sites

Once you have reconnected to services, make sure to add them to sites in your system so they can be fully managed by Mission Control

Repeat this for all the Artifactory services being managed by Mission Control.

Centos Upgrade



Upgrading from versions below 1.3 to versions 1.3 and above?

When upgrading from versions below 1.3 to version 1.3 and above, your first step should be to delete `$MC_HOME/data/storage/instanceConfigDescriptors` before you proceed with the upgrade.

To upgrade Mission Control that is run as an Centos, download the latest version and browse to its location on your file system.

Execute the following command:

Centos Upgrade

```
sudo su
service mission-control stop
tar -xvf jfmc-centos-<version>.tar.gz
cd jfmc-centos-<version>
./installJFMC-centos.sh
```



Externalizing a Database?

Mission Control gives you the option to externalize one or more of its databases during an upgrade.

To externalize a database, when prompted with `Perform a standard Installation? [Y/n]:`, respond with "n" and then follow the prompts as described in [Using External Databases](#).

Ubuntu Upgrade

To upgrade Mission Control that is run as a Ubuntu installation, download the latest version and browse to its location on your file system.

Execute the following commands:

Ubuntu Upgrade

```
sudo su
service mission-control stop
tar -xvf jfmc-ubuntu-<version>.tar.gz
cd jfmc-ubuntu-<version>
./installJFMC-ubuntu.sh
```



Externalizing a Database?

Mission Control gives you the option to externalize one or more of its databases during an upgrade.

To externalize a database, when prompted with `Perform a standard Installation? [Y/n]:`, respond with "n" and then follow the prompts as described in [Using External Databases](#).

RedHat Upgrade

To upgrade Mission Control that is run as a RedHat installation, download the latest version and browse to its location on your file system.

Execute the following commands:

Redhat Upgrade

```
sudo su
service mission-control stop
tar -xvf jfmc-redhat-<version>.tar.gz
cd jfmc-redhat-<version>
./installJFMC-redhat.sh
```



Externalizing a Database?

Mission Control gives you the option to externalize one or more of its databases during an upgrade.

To externalize a database, when prompted with `Perform a standard Installation? [Y/n]:`, respond with "n" and then follow the prompts as described in [Using External Databases](#).

Debian Upgrade

To upgrade Mission Control that is run as a Debian installation, download the latest version and browse to its location on your file system.

Execute the following commands:

Debian Upgrade

```
sudo su
service mission-control stop
tar -xvf jfmc-debian-<version>.tar.gz
cd jfmc-debian-<version>
./installJFMC-debian.sh
```



Externalizing a Database?

Mission Control gives you the option to externalize one or more of its databases during an upgrade.

To externalize a database, when prompted with `Perform a standard Installation? [Y/n]:`, respond with "n" and then follow the prompts as described in [Using External Databases](#).

Migrating Your Data

Following an upgrade from a version 1.x to 2.0 and above, if you want to Mission Control to continue providing historical data you collected with version 1.x, you need to migrate your data. For details, please refer to [Migrating Data from InfluxDB to Elastic Search](#).



Running with Docker

Mission Control as a Docker Image

Mission Control can be installed as a Docker image and run as a container. To do this, you need to have Docker client properly installed and configured on your machine. For details about installing and using Docker, please refer to the [Docker documentation](#).

If you are running on a Windows or Mac, you need to install the Docker native client. Please note that we have only tested Docker installations on Linux and Mac.

Image Contents

The Mission Control Docker image contains the following components:

- Debian jessie 8
- OpenJDK
- Installation of Mission Control

Page Contents

- [Mission Control as a Docker Image](#)
 - [Image Contents](#)
- [License](#)
- [Download and Installation](#)
 - [Download](#)
 - [Installation](#)
- [Upgrading Mission Control](#)
- [Interacting with the Docker Installer](#)
- [Microservices](#)
- [Migrating data from 1.5.x to 2.0](#)
- [Accessing Mission Control](#)
 - [Default Admin User](#)

License

Mission Control itself is provided for free and has no license requirements. However, while you may view any Artifactory service from Mission Control, you may only configure and see relationships between services that are activated with an **Artifactory Enterprise license**.



Viewing Artifactory OSS, Artifactory Pro and Artifactory SaaS

You may add services of Artifactory OSS, Artifactory Pro to Mission Control, but for these instances, you can only view basic information. You cannot perform any actions on these services through Mission Control.

As a shared managed service, Artifactory SaaS is maintained for you by JFrog and can not be managed by Mission Control.

However, if you are running Artifactory SaaS as a **dedicated server**, this can be fully managed by Mission Control.

Download and Installation

Download

The JFrog Mission Control Docker installer can be downloaded from the [Mission Control Download Page](#).



Keep Mission Control on your \$PATH

Make sure to save the downloaded file in one of the locations defined in your \$PATH environment variable so it is accessible from anywhere on your machine.

Installation



Before you begin

Since Mission Control uses Elastic Search as its database for historical data, you need to set the mmap count to a larger value than default to avoid any memory leaks. Please refer to this [recommendation from Elastic Search](#).

To set the mmap count, run this command:

```
sysctl -w vm.max_map_count=262144
```




Docker Volume Mount

If requested at any time during the installation or upgrade process, make sure to provide the correct path to your Docker volume mount in the likely event that you're not using the default specified in the installation and upgrade scripts.

The JFrog Mission Control Docker image may be installed on any platform supporting Docker CE v17.x and above. To install Mission Control as a Docker image, follow the instructions below:

1. Make mission-control executable

To give the Mission Control installation script execute privileges on your machine, run:

```
chmod +x mission-control
```

2. Install Mission Control

The installation process will prompt you for a "root folder". You may keep the default (current) location or specify another location on your machine. Choose this location carefully since you may not change it later, and this is where JFrog Mission Control saves its data, configuration files and logs. The Mission Control installer will only prompt you for this location for initial installation. It is stored for later use when upgrading.

To install Mission Control, simply run:

```
./mission-control install
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using its own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

Note that for a Docker installation, Mission Control cannot seed an external MongoDB or Postgres database and requires you to **seed these databases manually**.

3. Start Mission Control

```
./mission-control start
```



Working in the Docker container

You can work in the Docker container using:

```
docker exec -it <container name> /bin/bash
```

Upgrading Mission Control

Upgrading Mission Control may vary slightly depending on your current version and the new version you are upgrading to.

- Download the latest installation script as described [above](#).
- Stop your current installation of Mission Control using the following command:

```
./mission-control stop
```

- Run the upgrade according to your version as follows:

- To upgrade from version 1.6 and above, run

```
./mission-control upgrade
./mission-control start
```

- To upgrade from version 1.5.x and below, you first need to upgrade to version 1.6 and then upgrade to version 2.x.

- To upgrade from version 1.5.x and below to version 1.6, first download the [JFrog Mission Control 1.6 installation script](#) and then run:

```
./mission-control install
./mission-control start
```

Note that this is not an error. The `mission-control` script's `install` function is used to upgrade from version 1.5.2 and below to version 1.6 and above to ensure that future upgrades work as intended.

- Then, to upgrade to version 2.x, run the latest installation script you downloaded as described [above](#) (Both installation scripts have the same name, so be careful not to confuse them):

```
./mission-control upgrade
```



Using External Databases

JFrog Mission Control uses several databases for different features of its operation. Until version 2.1, Mission Control installed an instance of all of these databases dedicated for its own use.

From version 2.1, Mission Control gives you the option of using your own **MongoDB**, **Postgres** or **Elasticsearch** databases if you have these already installed and in use in your organization.

When you run the installer, it will issue prompts asking if you want to install Mission Control using its own internal databases, or if you prefer to use your own external databases.

For details on how to respond to these prompts, please refer to [Using External Databases](#).

Note that for a Docker installation, Mission Control cannot seed an external MongoDB or Postgres database and requires you to **seed these databases manually**.

- Start Mission Control

```
./mission-control start
```



Upgrading from older versions or ZIP installation

If you are upgrading from an older version that was installed without the installation script, or you previously installed Mission Control as a [standalone ZIP installation](#), you may be prompted for the "root folder". Make sure to specify the same `MC_HOME` folder your current installation is using.

Interacting with the Docker Installer

In addition to managing installation, the `mission-control` installation script can provide additional information or perform additional tasks on your installation such as restarting Mission Control, displaying log files and more. For details, run:

```
./mission-control help
```

Microservices

Mission Control 2.0 is composed of several microservices that manage different aspects of the product:

server	jfmc_server_1	Primary Mission Control service
--------	---------------	---------------------------------

core	jfmc_core_1	Handles core functions of for managing graph data
scheduler	jfmc_scheduler_1	Manages scheduling for different internal Mission Control tasks
executor	jfmc_executor_1	Executes tasks related to data management
elasticsearch	jfmc_elasticsearch_1	The database used for all time-series data used to generate graphs
postgres	jfmc_postgress_1	Database used by the scheduler
mongodb	jfmc_mongodb_1	Database used by the core
influxdb	jfmc_influxdb_1	Database used by Mission Control

Migrating data from 1.5.x to 2.0

Mission Control 1.5.x and above used InfluxDB to store the historical data for graphs. Starting Mission Control 2.0, [Elasticsearch](#) is the store for this data. To migrate the existing data in InfluxDB to Elasticsearch, please refer to this [document](#).

Accessing Mission Control

Mission Control can be accessed at the following URL:

`http://localhost:8080`

Default Admin User

Once installation is complete, Mission Control has a default user with admin privileges predefined in the system:

User: admin

Password: password



Change the admin password

We strongly recommend changing the admin password as soon as installation is complete.

Configuring Mission Control

Overview

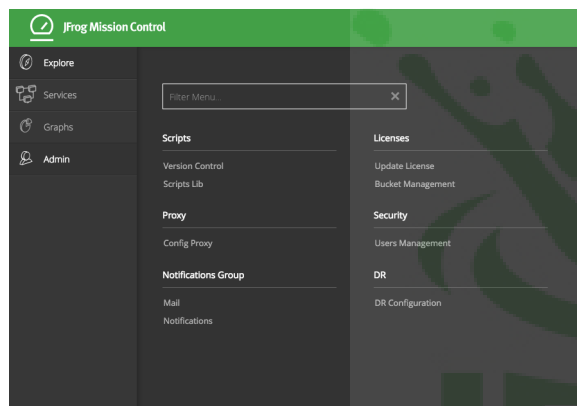
The Mission Control **Admin** module provides access to a variety of configuration screens as well as scripting and other functionality as follows:

Scripts	Create and modify service configuration scripts and configure how they are managed in version control.
Proxy	Define and configure proxies through which Mission Control will access the services under its control.
Security	Add and modify users and their privileges in Mission Control.
Notifications Group	Define usage policies and thresholds that specify when notifications will be sent and configure a mail server through which to send them.
Licenses	Manage license buckets or individual licenses for specific services.
DR	Configure Master and Target pairs for disaster recovery.

Page Contents

- [Overview](#)
- [Scripts](#)
 - [Version Control](#)
- [Configuring Proxies](#)
- [Configuring Users](#)
- [Mail](#)
- [Notifications](#)
- [Licenses](#)
- [Disaster Recovery](#)

Read More



Scripts

Scripts are an easy way to define standard, repeatable and automated configuration for the different managed services.

To access your library of scripts, under the **Admin** module select **Scripts | Scripts Lib**.

For full details on how to work with scripts, please refer to [Configuration Scripts](#).

Version Control

Mission Control supports versioning for configuration scripts in a Git repository. To configure a Git repository for configuration script versions, in the **Admin** module select **Scripts | Version Control**.

For details, please refer to [Git Integration](#).

Configuring Proxies

For each managed service, you can configure a proxy through which Mission Control will access it.

To view the list of proxies configured, in the **Admin** module, select **Proxy | Config proxy**.

Proxies

⊕ Create Proxy

3 Proxies

Filter by Name

< Page 1 of 1 >

Name

proxy-west

proxy-east



proxy-bangalore

To edit or delete a proxy, hover over the proxy's row in the list and click the corresponding icon.

To create a new proxy, click **Create Proxy**.

Create Proxy

Proxy Settings

Name *

URL *

Credentials

User Name

Password

Name	A logical name for this proxy
URL	The proxy URL
User Name	A user name required to access the proxy server (optional).
Password	The password required to access the proxy server (optional).

Configuring Users

To manage users in Mission Control, in the **Admin** module select **Security | User Management**.



Mission Control users vs. Artifactory users

Be careful not to confuse this feature with managing users in the services managed by Mission Control.

Users

3 Users

Filter by Name or Email

[+ Create User](#)

< Page 1 of 1 >

Name	Email	
admin	admin@email.com	
johns	johns@email.com	✎ ✕
janes	janes@email.com	

To edit or remove an existing user, click the corresponding link in the user's row in the table.

To create a new Mission Control user, click "Create User".

Edit User

User Settings

User Name *

johns

Email Address *

johns@email.com

Password

Password Confirmation

Fill in the fields and click "Submit"



MC password

The Mission Control password must be at least 8 characters long and contain both letters and numbers.

Mail

Mission Control sends email notifications to users for different events that occur; for example, notifications that are generated due to storage policies that are defined for instances or repositories.

To enable mail notifications, you need to configure Mission Control with your mail server details in the Admin module under **Notifications Group | Mail**.

JFrog Mission Control

Add Service
Execute Script
Welcome, admin (Log Out)
Help

Mail

Mail Setup

☐ Enable

Host *

Port *

User Name

Password

From *

Subject prefix

☐ Use SSL/TLS

Enable	Enables the configured mail server
Host	The host name of the mail server.
Port	The port of the mail server.
Username	The username for authentication with the mail server.
Password	The password for authentication with the mail server.
From	The "from" address header to use in all outgoing mails.
Subject Prefix	A prefix to use for the subject of all outgoing mails.
Use SSL/TLS	When set, uses Transport Layer Security when connecting to the mail server.
Send Test Mail	Click to send a test message to the specified email address.

Notifications

Mission control can send email notifications based on policies you define for instances and repositories. For example, you can configure Mission Control to send an email message if any Artifactory instance exceeds a limit for usage of storage. For details, please refer to [Notifications](#).

Licenses

The **Admin** module provides access to the Update Licenses and License Bucket screens where you can manage licenses for specific services or license buckets for a group of services. For details, please refer to [Licenses](#) under [Managing Services](#).

Disaster Recovery

The **Admin** module provides access to the Disaster Recovery screen where you can configure Master and Target pairs of Artifactory services. For details, please refer to [Disaster Recovery](#).

Configuration Scripts

Overview

Mission control embraces the configuration-as-code approach and uses scripts to configure the services that it manages. Scripts are reusable pieces of code which can be applied to one or more services at a time to perform a variety of actions. These can range from simple configurations, such as setting a new caching policy for a set of remote repositories in an Artifactory service, to more complex ones like creating a combination of local and remote repositories in a master-slave topology, or even to set up watches in a managed Xray service.

Here is a simple example of a configuration script that creates a local repository that will be a Docker registry in a managed Artifactory service called "Art1":

```
artifactory('Art1'){
  localRepository("docker-local") {
    packageType "docker"
    description "My local Docker registry"
  }
}
```

Using configuration scripts presents several benefits for the management of services under Mission Control

- **Automation:** Configuration scripts improve your efficiency by enabling you to automate your service configuration tasks, preventing the need to perform repetitive and error prone manual configuration, especially when managing multiple services.
- **Reliability:** Configuration scripts improve the reliability of your configuration tasks by letting you reuse the same configuration on multiple services that may also be running on different runtime environments (e.g. development, staging, production).
- **Standardization:** Configuration scripts can be used to enforce standards when configuring things such as repository names, include/exclude patterns, caching policies and more.



Scripting in version 2.x

In version 2.0, scripting in Mission Control underwent significant changes. Any scripts written for Mission Control v1.x will not work and need to be migrated. For details, please refer to [Migrating Scripts from Version 1.x to Version 2.x](#).

Page Contents

- [Overview](#)
- [Working with Scripts](#)
 - [Script Library](#)
 - [Running a Script](#)
 - [Selecting a Script to Run](#)
 - [Editing a Script](#)
 - [Entering User Input](#)
 - [Doing a Dry Run](#)
 - [Executing a Script](#)
- [Creating and Editing Scripts](#)
 - [Using the Script Editor](#)
 - [Block Templates](#)
 - [Using an External Editor](#)
- [Script Elements](#)
 - [Service Closures](#)
 - [Configuration Blocks](#)
 - [Properties](#)
 - [Example](#)
- [User Input](#)
 - [Input Types and Properties](#)
 - [Example](#)
- [Global Variables](#)
- [Running Scripts via REST API](#)
- [Migrating Scripts from Version 1.x to Version 2.x](#)
 - [Best Practices](#)
 - [Aggregating Scripts](#)
 - [Dry Run](#)
- [Best Practices](#)

Read More

- [Configuration DSL](#)
- [Git Integration](#)

Working with Scripts

Scripts are written using the [Groovy programming language](#) and are managed under version control in a Git repository. This allows you to edit scripts directly in your Git provider's editor, or using any other editor you prefer to work with. Once your scripts are committed to your Git repository, Mission Control accesses them using the configuration you specify under [Git Integration](#) and is automatically synchronized with any additions or deletions you make from the Git repository.



Create vs. Update

Mission Control scripts are written in the same way whether they create new entities or just update them. If the relevant entity (service or repository) already exists on the target service, then it will just be updated with the parameters specified in the script. If the entity does not exist, it will be created with the parameters specified. Any optional parameters not specified will take default values.

For example, consider this simple script :

```
artifactory('Art1'){
  localRepository("maven-local-dev") {
    packageType "maven"
    description "This is my Maven repository for development"
  }
}
```

If the "Art1" Artifactory service already has a Maven repository called "maven-local-dev", its description will be updated to the value provided. If the repository does not exist, it will be created with the description provided and all other parameters set to their default values.

Note that when doing a [dry run](#) on a script, Mission Control will show the changes that the script will implement in the case of an update.

Script Library

Scripts are managed in the Script Library which can be accessed from the **Admin** module under **Scripts | Script Lib**. Hovering over an item in the library displays icons that let you **delete**, **edit** or **run** the script. Scripts are maintained in a MongoDB database which can only be accessed by Mission Control. If you have configured [Git Integration](#), all scripts will be synchronized with the Git repository specified.

The screenshot shows the JFrog Mission Control interface. The top navigation bar is green and contains the JFrog logo, 'JFrog Mission Control', a search icon, 'Add Service', 'Run Script', 'Welcome, admin (Log Out)', and 'Help'. The left sidebar is dark grey with icons for Home, Scripts, Settings, and Users. The main content area is titled 'Scripts Library' and features a '+ Add New Script' button. Below this is a section for '3 Scripts' with a search filter 'Filter by Name or Description'. A table lists the scripts:

Name	Description	
push-replication	Specifies a push replication relationship	
new-docker-local	Creates a new local Docker registry	
new-maven-remote	Creates a new Maven remote repo letting you specify the remote proxied resource	

At the bottom right of the table, there are icons for delete, edit, and run. The page number 'Page 1 of 1' is displayed at the bottom right of the script list.

Running a Script

Running a script involves the following steps:

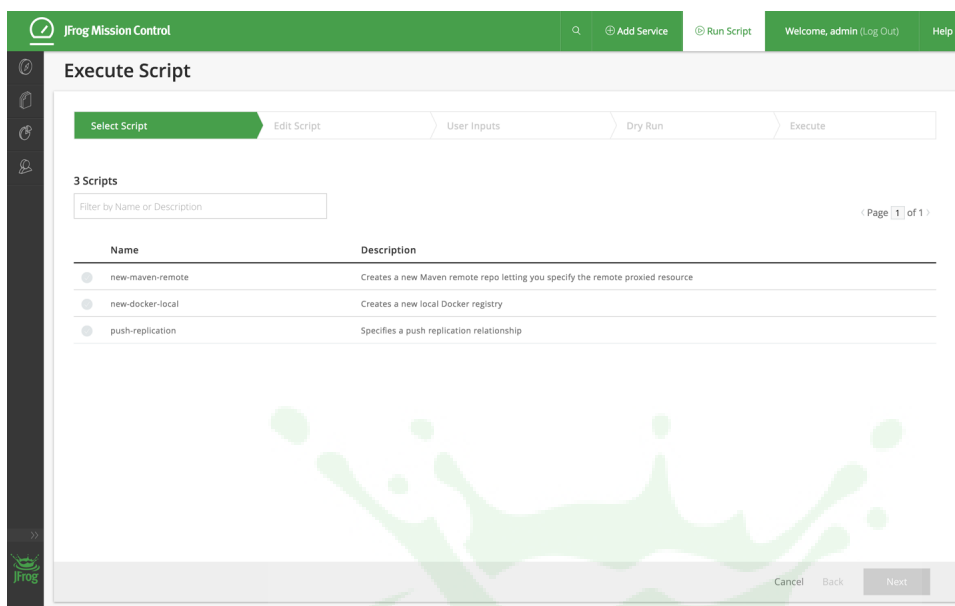
- **Selection** - select the script you want to run out of the list
- **Editing** - make any edits needed to the script
- **Providing input** - provide any input required by the script
- **Dry run** - do a dry run of the script to ensure there are no errors
- **Execute** - execute the script

These main steps are described in more detail in the sections below.

Selecting a Script to Run

There are two ways to select a script to run:

1. Selecting **Run Script** from the Mission Control top ribbon. This will present the list of scripts available in your script library



Select the script you want to run and click "Next".

2. Hovering over the script in the Script Library and clicking the **Run** icon.

Both of these actions will take you to the next step of editing the script.

Editing a Script

The **Edit Script** tab lets you make any changes necessary to the script to meet the specific needs of the current run.



Tweak, don't make big changes

This feature was designed to let you make minor changes that you may need to make to accommodate slightly different scenarios to which you would apply a script. To make significant changes to a script, we recommend modifying the script using an external editor and committing it to your Git repository.

JFrog Mission Control

Add Service
Run Script
Welcome, admin (Log Out)
Help

Execute Script

Select Script Edit Script User Inputs Dry Run Execute

Name * Description

new-maven-remote Creates a new Maven remote repo letting you s

Configuration Script Editor *

Use "Ctrl-Space" or "Shift-Space" for auto-completion

```

1  repoName = userInput (
2    type : "STRING",
3    description : "Please provide a repository name"
4  )
5
6  artifactory('Art1') {
7    remoteRepository(repoName) {
8      description : "This is a description"
9      notes "Some internal notes"
10     includesPattern "**/*" // default
11     excludesPattern "" // default
12     repoLayoutRef "maven-2-default"
13     propertySets // (["ps1", "ps2"])
14     archiveBrowsingEnabled false

```

Save

Cancel Back Next

Note that you cannot modify the name or description of an existing script.

Click "Save" to save your changes.



Saving changes commits them to Git

Note that if you have a [Git repository](#) defined for your scripts, saving your changes will commit them to Git.



Syntax Errors

If there are any syntax errors in your script, Mission Control will display an alert. You need to fix the error in order to proceed.

Click "Next" to move on to the next step of adding user input.

Entering User Input

This is the step in which you provide any user input required by the script you are running.

JFrog Mission Control

QAdd ServiceRun ScriptWelcome, admin (Log Out)Help

Execute Script

Select Script

Edit Script

User Inputs

Dry Run

Execute

serviceName

Please provide an Artifactory service name

repoName

Please provide a Maven repository name

Cancel

Back

Next

Enter the user input required and click "Next" to move on to the next step of testing your script in a dry run.

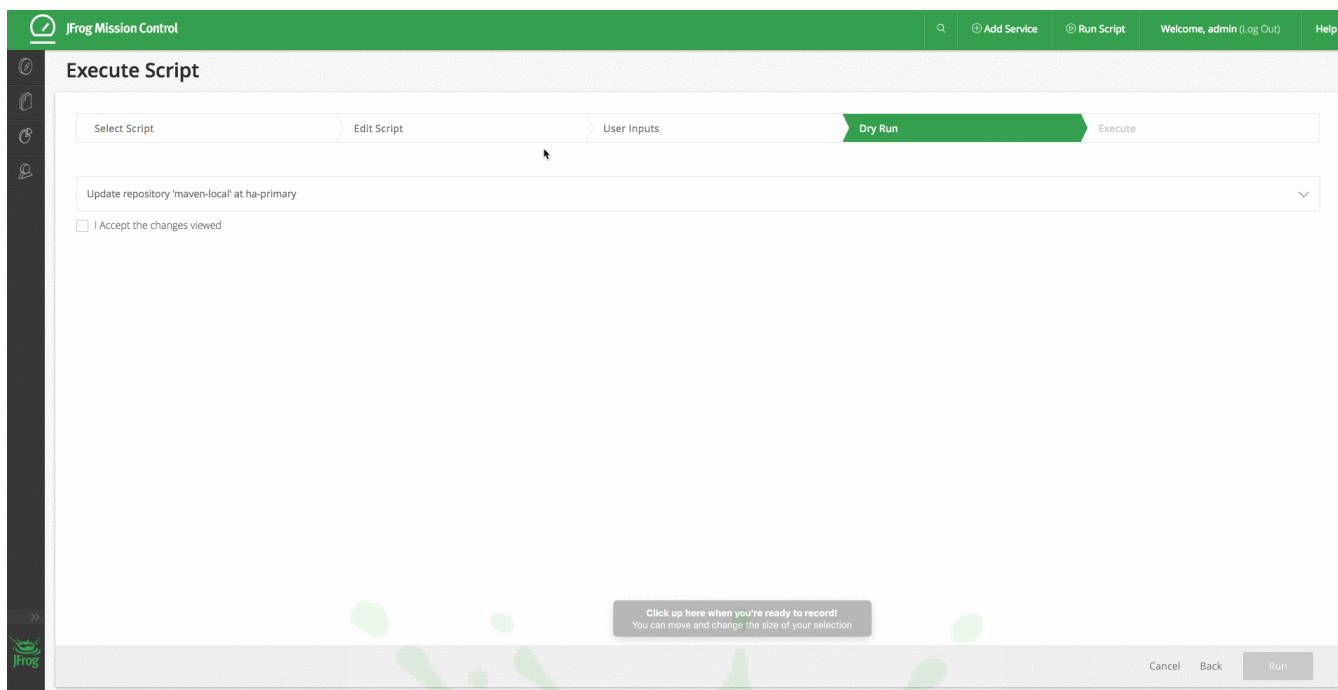


If there are invalid instructions in your script, Mission Control will display an error. You need to fix the error in order to proceed.

For example, if you have written a script that updates an Artifactory service called "Art1", but there is no such service recognized by Mission Control, then this is an error.

Doing a Dry Run

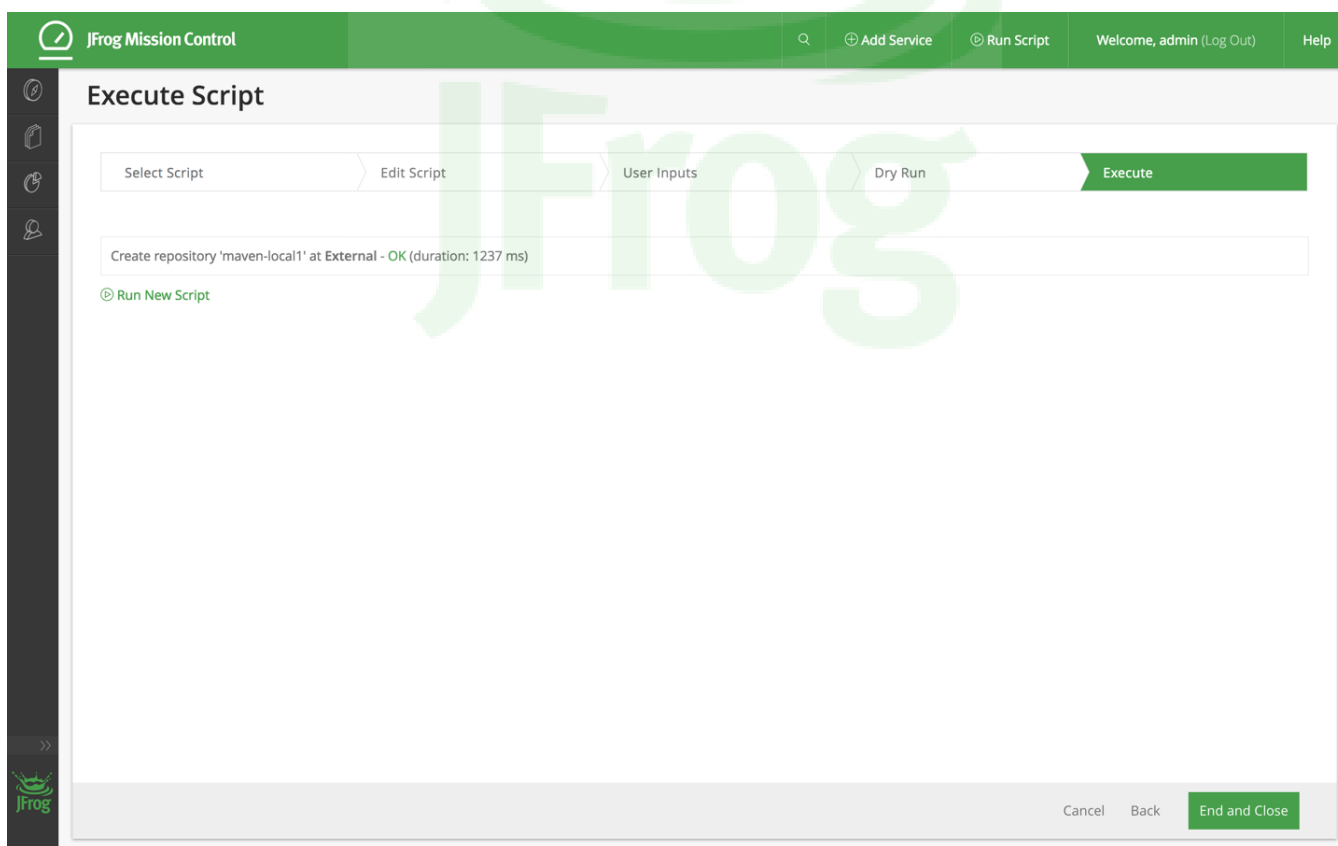
The dry run does not execute the script, but let's you know what changes will be implemented on the selected services and repositories when you do. To see the changes, click the the summary line.



To do a dry run, accept the changes your script will make and click "Run".

Executing a Script

If the dry run is successful, you can click **Run New Script** to actually execute the code in the script. Take care, and note that this time, any changes programmed into your script will actually be executed on the corresponding services and repositories.



Upon successful execution, Mission Control will again display the list of scripts available for you to run.

At any step along the way, you can stop the process of running a script by clicking "Cancel", or by clicking "End and Close" in the Execute step.

Creating and Editing Scripts

As described above, scripts are written in Groovy.

There are two ways to create and edit scripts:

- Using an external editor
- [Using the Script Editor](#)



Use and external editor

As a best practice, we recommend creating scripts outside of Mission Control using your preferred editor and then committing them to the Git repository configured in the [Version Control](#) page.

Using the Script Editor

To create a script from within Mission Control, select **Add New Script** in the Scripts Library. To edit the script select the "Edit" icon while hovering over the script's entry in the [Script Library](#).



When you have a Git repository configured in the [Version Control](#) page, Mission Control will commit any new scripts you create in the script editor to the Git repository and commit new versions when you edit the scripts.

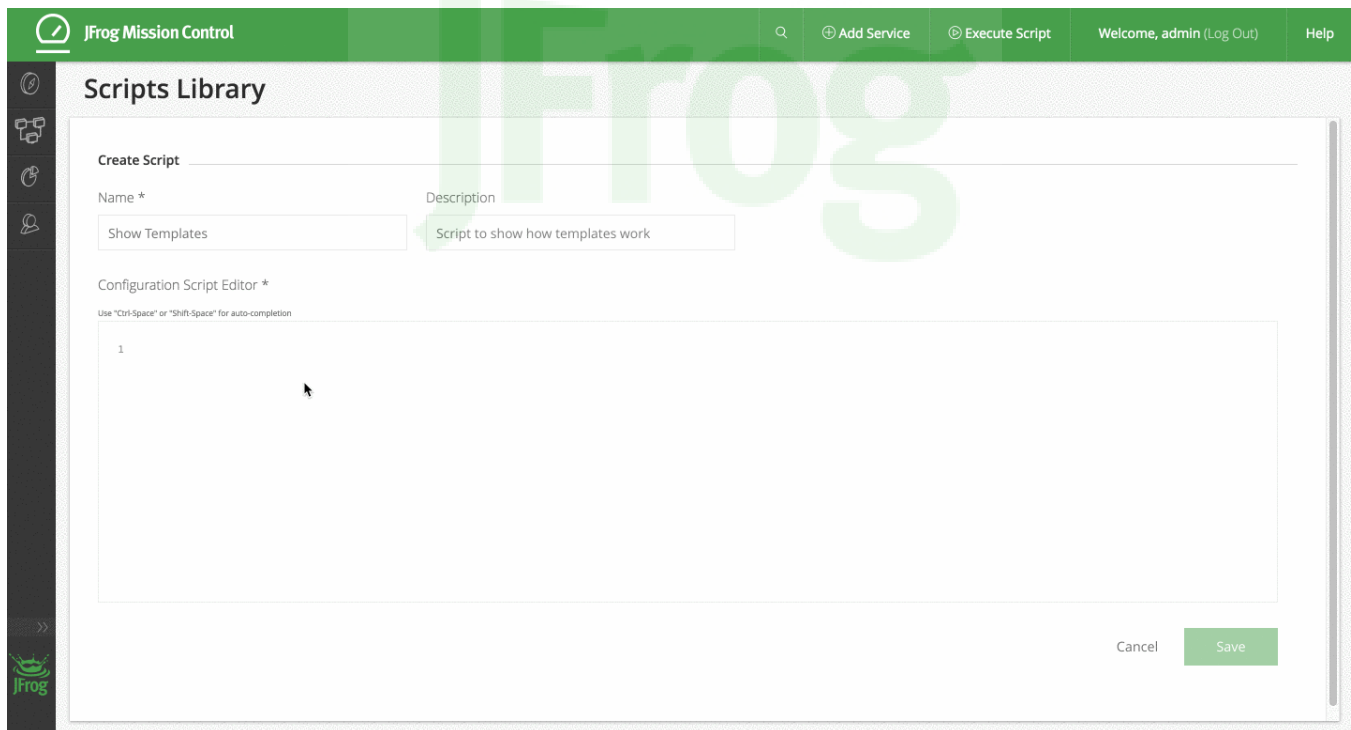
Once in the script editor you can create and/or edit the script as needed. When done, click "Save" to save the script. If a Git repository is configured, Mission Control will commit your changes.

Block Templates

As a convenience, Mission Control offers configuration blocks as built-in templates that you can use when creating or editing scripts. The templates available are exposed by typing the first few letters and then **ctrl + space** or **shift + space** to show the auto-complete options. When you select a template from the list, Mission Control will insert it into your script with all the parameters of the selected configuration block.

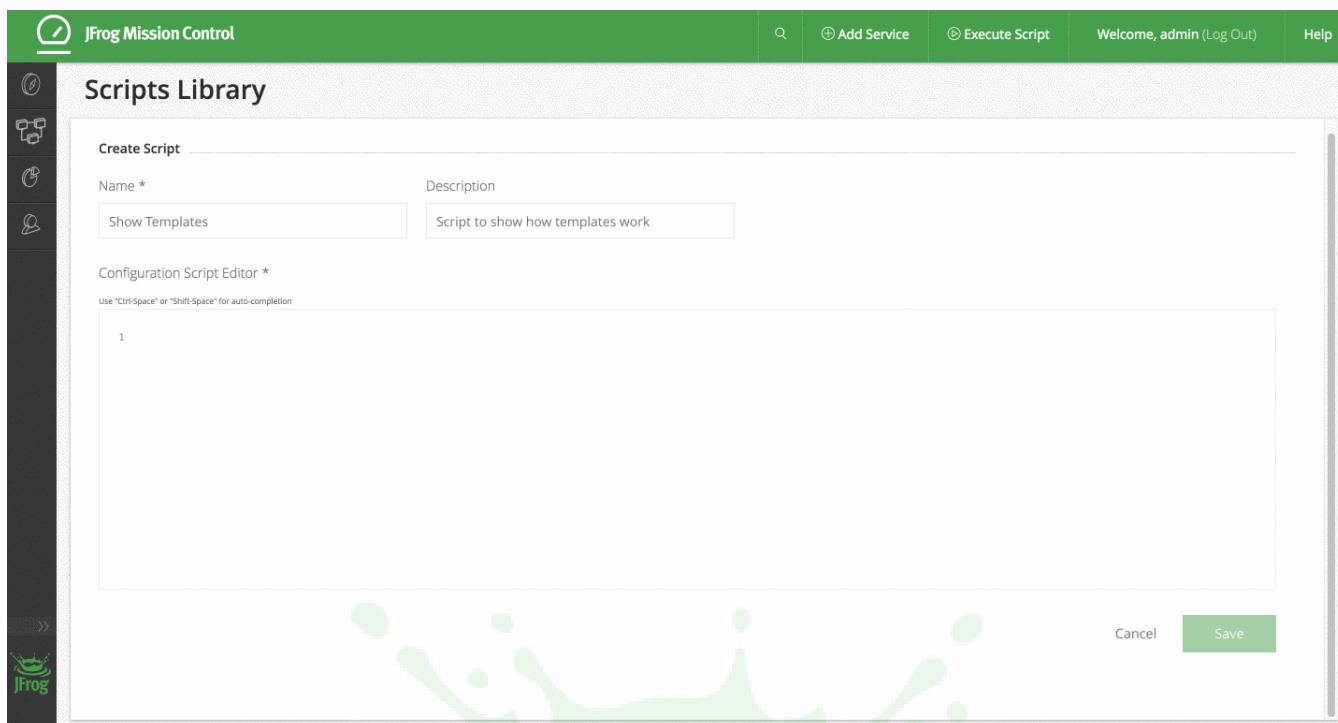
Example 1

The example below shows how to insert an Artifactory service and Local repository template:



Example 2

The example below shows how to add an Xray service template:



Using an External Editor

You can create or edit scripts using any external editor. Just make sure to commit any changes to the Git repository configured in the [Version Control](#) page.

Script Elements

Scripts are constructed from Service Closures, Configuration Blocks and Properties. Scripts are very flexible, and can contain any number of service closures, each of which may contain any number of configuration blocks and properties.

For a complete list and full specification of service closures, configuration blocks and properties available, please refer to [Configuration DSL](#).

Service Closures

Service closures define the service (Artifactory or Xray) that the script acts on. Artifactory or Xray services can only be created or modified through configuration blocks which must be enclosed in service closures.

Configuration Blocks

Configuration blocks define the parameters or changes that should be implemented on the services specified in their enclosing service closures.



Configuration blocks must be in service closures

A configuration block must be placed inside a service closure that specifies the service on which it should be applied.

Properties

Properties are used to configure the relevant parameters of a configuration block.

Example

The simple example script below shows:

- An Artifactory closure that includes a configuration block that creates a generic local repository called "generic-local" on an Artifactory service called "Art1". The configuration block includes a property that specifies the package type for the repository
- An Xray closure that [connects an Xray service](#) called "X1" to the Artifactory service "Art1" for indexing.

Example script

```
artifactory('Art1') {  
  localRepository('generic-local') {  
    packageType "generic"  
  }  
}  
  
xray('X1') {  
  binaryManager('Art1')  
}
```

User Input

Static configuration scripts can be useful in some cases. For example, when you would like to create a property set with pre-defined properties. In other cases you need the script to be more dynamic. For example, when creating a new repository you might want to provide the repository name only when the script is applied.

To create more dynamic configuration scripts, the Mission Control configuration DSL lets you ask for user input when the script is applied.

There are two ways of declaring that you would like to get user input:

1. Asking for user input for a specific property

```
localRepository("my-repository") {  
  description userInput (   
    type : "STRING",  
    value : "This is a generic description",  
    description : "Please provide a description"  
  )  
}
```

In this example, we ask the user to provide a value for the description property of a repository.

2. Assigning the user input to a variable and using the variable

```
name = userInput (   
  type : "STRING",  
  value : "This is a generic description",  
  description : "Please provide a repository name"  
)  
  
localRepository(name) {  
  ...  
}
```

In this example, the value of the name variable is used in a Groovy string to create the repository name.



Don't use "def"

Take care not to use "def" when declaring user input for a script (e.g. `def name = userInput...`). When using def, the script will not work correctly as it will refer to the user input object rather than the dynamic value entered by the user

When a configuration script with user input is applied, Mission Control will generate a form that prompts you for all user input fields defined. The user applying the script will need to provide the input fields in order to proceed with a dry run and execution of the script.

JFrog Mission Control

Add Service
Run Script
Welcome, admin (Log Out)
Help

Execute Script

Select Script Edit Script **User Inputs** Dry Run Execute

serviceName

Please provide an Artifactory service name

repoName

Please provide a Maven repository name

Cancel Back Next

Input Types and Properties

When requesting user input in a script, you need to specify the following parameters:

type	<p>This can take one of the following values:</p> <p>STRING - the input is a simple string</p> <p>BOOLEAN - the input is a simple boolean</p> <p>INTEGER - the input is a simple integer</p> <p>SERVICE - the input is one of the services managed by Mission Control. The user input screen will display a list of services for the user to select from.</p> <p>ARTIFACTORY - the input is an Artifactory service managed by Mission Control. The user input screen will display a list of Artifactory services for the user to select from.</p> <p>REPOSITORY - the input is a repository in an Artifactory service managed by Mission Control. The user input screen will display a list of Artifactory services for the user to select from.</p> <p>XRAY - the input is an Xray service managed by Mission Control. The user input screen will display a list of Xray services for the user to select from.</p> <p>PACKAGE_TYPE - The input is one of the package types supported by Artifactory (e.g. "docker", "npm" "debian"). For a full list of supported package types, please refer to Repository Configuration JSON in the Artifactory User Guide.</p>
value (Optional)	A default value. If a default value is not specified, the variable becomes mandatory and the user must provide the input string.
multivalued (Optional)	When true, mission control will allow the user to provide more than one value. The following types can take multiple values: SERVICE, ARTIFACTORY, XRAY, REPOSITORY
description (Optional)	A description to be displayed in the web UI

According to the input type requested, once entered, the script has access to different properties related to the input value as described in the table below:

Input Type	Property Name	Property Type	Description
SERVICE, ARTIFACTORY or XRAY	name	String	Service name in mission control
	url	String	Service URL
	type	EntityType (enum)	Service type (ARTIFACTORY, XRAY)
	description	String	Service description
	credentials	Credentials	Service credentials, ex: credentials.userName, credentials.password
REPOSITORY	url	String	Service URL
	repository	LocalRepositoryImpl, RemoteRepositoryImpl, VirtualRepositoryImpl	Repository properties as described in Repository Configuration JSON

Example

The following script requests a target Artifactory service as user input to create a replication relationship with another Artifactory service called "LocalK".

Once the target instance has been provided, the script uses its **url** property to specify its **maven-local** repository as the replication target.

```
targetArtifactory = userInput (
  name : "Target Artifactory",
  type : "ARTIFACTORY",
  description : "please provide the artifactory instance you want to replicate to"
)

artifactory('Local') {
  localRepository("maven-local") {
    replication {
      url "${targetArtifactory.url}/maven-local"
      username targetArtifactory.credentials.userName
      password targetArtifactory.credentials.password
      cronExp "0 0/9 14 * * ?"
      socketTimeoutMillis 15000
    }
  }
}
```

Global Variables

In addition to the objects and properties available to scripts as a result of user input, scripts also have access to a global variable called **services**.

The **services** variable is a map of all services being managed by Mission Control and provides access to the service object using its **name** property as the key.

For example, if Mission Control is managing an Artifactory service called "Art1" and an Xray service called "X1", a script can access these service objects using the following lines:

```
def serviceArtifactory = services['Art1']
def serviceXray = services['X1']
```

Once the respective service objects are acquired, the script can then reference the different properties such as name, url, description etc. as described in the table above.

Running Scripts via REST API

The REST API for interacting with and running scripts has changed in Mission Control 2.0. For details, please refer to [SCRIPTS](#) in the Mission Control REST API documentation.

Migrating Scripts from Version 1.x to Version 2.x



Following an upgrade of Mission Control from version 1.x to version 2.x, all scripts created with version 1.x should be available in the [Script Library](#) of version 2.x.

In version 2.0 JFrog Mission Control scripting functionality underwent several changes. The most significant change is in how you select which services to apply the script to.

In version 1.x, you could create scripts that operated on Artifactory instances and repositories, but would only have to select those instances and repositories when actually running the script.

From version 2.0 any operations on services (whether Artifactory or Xray) must be enclosed in a [service closure](#) (as described above) that specifies the service on which the script should be applied. While the specific service may also be entered with user input, the enclosing closure must be present in the script. As a result, you need to migrate all scripts written for version 1.x and **enclose the content of the script in a service closure**. The following example shows how a simple script written for version 1.x would be migrated to be compatible with version 2.x

Example

Create a local Docker registry called "docker-local" in an Artifactory service.

In version 1.x, the Artifactory service would be selected during the flow of running the script.

In version 2.x we add an artifactory closure

Script in version 1.x	Script migrated for version 2.x
<pre>localRepository("docker-local") { packageType "docker" description "My local Docker registry" }</pre>	<p>Option 1: Specify a specific Artifactory service called "Art1"</p> <pre>artifactory('Art1'){ localRepository("docker-local") { packageType "docker" description "My local Docker registry" } }</pre> <p>Option 2: Let the user select the Artifactory service with user input:</p> <pre>whichArtifactory = userInput (type : "ARTIFACTORY", name : "Which Artifactory", description : "Please specify the Artifactory service on which to create the repository") artifactory(whichArtifactory.name){ localRepository("docker-local") { packageType "docker" description "My local Docker registry" } }</pre>

Best Practices

Here are some best practices that we recommend you follow when migrating your scripts to Mission Control 2.0 and above:

Aggregating Scripts

In version 1.x, Mission Control scripts could only perform one action, and were limited to acting either on an Artifactory instance, or on a repository. From version 2.0 scripting is much more flexible and scripts can be written to perform any number of actions. We recommend taking a series of small, single-action scripts and aggregating them into a larger script that performs several functions.

Dry Run

To make sure you have migrated your scripts correctly, we recommend doing a dry run and verifying the changes that Mission Control would make if you were to actually run the script.

Best Practices

To learn more about configuration scripts by example, check out [JFrog's public GitHub repository](#) of scripts for Mission Control.

These scripts provide best practices for writing scripts related to:

- Creating a single generic repository
- Onboarding a team of users
- Setting up replication relationships including implementation of a Star Topology and Mesh Topology

We recommend watching this repository for updates with more script examples.



Configuration DSL

Overview

Mission Control comes with a comprehensive set of built-in configuration blocks that are intended as guides to make it easier for you to define configuration scripts using the allowed DSL.



Configuration blocks must be in service closures

Note that the script segments described on this page represent configuration blocks and cannot be used by themselves. Configuration blocks **must** be placed within a **service closure** in order to create an executable script.

Artifactory Configuration Blocks

This section presents configuration blocks that can be used to configure different administrative features of Artifactory services. As with any configuration block, these must be placed within an Artifactory **service closure** as shown below.

```
Artifactory service closure

artifactory('<Artifactory service name>'){

    <configuration blocks>

}
```

Page Contents

- [Overview](#)
- [Artifactory Configuration Blocks](#)
 - [Property Sets](#)
 - [Repository Layout](#)
 - [Proxies](#)
 - [LDAP Settings](#)
 - [LDAP Groups](#)
 - [Security Settings](#)
- [Repository Configuration Blocks](#)
 - [Local Repository](#)
 - [Remote Repository](#)
 - [Virtual Repository](#)
 - [Replication](#)
 - [Star Topology](#)
 - [Push Replication](#)
 - [Pull Replication](#)
- [Xray Configuration Blocks](#)
 - [Link to Artifactory - Binary Manager](#)
 - [Watches](#)

Property Sets

The parameters for a `propertySets` configuration block are described below. For more details on these parameters, please refer to [Property Sets](#) in the [Artifactory User Guide](#).

```
propertySets block

propertySets {
  propertySet('property_set_name') {
    singleSelect('property_name') {
      defaultValue "value_1"
      value "value_1"
      value "value_2"
    }
    multiSelect('property_name_multi') {
      defaultValue "value_1"
      defaultValue "value_2"
      value "value_1"
      value "value_2"
    }
    anyValue('another_property_name') {
      defaultValue "value_1"
      value "value_1"
    }
  }
}
```

property_set_name	An identifier for this property set. The name must be unique in all Artifactory instances on which it is applied.
singleSelect parameters	
property_name	A unique identifier for the single selection property within this property set.
defaultValue	A default value for the single selection property.
value	The selection options for this property

multiSelect parameters	
property_name_multi	A unique identifier for the multiple selection property within this property set.
defaultValue	The default selected values for the multiple selection property.
value	The selection options for this property
anyValue parameters	
another_property_name	A unique identifier for the free-text property within this property set.
defaultValue	A default value for the free-text property.
value	The selection options for this property

Repository Layout

The parameters for a `repoLayout` configuration block are described below. For more details on these parameters, please refer to [Repository Layouts](#) in the [Artifactory User Guide](#).

repoLayout block	
<pre>repoLayouts { repoLayout ('repo_layout_name') { folderIntegrationRevisionRegExp "SNAPSHOT" fileIntegrationRevisionRegExp "SNAPSHOT (?:(?:[0-9]{8}.[0-9]{6})-(?:[0-9]{+}))" distinctiveDescriptorPathPattern true artifactPathPattern "[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).[ext]" descriptorPathPattern "[orgPath]/[module]/[baseRev](-[folderItegRev])/[module]-[baseRev](-[fileItegRev])(-[classifier]).pom" } }</pre>	

repo_layout_name	An identifier for this repository layout. The name must be unique in all Artifactory instances on which it is applied.
folderIntegrationRevisionRegExp	A regular expression for folder integration revision.
fileIntegrationRevisionRegExp	A regular expression for file integration revision.
distinctiveDescriptorPathPattern	A distinctive descriptor path pattern is used to recognize descriptor files.
artifactPathPattern	The typical structure in which all module artifacts are expected to be stored.
descriptorPathPattern	The pattern used to recognize descriptor files (such as <code>.pom</code> or <code>ivy.xml</code> files).

Proxies

The parameters for a `proxies` configuration block are described below. For more details on these parameters, please refer to [Managing Proxies](#) in the [Artifactory User Guide](#).

<pre>proxies { proxy('proxy_key') { host 'proxy host' // mandatory port 8888 // mandatory username 'username' password 'password' defaultProxy false ntHost 'NT Host' ntDomain 'NT Domain' redirectedToHosts(['host1', 'host2', 'host3']) } }</pre>	
---	--

proxy_key	The ID of the proxy. Must be unique within the Artifactory instance.
host	The name of the proxy host.
port	The proxy port number
username	The proxy username when authentication credentials are required.
password	The proxy password when authentication credentials are required.
defaultProxy	When true, this proxy will be the default proxy for new remote repositories and for internal HTTP requests.
ntHost	The computer name of the machine (the machine connecting to the NTLM proxy).
ntDomain	The proxy domain/realm name.
redirectedToHosts	An optional list of newline or comma separated host names to which this proxy may redirect requests.

LDAP Settings

The parameters for an `ldap` configuration block are described below. For more details on these parameters, please refer to [Managing Security with LDAP](#) in the [Artifactory User Guide](#).

```
ldap {
  settings('settings_name') {
    url 'ldap://myserver:myport/DC=sampldomain,DC=com' // mandatory
    userDnPattern 'uid={0},ou=People'
    emailAttribute 'mail'
    enabled true // default value - true
    autoCreateUser true // default value - true
    search {
      filter '(uid={0})'
      base 'OU=dev,DC=sampldomain,DC=com'
      searchSubTree true
      managerDn 'CN=admin,OU=ops,DC=sampldomain,DC=com'
      managerPassword 'password'
    }
  }
}
```

settings_name	The ID of the LDAP setting. Must be unique within the Artifactory instance being configured.
url	Location of the LDAP server in the following format: <code>ldap://myserver:myport/dc=sampldomain,dc=com</code> .
userDnPattern	A DN pattern used to log users directly in to the LDAP database. This pattern is used to create a DN string for "direct" user authentication, and is relative to the base DN in the LDAP URL.
emailAttribute	An attribute that can be used to map a user's email to a user created automatically by Artifactory.
enabled	When true, these settings are enabled.
autoCreateUser	When true, Artifactory will automatically create new users for those who have logged in using LDAP, and assign them to the default groups.
filter	A filter expression used to search for the user DN that is used in LDAP authentication.
base	The Context name in which to search relative to the base DN in the LDAP URL. This is parameter is optional.
searchSubTree	When true, enables deep search through the sub-tree of the LDAP URL + Search Base.
managerDn	The full DN of a user with permissions that allow querying the LDAP server.
managerPassword	The password of the user binding to the LDAP server when using "search" authentication.

LDAP Groups

The parameters for an `ldap` groups configuration block are described below. For more details on these parameters, please refer to [LDAP Groups](#) in the [Artifactory User Guide](#).

```
ldap {
  groupSettings('static_group_settings_name') {
    settings // 'ldap settings ref'
    staticMapping {
      groupMemberAttribute 'uniqueMember' // mandatory
      groupNameAttribute 'cn' // mandatory
      descriptionAttribute 'description' // mandatory
      filter '(objectClass=groupOfNames)' // mandatory
      searchBase ''
      searchSubTree true
    }
  }
  groupSettings('dynamic_group_settings_name') {
    settings // 'ldap settings ref'
    dynamicMapping {
      groupMemberAttribute 'uniqueMember' // mandatory
      groupNameAttribute 'cn' // mandatory
      descriptionAttribute 'description' // mandatory
      filter '(objectClass=groupOfNames)' // mandatory
      searchBase ''
      searchSubTree true
    }
  }
  groupSettings('hierarchy_group_settings_name') {
    settings // 'ldap settings ref'
    hierarchyMapping {
      userDnGroupKey 'uniqueMember' // mandatory
      groupNameAttribute 'cn' // mandatory
      descriptionAttribute 'description' // mandatory
      filter '(objectClass=groupOfNames)' // mandatory
    }
  }
}
```

<code>static_group_settings_name</code>	A logical name for a static group mapping strategy.
<code>dynamic_group_settings_name</code>	A logical name for a dynamic group mapping strategy.
<code>hierarchy_group_settings_name</code>	A logical name for a hierarchy group mapping strategy.
<code>settings</code>	The LDAP settings reference.
<code>groupMemberAttribute</code>	The group membership attribute for this LDAP group.
<code>groupNameAttribute</code>	The group name attribute for this LDAP group.
<code>descriptionAttribute</code>	The description attribute for this LDAP group.
<code>filter</code>	A filter expression used to search for the user DN that is used in LDAP authentication.
<code>searchBase</code>	The Context name in which to search relative to the base DN in the LDAP URL.
<code>searchSubTree</code>	When true, enables deep search through the sub-tree of the LDAP URL + Search Base.

Security Settings

The configuration blocks for security settings related to users, groups and permissions are described below. For more details on these parameters, please refer to [Configuring Security](#) in the [Artifactory User Guide](#).


```

security {
  users {
    conflictResolutionPolicy "OVERRIDE" // default
    user('name') {
      email 'login_1@jfrog.com'
      password 'passwd_1'
      admin false
      profileUpdatable false
      internalPasswordDisabled false
      groups(['groupA', 'groupB']) // values(['groupA', 'groupB']) are examples. Please set existing
values from the instance
    }
  }

  groups {
    conflictResolutionPolicy "OVERRIDE" // default
    group('name') {
      description 'desc_1'
      autoJoin false
    }
  }

  permissions {
    conflictResolutionPolicy "OVERRIDE" // default
    permission('name') {
      includesPattern '**'
      excludesPattern ''
      anyLocal false
      anyRemote false
      anyDistribution false
      repositories(["local-rep1", "local-rep2"]) // values(["local-rep1", "local-rep2", ...]) are
examples. Please set existing values from the instance
      users {
        userA(['manage', 'delete', 'deploy', 'annotate', 'read']) // value userA - is example. Please set
existing user names from the instance
      }
      groups {
        groupsG1(['manage', 'delete', 'deploy', 'annotate', 'read']) // value groupsG1 - is example. Please
set existing group names from the instance
      }
    }
  }
}

```

Users block	Creates or updates users in the instance
conflictResolutionPolicy	Default: OVERRIDE Specifies what to do if any setting in the users block of the configuration script conflicts with an existing value for the specified user. Currently, the only option available is the default OVERRIDE which means that values specified in the configuration script will override any existing values.
email	The user's email address.
password	The user's login password.
admin	When true, this user is an administrator with all the ensuing privileges
profileUpdatable	When true, this user can update their profile details (except for the password. Only an administrator can update the password).
internalPasswordDisabled	When true, disables the fallback of using an internal password when external authentication (such as LDAP) is enabled.
groups	Specifies the groups to which this user should belong

Groups block	Creates or updates groups in the instance
conflictResolutionPolicy	Default: OVERRIDE Specifies what to do if any setting in the groups block of the configuration script conflicts with an existing value for the specified user. Currently, the only option available is the default OVERRIDE which means that values specified in the configuration script will override any existing values.
description	A free text description for the group.
autoJoin	When true, any new users defined in the system are automatically assigned to this group.
Permissions block	Creates or updates permission targets in the instance.
conflictResolutionPolicy	Default: OVERRIDE Specifies what to do if any setting in the permissions block of the configuration script conflicts with an existing value for the specified user. Currently, the only option available is the default OVERRIDE which means that values specified in the configuration script will override any existing values.
includesPattern/excludesPattern	"Ant-like" expressions that specify repositories and paths to be included or excluded from the permission target
anyLocal	When true, all local repositories are included in the permission target.
anyRemote	When true, all remote repositories are included in the permission target.
anyDistribution	When true, all distribution repositories are included in the permission target.
repositories	Specific repositories on which to apply the permission target.
users	The users on whom to apply the permission target and the corresponding permissions they are given.
groups	The groups on which to apply the permission target and the corresponding permissions they are given.

Repository Configuration Blocks

This section presents all the configuration blocks that may be used to configure Artifactory repositories.

Local Repository

The parameters for a `localRepository` block are described below. For more details on these parameters, please refer to [Common Settings](#) and [Local Repositories](#) in the [Artifactory User Guide](#).

localRepository block

```
localRepository('repository-key') {
  description "Public description"
  notes "Some internal notes"
  includesPattern "**/*" // default
  excludesPattern "" // default
  repoLayoutRef "maven-2-default"
  packageType "generic" // "maven" | "gradle" | "ivy" | "sbt" | "nuget" | "gems" | "npm" | "conan" | "helm" |
                                     // "bower" | "debian" | "pypi" | "docker" | "vagrant" |
  "gitlfs" | "yum" | "generic"
  debianTrivialLayout false
  checksumPolicyType "client-checksums" // default | "server-generated-checksums"
  handleReleases true // default
  handleSnapshots true // default
  maxUniqueSnapshots 0 // default
  snapshotVersionBehavior "unique" // "non-unique" | "deployer"
  suppressPomConsistencyChecks false // default
  blackedOut false // default
  propertySets // ([ "ps1", "ps2" ])
  archiveBrowsingEnabled false
  calculateYumMetadata false
  yumRootDepth 0
  xrayIndex false
  blockXrayUnscannedArtifacts false
  xrayMinimumBlockedSeverity "" // "Minor" | "Major" | "Critical"
  enableFileListsIndexing ""
  yumGroupFileNames ""
}
```

repository-key	The Repository Key is a mandatory identifier for the repository and must be unique within an Artifactory instance. It cannot begin with a number or contain spaces or special characters. For local repositories we recommend using a "-local" suffix (e.g. "libs-release-local").
description	A free text field that describes the content and purpose of the repository.
notes	A free text field to add additional notes about the repository. These are only visible to the Artifactory administrator and to Mission Control.
includesPattern and excludesPattern	These parameters provide a way to filter out specific repositories when trying to resolve the location of different artifacts.
repoLayoutRef	Sets the layout that the repository should use for storing and identifying modules. The layout should correspond to the value set in the packageType property.
packageType	The repository's package type.
debianTrivialLayout	Only valid if packageType is set to Debian. If true, the Debian repository will use the Trivial layout.
checksumPolicyType	Only valid if for Maven, Gradle, Ivy and SBT repositories. Determines how Artifactory behaves when a client checksum for a deployed resource is missing or conflicts with the locally calculated checksum.
handleReleases	If true, users will be able upload Release artifacts to this repository
handleSnapshots	If true, users will be able upload Snapshot artifacts to this repository
maxUniqueSnapshots	Specifies the maximum number of unique snapshots of the same artifact that should be stored. A value of 0 (default) indicates that there is no limit on the number of unique snapshots.
snapshotVersionBehavior	Artifactory supports centralized control of how snapshots are deployed into a repository, regardless of end user-specific settings. This can be used to guarantee a standardized format for deployed snapshots within your organization.
suppressPomConsistencyChecks	If true, Artifactory will reject a deployment in which the <code>groupId:artifactId:version</code> set in the path conflicts with the deployed path.
blackedOut	If true, Artifactory will ignore this repository when trying to resolve, download or deploy artifacts.

propertySets	Defines the property sets that will be available for artifacts stored in this repository.
archiveBrowsingEnabled	If true, allows users to view archive file contents (e.g., Javadoc browsing, HTML files) directly from Artifactory.
calculateYumMetadata	Only valid for YUM repositories. If true, YUM metadata calculation will be automatically triggered for the events described in Triggering RPM Metadata Updates .
yumRootDepth	Only valid for YUM repositories. Informs Artifactory under which level of directory to search for RPMs and save the repository directory.
xrayIndex	If true, this repository should be indexed by JFrog Xray connected to the Artifactory instance.
blockXrayUnscannedArtifacts	If true, artifacts that have not yet been scanned by the connected JFrog Xray will be blocked for download.
xrayMinimumBlockedSeverity	The minimum severity of an issue detected for an artifact to be blocked for download.
enableFileListsIndexing	For an RPM repository, this field specifies if the RPM file lists metadata file should be indexed by Artifactory or not.

Remote Repository

The parameters for a `remoteRepository` block are described below. For more details on these parameters, please refer to [Common Settings](#) and [Remote Repositories](#) in the [Artifactory User Guide](#).



remoteRepository block

```
remoteRepository('repository-key') {
    url "http://host:port/some-repo"
    username "remote-repo-user"
    password "pass"
    proxy "proxy1"
    description "Public description"
    notes "Some internal notes"
    includesPattern "**/*" // default
    excludesPattern "" // default
    packageType "generic" // "maven" | "gradle" | "helm" | "ivy" | "sbt" | "nuget" | "gems" | "npm" | "bower"
    | "debian" | "pypi" | "docker" | "yum" | "vcs" | "p2" | "generic"
    remoteRepoChecksumPolicyType "generate-if-absent" // default | "fail" | "ignore-and-generate" | "pass-thru"
    handleReleases true // default
    handleSnapshots true // default
    maxUniqueSnapshots 0 // default
    suppressPomConsistencyChecks false // default
    offline false // default
    blackedOut false // default
    storeArtifactsLocally true // default
    socketTimeoutMillis 15000
    localAddress "123.123.123.123"
    retrievalCachePeriodSecs 43200 // default
    failedRetrievalCachePeriodSecs 30 // default
    missedRetrievalCachePeriodSecs 7200 // default
    unusedArtifactsCleanupEnabled false // default
    unusedArtifactsCleanupPeriodHours 0 // default
    fetchJarsEagerly false // default
    fetchSourcesEagerly false // default
    shareConfiguration false // default
    synchronizeProperties false // default
    propertySets // (["ps1", "ps2"])
    allowAnyHostAuth false // default
    enableCookieManagement false // default
    xrayIndex false
    blockXrayUnscannedArtifacts false
    xrayMinimumBlockedSeverity "" // "Minor" | "Major" | "Critical"
    enableFileListsIndexing ""
}
```

repository-key	Please refer to the description for this parameter in a Local Repository block.
url	The URL for the remote repository. Currently only HTTP and HTTPS URLs are supported.
username	The username that should be used for HTTP authentication when accessing the remote proxy.
password	The password that should be used for HTTP authentication when accessing the remote proxy.
proxy	If the organization in which the Artifactory instance is hosted requires users to go through a proxy to access a remote repository, this parameter lets you select the corresponding Proxy Key .
description	Please refer to the description for this parameter in a Local Repository block.
notes	Please refer to the description for this parameter in a Local Repository block.
includesPattern	Please refer to the description for this parameter in a Local Repository block.
excludesPattern	Please refer to the description for this parameter in a Local Repository block.
packageType	Please refer to the description for this parameter in a Local Repository block.

remoteRepositoryChecksumPolicyType	Specifies how the Artifactory instance should behave when a client checksum for a remote resource is missing or conflicts with the locally calculated checksum.
handleReleases	Please refer to the description for this parameter in a Local Repository block.
handleSnapshots	Please refer to the description for this parameter in a Local Repository block.
maxUniqueSnapshots	Please refer to the description for this parameter in a Local Repository block.
suppressPomConsistencyChecks	Please refer to the description for this parameter in a Local Repository block.
offline	If true, the repository will be considered offline and no attempts will be made to fetch artifacts from it.
blackedOut	Please refer to the description for this parameter in a Local Repository block.
storeArtifactsLocally	If true, artifacts from the repository will be cached locally. If not set, direct repository-to-client streaming is used.
socketTimeoutMillis	The time that the Artifactory instance should wait for both a socket and a connection before giving up on an attempt to retrieve an artifact from a remote repository.
localAddress	When working on multi-homed systems, this parameter lets you specify which specific interface (IP address) should be used to access the remote repository.
retrievalCachePeriodSecs	Defines how long before the Artifactory instance should check for a newer version of a requested artifact in a remote repository. A value of 0 means that Artifactory will always check for a newer version.
missedRetrievalCachePeriodSecs	If a remote repository is missing a requested artifact, Artifactory will return a "404 Not found" error. This response is cached for the period of time specified by this parameter. During that time, Artifactory will not issue new requests for the same artifact. A value of 0 means that the response is not cached and Artifactory will always issue a new request when demanded.
unusedArtifactsCleanupPeriodHours	Specifies how long an unused artifact will be stored in the Artifactory instance before it is removed. A value of 0 means that the artifact is stored indefinitely.
fetchJarsEagerly	If true, if a POM is requested, the Artifactory instance attempts to fetch the corresponding jar in the background. This will accelerate first access time to the jar when it is subsequently requested.
fetchSourcesEagerly	If true, if a binaries jar is requested, the Artifactory instance attempts to fetch the corresponding source jar in the background. This will accelerate first access time to the source jar when it is subsequently requested.
synchronizeProperties	Only valid for Smart Remote Repositories . If true, properties for artifacts that have been cached in the repository will be updated if they are modified in the artifact hosted at the remote Artifactory instance.
propertySets	Please refer to the description for this parameter in a Local Repository block.
allowAnyHostAuth	If true, allows using the repository credentials on any host to which the original request is redirected.
enableCookieManagement	If true, the repository will allow cookie management to work with servers that require them.
xrayIndex	Please refer to the description for this parameter in a Local Repository block.
blockXrayUnscannedArtifacts	Please refer to the description for this parameter in a Local Repository block.
xrayMinimumBlockedSeverity	Please refer to the description for this parameter in a Local Repository block.

enableFileListsIndexing

For an RPM repository, this field specifies if the RPM file lists metadata file should be indexed by Artifactory or not.

Virtual Repository

The parameters for a `virtualRepository` block are described below. For more details on these parameters, please refer to [Common Settings](#) and [Virtual Repositories](#) in the [Artifactory User Guide](#).

virtualRepository block

```
virtualRepository('repository-key') {
  repositories (["local-rep1", "local-rep2"]) // values (["local-rep1", "local-rep2", ...]) are examples.
  Please set existing values from the instance
  description "Public description"
  notes "Some internal notes"
  includesPattern "**/*" // default
  excludesPattern "" // default
  packageType "generic" // "maven" | "gradle" | "helm" | "ivy" | "sbt" | "nuget" | "gems" | "npm" | "bower"
  | "pypi" | "p2" | "generic"
  debianTrivialLayout false
  artifactoryRequestsCanRetrieveRemoteArtifacts false
  keyPair "keypair1" //value "keypair1" is example. Please set existing value from the instance
  pomRepositoryReferencesCleanupPolicy "discard_active_reference" // default | "discard_any_reference" |
  "nothing"
  defaultDeploymentRepo "local-rep1"
}
```

repository-key	Please refer to the description for this parameter in a Local Repository block.
repositories	The list of repositories that should be aggregated in this virtual repository.
description	Please refer to the description for this parameter in a Local Repository block.
notes	Please refer to the description for this parameter in a Local Repository block.
includesPattern	Please refer to the description for this parameter in a Local Repository block.
excludesPattern	Please refer to the description for this parameter in a Local Repository block.
packageType	Please refer to the description for this parameter in a Local Repository block.
debianTrivialLayout	Please refer to the description for this parameter in a Local Repository block.
artifactoryRequestsCanRetrieveRemoteArtifacts	If true, the virtual repository should search through remote repositories when trying to resolve an artifact requested by another Artifactory instance.
keyPair	A named key-pair to use for automatically signing artifacts.
pomRepositoryReferencesCleanupPolicy	This setting gives you the ability to ensure Artifactory is the sole provider of Artifacts in your system by automatically cleaning up the POM file.

Replication

A replication block is used for creating push/pull replication. This block should always be nested inside a repository block: `localRepository` for push replication, `remoteRepository` for pull replication or a `repository` block for both.

replication block

```
localRepository("example") {  
  replication(<REPLICATION_TARGET>) {  
    cronExp "0 0/9 14 * * ?"  
    socketTimeoutMillis 15000  
    username "remote-repo-user"  
    password "pass"  
    proxy // "proxy-ref"  
    enableEventReplication true  
    enabled true  
    syncDeletes false  
    syncProperties true  
    clientTlsCertificate ""  
  }  
}
```

REPLICATION_TARGET

Target local repository for push replication. There are two ways to provide a target repository:

1. **Target repository URL** - the URL of the target local repository (String)
2. **Target Artifactactory object** - the service object the target repository belongs to. This applies **only** when creating a new repository. Mission control will select the repository about to be created on the target instance within the current configuration context (current configuration action). An instance object can be obtained from the context variables or by asking for user input of type ARTIFACTORY.

For pull replication there is no need to set the target repository

cronExp

Define the replication task schedule using a valid [cron](#) expression

socketTimeoutMillis

The network timeout in milliseconds to use for remote operations

username

The HTTP authentication username

password

The HTTP authentication password

proxy

The key of a proxy configuration to use when communicating with the remote instance

enableEventReplication

When set, event-based push replication is enabled

enabled

When set to true, this replication will be enabled

syncDeletes

When set, items that were deleted remotely should also be deleted locally

syncProperties

When set, the task also synchronizes the properties of replicated artifacts

clientTlsCertificate

The SSL/TLS certificate used for authentication

Star Topology

Mission Control provides built-in configuration blocks that make it very easy to create replication relationships between different Artifactory services in a star topology. This is done using the following two key words to create push replication and pull replication configurations respectively:

- starPush
- starPull

Push Replication

The example below shows how to create a star topology using push replication.



Star topology using multi-push replication

This example shows:

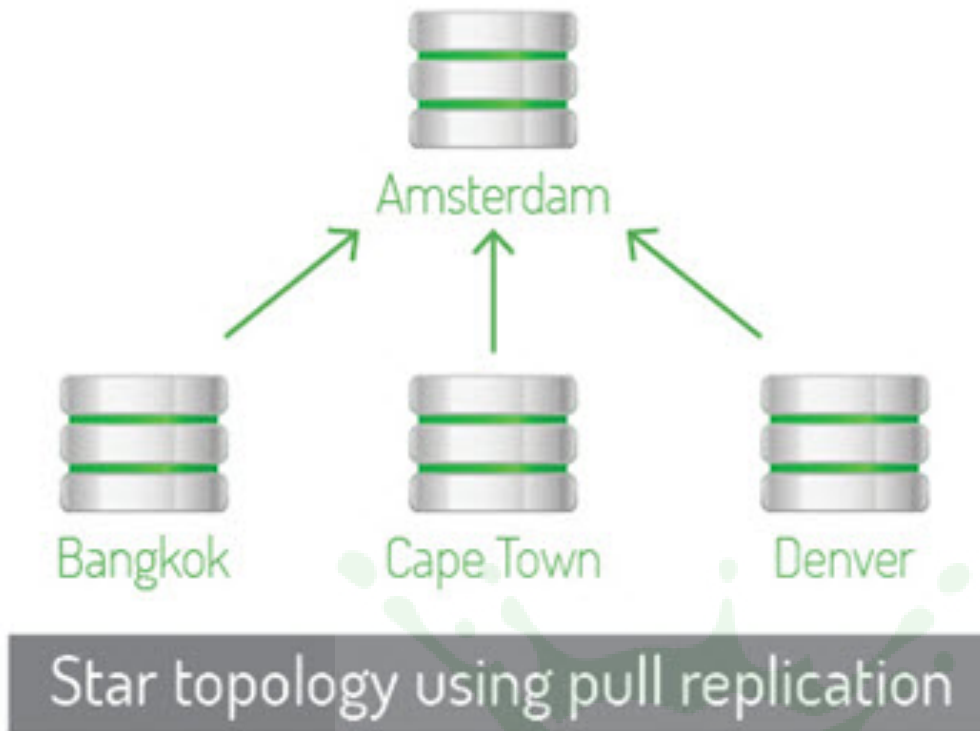
- An Artifactory service named "Amsterdam" with two local repositories, "maven-local-1" and "maven-local-2"
- The configuration block will create or update a push replication relationship between "maven-local-1" and "maven-local-2" in "Amsterdam" to the corresponding repositories in each of the Artifactory services named "Bangkok", "Cape Town" and "Denver".
- If "maven-local-1" or "maven-local-2" do not already exist in any of the target Artifactory services, Mission Control will create them

multi-push replication block

```
artifactory('Amsterdam') {  
  repository("maven-local-1", "maven-local-2") {  
    starPush('Bangkok', 'Cape Town', 'Denver')  
  }  
}
```

Pull Replication

The example below shows how to create a star topology using pull replication.



This example shows:

- An Artifactory service named "Amsterdam" with two repositories, "maven-local" and "maven-virtual"
- The configuration block will create or update a pull replication relationship in corresponding repositories in each of the Artifactory services named "Bangkok", "Cape Town" and "Denver" to pull replicate from the "maven-local" and "maven-virtual" repositories in "Amsterdam" .
- If "maven-local" or "maven-remote" don't exist in "Bangkok", "Cape Town" or "Denver", Mission Control will create them

multi-push replication block

```
artifactory('Amsterdam') {
  repository("maven-local", "maven-virtual") {
    starPull('Bangkok', 'Cape Town', 'Denver')
  }
}
```

Xray Configuration Blocks

This section presents configuration blocks that can be used to configure different administrative features of Xray services. As with any configuration block, these must be placed within an Xray [service closure](#) as shown below.

Artifactory service closure

```
xray('<Xray service name>'){
  <configuration blocks>
}
```

Link to Artifactory - Binary Manager

The following is an example of a configuration block that adds an Artifactory service as a binary manager to an Xray service.

If `Artifactory-prod` is already registered as a service in Mission Control, the admin login credentials specified when registering will be provided to Xray when configuring it as the binary manager.

```
{
  binaryManager('Artifactory-prod')
}
```

If `Artifactory-prod` is not registered as a service in Mission Control, you need to provide all properties of the service as follows:

```
{
  binaryManager('Artifactory-prod') {
    url 'http://artifactory.com/artifactory'
    login 'login'
    password 'password'
  }
}
```

In this case, Mission Control will create an Artifactory service with these properties and then configure the Xray service in the enclosing [service closure](#) with it as its binary manager.

Watches

Below is an example of a watch configuration block. For more details on these parameters, please refer to [Watches](#) in the [JFrog Xray User Guide](#).

```
{
  watch('watch') {
    binaryManagerId 'binaryManagerId'
    targetType 'repository'
    description 'description'
    active true
    postActions {
      emails(['email1@email.com', 'email2@email.com'])
      slacks 'slacks'
      webhooks(['webhook1', 'webhook2'])
      failBuild true
    }
    filters {
      filter {
        type 'license_black'
        value 'value1'
      }
      filter {
        type 'regex'
        value 'value1'
      }
    }
    repoType 'repoType'
    severity 'severity'
    system true
    targetName 'targetName'
    temp true
  }
}
```

Git Integration

Overview

In addition to storing and managing [configuration scripts](#) in an internal database, Mission Control also lets you synchronize them to a version control systems allow versioning, sharing and collaboration on scripts as well as allowing you to create scripts using any editor you prefer. Currently, the only VCS system supported is Git.

This section describes how to configure Mission Control to manage configuration scripts in a Git repository.

Page Contents

- [Overview](#)
- [Configuring Git Integration](#)

Configuring Git Integration

To configure Mission Control to access the Git repository where you host your configuration scripts, in the **Admin** module, select **Scripts | Version Control**.

You can access your Git account using your username and password or via SSH.



Using SSH to access your Git account?

When using SSH to access your Git account, you don't need to enter your username and password.

Note also that your keys must to be under the `~/.ssh` folder of the machine on which Mission Control is running.

Check **Enable Git** and fill in the details of the integration.

The screenshot shows the 'Version Control' configuration page in JFrog Mission Control. The page is titled 'Version Control' and contains a 'Git Integration' section. This section has a checkbox for 'Enable Git', three input fields for 'Remote URL *', 'User Name', and 'Password', and two checkboxes for 'Bidirectional' and 'Restore scripts from remote repositories'. An 'Ok' button is at the bottom right.

Remote URL	URL to the Git repository where scripts will be hosted. You may only work with the master branch and it must exist before being configured in Mission Control. You may access the Git repository via SSH, and in this case, the User Name and Password fields may remain empty.
User Name	A valid Git user with pull and push permissions.
Password	Password for the Git account. This only needs to be provided when initially enabling Git integration, or if the password was updated.
Bidirectional	When set, scripts can be developed externally to Mission Control (in the user's IDE), then pushed to Remote URL and loaded into Mission Control. Note that empty scripts do not get synchronized to your Git repository.

Restore scripts from repository	When set, scripts are loaded from the Git repository to Mission Control immediately after the configuration is saved.
---------------------------------	---



Exploring Sites

Overview

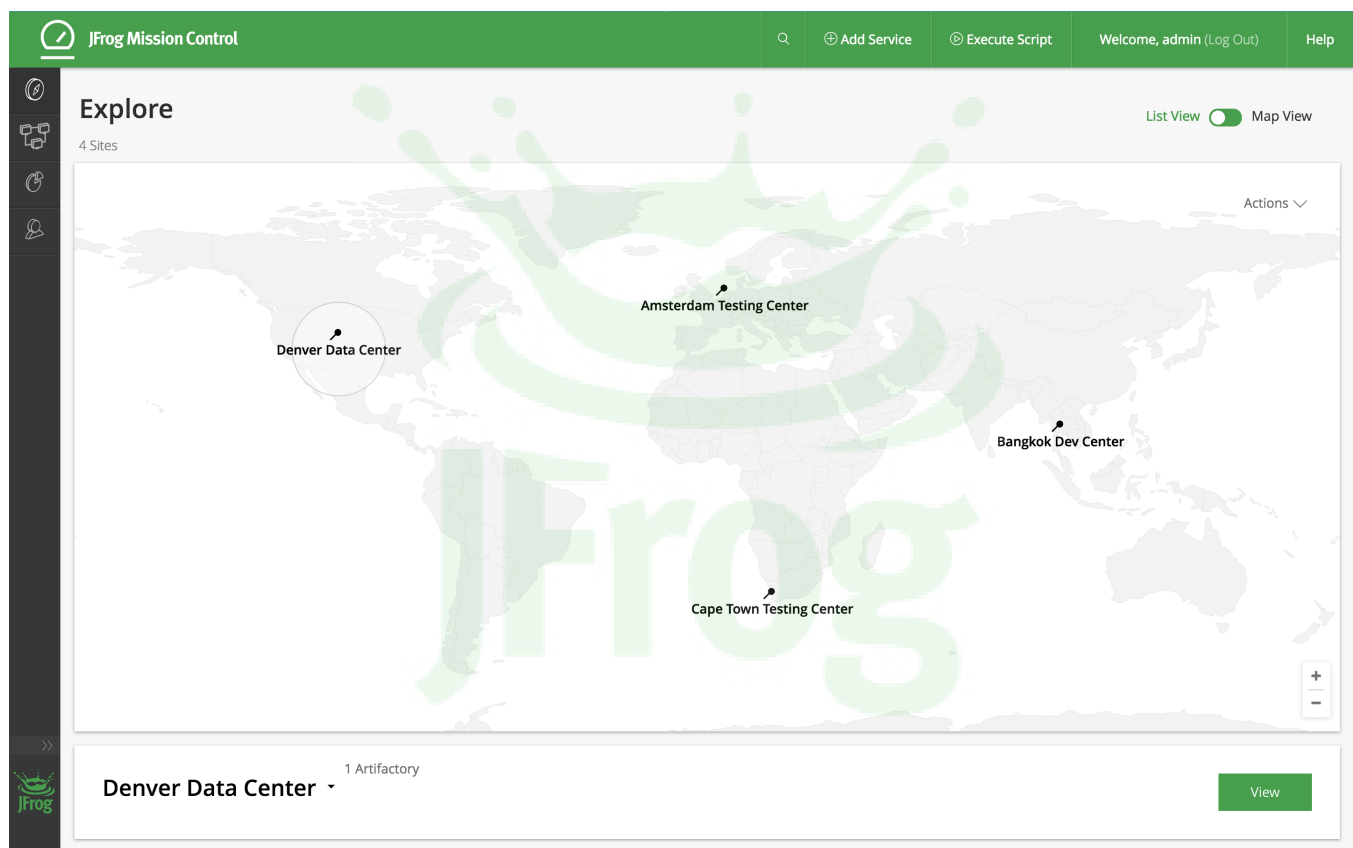
A site, in Mission Control, represents a set of managed services grouped into geographical locations. This gives you an easy way to identify the various services managed by Mission Control according to their physical proximity to each other and manage them as a group.

The site explorer is your landing page into Mission Control and can present your sites either on a **map** or as a **list**. You can toggle between these two views with the toggle at the top right corner of the screen.

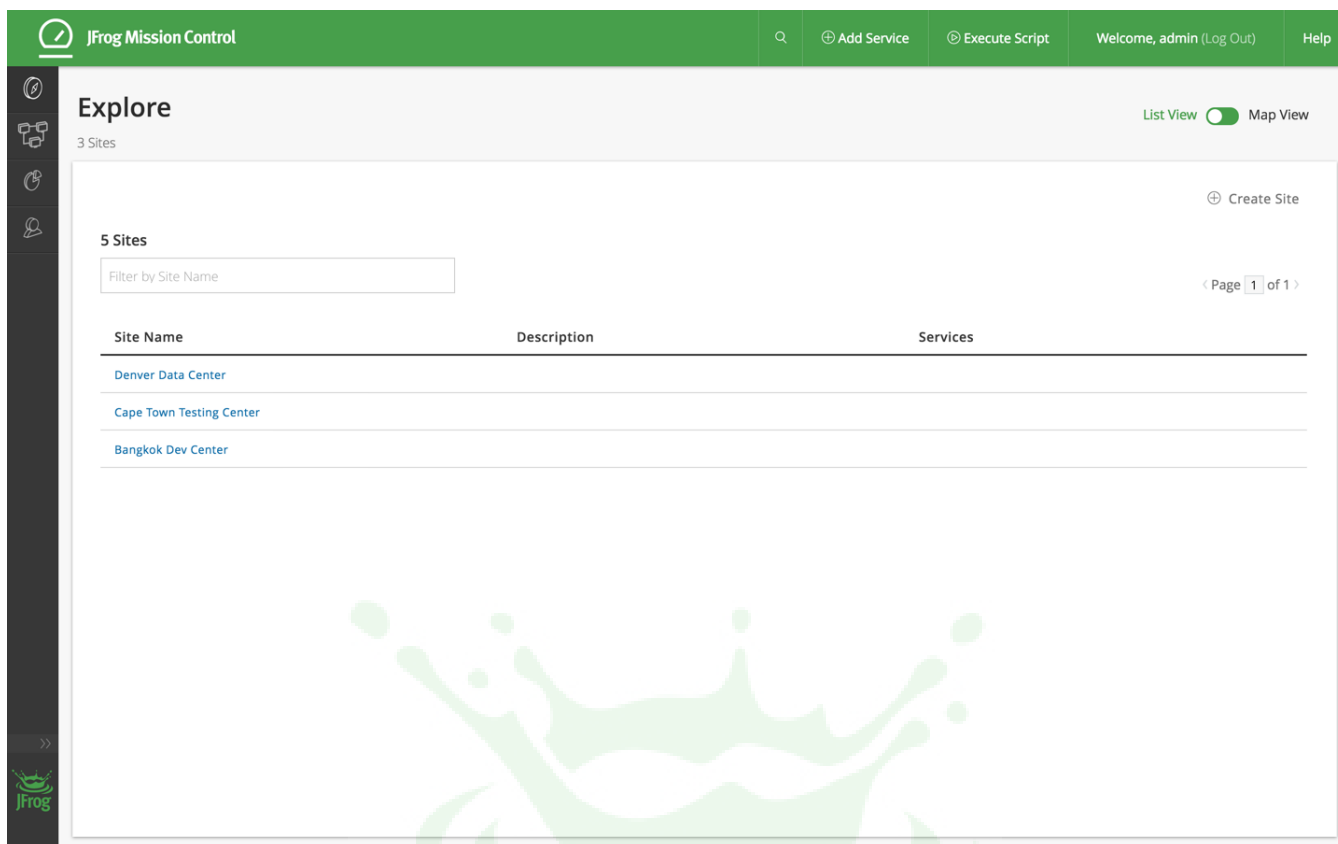
Page Contents

- [Overview](#)
 - [Map View](#)
 - [List View](#)
- [Managing a site](#)
 - [Creating a Site](#)
 - [Viewing a Site](#)
 - [Selecting Services within a Site](#)
 - [Filtering Services within a Site](#)

Map View



List View

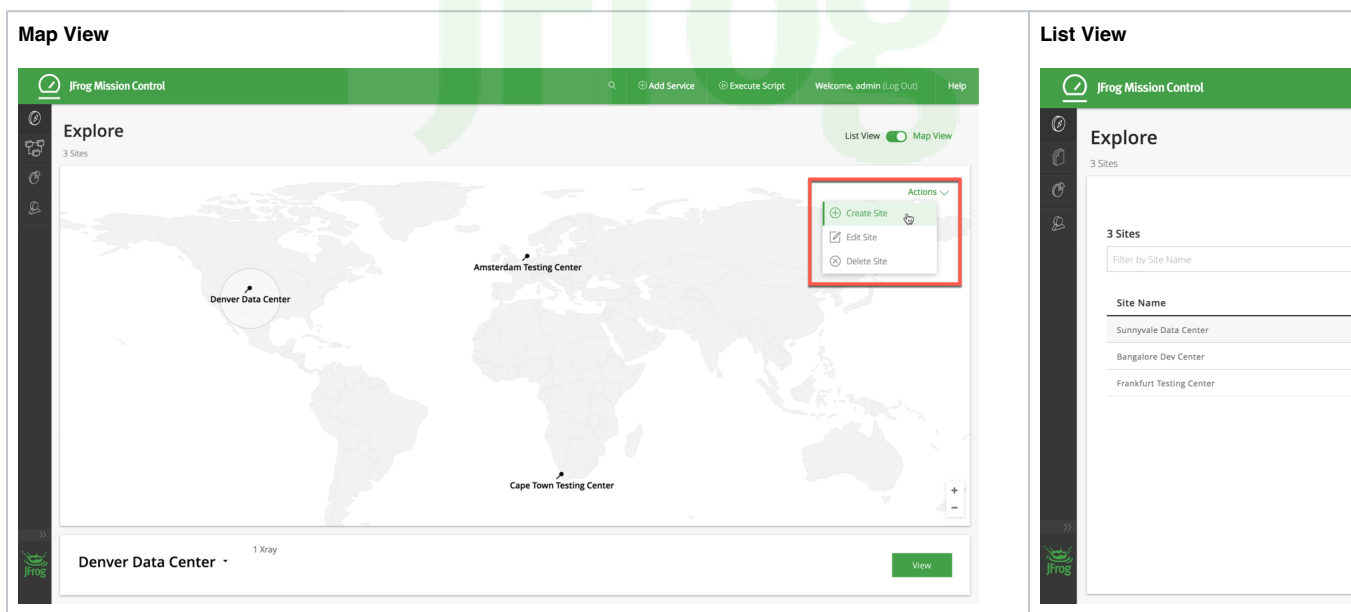


Managing a site

Creating, editing and deleting a site can be done in the **Map View** from the Actions menu and in the **List View**.



A site that contains services cannot be deleted. All services needs to be removed 1st and than delete the site.



Creating a Site

To create a new site, click "Create Site" in the **List View** or select "Create Site" from the **Actions** menu in the **Map View**.

Create Site

Site Name *

New York

Description

Production site

Location *

City Name: New York City (US) Coordinates: 40.714270 -74.005970

Cancel

Create

Site Name	A logical name for the site.
Description	A description of the site (optional).
Location	The name of the city in which the site is located. You can specify more than one site in a city.

Viewing a Site

The site page allows you to view details on the services within a site.

It is sectioned into three parts:

Services List	All services within the site. Select a service from the list to highlight it in the connections diagram below. Click on a specific service to view its service page .
Connections Diagram	A visual representation of the connections between the services, and connections to external services in other sites. Hover over the connections and services to view additional details such as replication type and service status.
External Connections	Displays a list of other sites that have services connected to services in this site. In the example below, JFIL is a site that has 2 service connections to services in the GCP-EU site.

GCP-EU

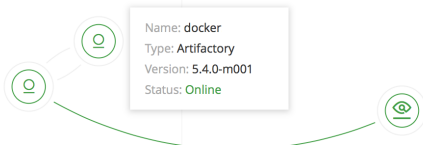
Service Name ▾

Service List

Page 1 of 1

Service Type	Service Name	Status	Version	Site	License	License valid Thru
○	repo	ONLINE	5.6.0-m001	GCP-EU	HA	Jun 5, 2018
○	docker	ONLINE	5.4.0-m001	GCP-EU	HA	Oct 12, 2018
○	xray-prod	ONLINE	1.8.6.1	GCP-EU	COMMERCIAL	

Connections



External Connections

JFIL
2 connections

GCP-US
12 connections

AS-SANDBOX

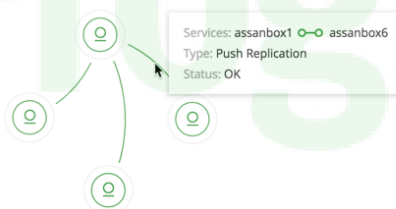
Service Name ▾

Service List

Page 1 of 1

Type	Name	URL	Version	Status	Status Detail...	Site	License	License valid...
○	assanbox1		5.5.2	ONLINE	✓	AS-SANDBOX	ENTERPRISE	May 15, 2018
○	assanbox2		5.5.2	ONLINE	✓	AS-SANDBOX	PRO	Jun 21, 2036
○	assanbox3		5.5.2	ONLINE	✓	AS-SANDBOX	PRO	May 15, 2018
○	assanbox6		5.5.2	ONLINE	✓	AS-SANDBOX	ENTERPRISE	May 15, 2018

Connections



External Connections

NA-DEV-PROD
1 connections

EU-CUSTOMER-HUB
1 connections

EU-DEV-STAGING
1 connections

SA-DEV-PROD
1 connections

Selecting Services within a Site









To view details on a specific service in a site, select it from the Service List or directly from the Connections view. Hover over the service to see additional details on it.

AS-SANDBOX

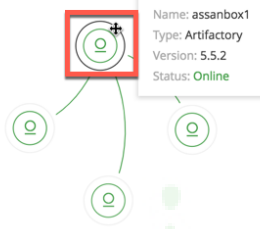
Service Name

Service List

< Page 1 of 1 >

Type	Name	URL	Version	Status	Status Detail...	Site	License	License valid...
	assanbox1	http://104.196.248.15:8084/artifacto...	5.5.2	ONLINE		AS-SANDBOX	ENTERPRISE	May 15, 2018
	assanbox2	http://104.196.248.15:8085/artifacto...	5.5.2	ONLINE		AS-SANDBOX	PRO	Jun 21, 2036
	assanbox3	http://104.196.248.15:8086/artifacto...	5.5.2	ONLINE		AS-SANDBOX	PRO	May 15, 2018
	assanbox6	http://104.196.248.15:8089/artifacto...	5.5.2	ONLINE		AS-SANDBOX	ENTERPRISE	May 15, 2018

Connections



External Connections

NA-DEV-PROD
1 connections

EU-CUSTOMER-HUB
1 connections

EU-DEV-STAGING
1 connections

SA-DEV-PROD
1 connections

Filtering Services within a Site

To find a service within a site, filter it by type and name.


SA-DEV-PROD


Type

Select (1)

Select





Select All Unselect All

☒  Artifactory

☐  Xray

Service List

< Page 1 of 1 >

Type	Name	URL	Version	Status	Status Detail...	Site	License	License valid...
	sadevp1	http://130.211.213.126:8081/artifact...	5.5.2	ONLINE		SA-DEV-PROD	ENTERPRISE	Jun 26, 2018
	sadevp2	http://130.211.213.126:8082/artifact...	5.5.2	ONLINE		SA-DEV-PROD	ENTERPRISE	Jun 26, 2018

Connections



External Connections

AS-SANDBOX
2 connections

NA-DEV-PROD
2 connections

EU-CUSTOMER-HUB
2 connections

EU-DEV-STAGING
1 connections

Managing Services

Overview

Artifactory and Xray services are managed in Mission Control grouped together within a [site](#). Each service managed by Mission Control must be contained within a site.

The services module displays all services.



Existing Artifactory services are automatically assigned to a site

When upgrading to Mission Control 2.0, any Artifactory services already managed by your current version of Mission Control will be assigned to new Sites that will be created according to the location of your Artifactory instances. For example, an Artifactory service located in San Francisco, will be assigned to a new site in Mission Control that's located in San Francisco.

Services without a location will be added to the "Default" site. Create your sites based on your datacenter and add your services to them.

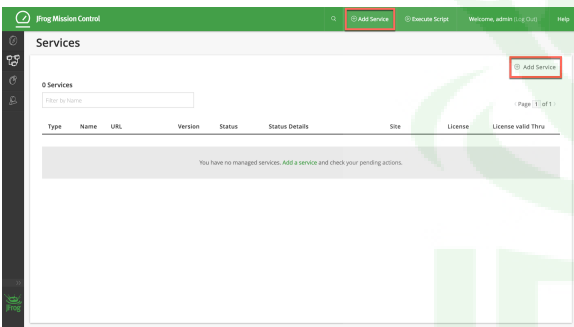
Page contents

- [Overview](#)
- [Adding Services](#)
- [Viewing All Services](#)
- [Viewing an Artifactory Service](#)
 - [Storage Summary](#)
 - [Replication](#)
 - [License](#)
 - [HA](#)
 - [Task Summary](#)
 - [System Info](#)
- [Viewing an Xray Service](#)
- [Editing and Deleting a Service](#)
- [Troubleshooting](#)

Adding Services

Once you have created your sites, you can add services to them according to their geographical locations.

To add a service click "Add Service" from the **Header** or the **Services** module.



Fill in the service details and click "Test Connectivity" to check your authorization. Make sure to select the type of service you are adding, Artifactory or Xray.

Once the connectivity is ok and select the site to add the service to. If you do not have a site to add your service to, click ["Create Site"](#).

JFrog Mission Control will validate the service and connect to it. Once added, your service will be assigned to the specified site and displayed in the Service module.

JFrog Mission Control
Add Service
Execute Script
Welcome, admin (Log Out)
Help

Add Service

Service Settings

Name *
Type *
ARTIFACTORY
URL *
Description

Set credentials

User Name *
Password *
Test Connectivity

Proxy(Optional)

Proxy
-- no proxy --

Select Site

4 Sites
Filter by Name
Page 1 of 1

Name	Description	Location	Services
JFrog insight		Bengaluru	
JFMC @ Toulouse Marais		Toulouse	RT-source
JFMC Lohika @ Lviv		Lviv	Efrat-5-8, RT-Dest2
bucket-testing		Sunnyvale	mill-1, mill-2

Create Site

Cancel Add

Name	A logical name for the service.
Type	The service type. Artifactory or Xray.
URL	The service URL.
Description	A description of the service (optional).
User Name	The service administrator user name.
Password	The service administrator password.
Proxy	The proxy through which MC accesses the service (optional).
Sites	The site to assign your service to.

Once the connectivity is ok, click "Next".

Add Service

Success

Service Details

Select Site

Service Settings

Name *

Xray-Sunnyvale

Type *

XRAY

URL *

Description

Set credentials

User Name *

admin

Password *

Test Connectivity

Proxy(Optional)

Proxy

-- no proxy --

Cancel

Back

Next

Select the site to add the service to, and click "Add". If you do not have a site to add your service to, click ["Create Site"](#).

Add Service

Service Details

Select Site

Selected Service

Version

Url

Status

Error

ONLINE

Site is missing

Select Site

5 Sites

Filter by Name

Page 1 of 1

Name	Description	Location	Services
<input type="radio"/> Amsterdam Testing Center		Amsterdam	
<input checked="" type="radio"/> Denver Data Center		Denver	
<input type="radio"/> Cape Town Testing Center		Cape Town	

Cancel

Back

Add

JFrog Mission Control will validate the service and connect to it. Once added, your service will be assigned to the specified site and displayed in the Service module.

Viewing All Services

The Services module allows you to view all services managed by Mission Control in one place.

JFrog Mission Control

Add Service
Execute Script
Welcome, admin (Log Out)
Help

Explore
Services
Graphs
Admin

Services

Add Service

2 Services

Filter by Name

Page 1 of 1

Type	Nam...	URL	Version	Status	Status Detail...	Site	License	License valid Thru
○	RT2	http://mill.jfrog.info:12149/artifactory	5.5.2	ONLINE	✓	New York Production	HA	Jun 29, 2018
○	RT4	http://mill.jfrog.info:12157/artifactory	5.5.2	ONLINE	✓	pp	HA	Sep 2, 2018

Mission Control
(build)
Licensed to jfrog
© Copyright 2017 Jfrog Ltd.
Display a menu

Type	The service type. Artifactory or Xray.
Name	The service name.
URL	The service URL.
Status	The service status.
Status Details	The service status details.
Site	The site the service is in.
License	The service license type.
License valid Thru	The service license expiration date.

Viewing an Artifactory Service

To view a single Artifactory service, click on the service name or hover over it from the Service module and select the view icon.

JFrog Mission Control

Add Service
Execute Script
Welcome, admin (Log Out)
Help

Services

Add Service

16 Services

Filter by Name

Page 1 of 1

Type	Name	URL	Version	Status	Status Details	Site	License	License valid Thru
○	assanbox6	http://...	5.5.2	ONLINE	✓	AS-SANDBOX	ENTERPRISE	May 15, 2018
○	assanbox2	http://...	5.5.2	ONLINE	✓	AS-SANDBOX	PRO	Jun 21, 2036
○	assanbox3	http://...	5.5.2	ONLINE	✓	AS-SANDBOX	PRO	May 15, 2018
○	assanbox1	http://...	5.5.2	ONLINE	✓	AS-SANDBOX	ENTERPRISE	May 15, 2018

The Artifactory service page is broken up into two parts:

1. The [service details](#).
2. [Artifactory specific information](#).

Storage Summary

The **Storage Summary** tab displays information about the number of binaries and the storage volume they occupy. For full details, please refer to [Monitoring Storage](#) in the Artifactory documentation.

Replication

The **Replication** tab displays information on all replications in which the selected Artifactory service is involved. If many replications are configured for the instance, you can filter the list by source or destination repository.

Storage SummaryReplicationsLicenseHATask SummarySystem Info

3 Replications

Filter by Source or Destination

< Page 1 of 1 >

Type	Source	▲	Source Artifacts	Destination	Destination Artifacts	Status	Last Run	Duration	Next Run	Proxy	Run Now
▼ MULTIPUSH	frog-product-2-local			nadevp1 : fro...							▶
	frog-product-2-local	0		nadevp1 : fro...	0	ok	19-11-17 00:...	00:00:00	20-11-17 00:...		▶
	frog-product-2-local	0		euchub1 : fro...	0	ok	19-11-17 00:...	00:00:00	20-11-17 00:...		▶
	frog-product-2-local	0		eudev1 : fro...	0	ok	19-11-17 00:...	00:00:00	20-11-17 00:...		▶
	frog-product-2-local	0		sadevp1 : fro...	0	ok	19-11-17 00:...	00:00:00	20-11-17 00:...		▶
PULL	pond-sandbox	2005		euchub1 : cu...	0	ok	12-11-17 23:...	00:00:34	07-07-18 04:...		▶
▶ MULTIPUSH	swamp-test1-local			http://104.19...							▶

Type	The replication type (push, multipush, pull).
Source	The replication source.
Source Artifacts	The replication source artifact count.
Destination	The replication destination.
Destination Artifacts	The replication destination artifact count.
Status	The last replication status.
Last Run	The previous replication run date.
Duration	The last replication duration time.
Next Run	The next scheduled replication run date.
Proxy	The proxy through which MC accesses the Artifactory instance.
Run Now	Invokes the selected replication immediately irrespective of cron expression specified for it.

License

The **License** tab provides information on the license with which the service is activated.

Storage Summary	Replications	License	HA	Task Summary	System Info
1 License					
Filter by Status					
Page 1 of 1					
Status	Artifactory identifier	License Hash	Licensed to	Valid Through	License Type
Active	assanbox1		JFrog	May 15, 2018	ENTERPRISE

Status	The license status.
Artifactory identifier	The Artifactory identifier.
License Hash	A hash code of the license.
Licensed To	The organization registered as owning the installed license
Valid Through	The date at which the license's validity will lapse.
License Type	Specifies if the license is Pro or Enterprise (HA)

HA

The **HA** tab provides information related to the HA configuration of the Artifactory service.

Storage Summary
Replications
License
HA
Task Summary
System Info

2 Members
Page 1 of 1

ID	Address	Heartbeat	State	Role
artifactory-01	http://192.168.1.100:8081/artifactory	2017-11-19T15:08:22.905Z	RUNNING	Primary
artifactory-02	http://192.168.1.101:8081/artifactory	2017-11-19T15:08:26.587Z	RUNNING	Member

menu

ID	The unique Artifactory server name
Address	The fully qualified URL of the Artifactory server
Heartbeat	The last time the server signaled that it is up and running.
State	The current state of the server.
Role	The Artifactory node role.

For more details, please refer to [Managing the HA Cluster](#) in the [Artifactory User Guide](#).

Task Summary

The **Task Summary** tab displays information on all running tasks on the selected Artifactory instance. This information can be used for advanced analysis of events or issues that may occur in the selected instance.

Storage Summary	Replications	License	HA	Task Summary	System Info
26 Tasks					
Filter by Name					
Page 1 of 2					
Name	State	Started	Description	NodeID	
BinaryStoreManagerServiceImpl.BinaryStoreErr...	scheduled		binary store error notification		
RemoteReplicationJob	scheduled		Remote Pull Replication - pond-sandbox		
XrayIndexEventJob	scheduled		Xray Indexing Events		
BackupJob	scheduled		Backup - backup-daily		
LocalReplicationJob	scheduled		Local Push Replication - swamp-test1-local		
StatsDelegatingServiceFlushJob	scheduled		Download Statistics Flushing Delegator		


System Info

The System Info tab displays the full set of system properties and environment variables for the selected Artifactory service.


Storage Summary	Replications	License	HA	Task Summary	System Info
User Info:					
user.dir					
user.home					
user.language					
user.name					
user.timezone					
Host Info:					
Available Processors					
Heap Memory Usage-Committed					
Heap Memory Usage-Init					
Heap Memory Usage-Max					
Heap Memory Usage-Used					


Viewing an Xray Service


To view a single Xray service, click on the service name or hover over it from the Service module and select the view icon.



JFrog Mission Control







 Add Service

 Execute Script

Welcome, admin (Log Out)

Help














Services

16 Services

Filter by Name

Page 1 of 1

Type ▼	Name	URL	Version	Status	Status Details	Site	License	License valid Thru
	sa-xray	https://...	1.8.6.2	OFFLINE	 Failed to connect to the s...	SA-DEV-PROD	COMMERCIAL	  
	euchub1	http://...	5.5.2	ONLINE		EU-CUSTOMER-HUB	ENTERPRISE	Jun 26, 2018
	euchub3	http://...	5.5.2	ONLINE		EU-CUSTOMER-HUB	ENTERPRISE	May 15, 2018
	assanbox6	http://...	5.5.2	ONLINE		AS-SANDBOX	ENTERPRISE	May 15, 2018

The service page is broken up into two parts:

1. The service details.
2. The Artifactory services that are connected to this Xray service.

The screenshot shows the JFrog Mission Control interface. At the top, there's a green header with the JFrog logo and navigation links. The main content area displays the details for the 'xray1' service, including its description, version (1.8.6.2), URL, site (Site1), and status (ONLINE). Below this, there's a section titled '2 Services' with a filter input and a table listing two services: Dev1 and Dev2, both pointing to Artifactory URLs and running version 5.6.0.

Editing and Deleting a Service

Editing and deleting a service can be done from the Actions menu in the single service page, or by hovering over the service from the Services module.

The screenshot shows the JFrog Mission Control interface for the 'Denver' service. The service details include its version (5.5.2), URL, and status (ONLINE). An error message is displayed: 'The license provided is duplicated.' A dropdown menu is open under the 'Actions' link, showing options: Edit, Upload Extensions, and Remove. Below the service details, there's a 'Storage Summary' section with a progress bar showing 'Used: 20.7 GB / 983.3 GB (2.1%)'. At the bottom, there's a '3 Repositories' section with a table listing repository keys, types, package types, percentages, used space, files, folders, and items.

Repository Key	Repository Type	Package Type	Percentage	Used Space	Files	Folders	Items
TOTAL	N/A	N/A	100%	1.03 MB	7	1	8
Trash Can	N/A	Trash	0%	0 bytes	0	0	0
artifactory-rpm	LOCAL	RPM	97.52%	1.00 MB	6	1	7
artifactory-generic	LOCAL	Generic	2.48%	26.17 KB	1	0	1
example-repo-local	LOCAL	Generic	0%	0 bytes	0	0	0

Troubleshooting

This section describes the possible actions required for services that were added to Mission Control with errors.

Error Description	Action Required
License invalid / duplicate / expired	The license provided is either invalid, duplicated or has expired. Contact JFrog Sales at sales@jfrog.com for an additional license.
Failed to upload extensions	Update extensions by selecting "Upload Extensions" from the Actions menu in the Single Service page .
User is not an admin	Only admin users can set up a connection to a service. Insert service admin credentials.
Failed to connect to the service.	Verify the service information or the service credentials.



Managing Licenses

Overview

The Mission Control **Licenses** sub-module manages activation, renewal and upgrade of all Artifactory services under your control. There are two ways to manage Services' licenses:

- Updating individual licenses manually
- Managing multiple licenses automatically with License Buckets

Updating Licenses Individually

You may update the license of several Artifactory services at a time by matching the service with its corresponding license, and applying all the licenses in a single bulk operation. To enter or update an Artifactory license, in the **Admin** module, select **Licenses | Update license**. The process is fully described in [Updating Licenses](#) below.

Using License Buckets

Using license buckets, you can automate, and therefore greatly simplify, managing licenses for a large number of Artifactory services. The process is fully described in [License Bucket Management](#) below.

Updating Licenses

The process of applying licenses has four steps:

1. [Select Artifactory Services](#)
2. [Add Licenses](#)
3. [Match Licenses to Services](#)
4. [Apply Licenses](#)

Select Artifactory Services

Select the one or more service for which you want to add or update a license, and click "Continue" to move on to the next step

Page contents

- [Overview](#)
 - [Updating Licenses Individually](#)
 - [Using License Buckets](#)
- [Updating Licenses](#)
 - [Select Artifactory Services](#)
 - [Add Licenses](#)
 - [Match Licenses to Services](#)
 - [Apply Licenses](#)
- [License Bucket Management](#)
 - [Initial Setup](#)
 - [Installing JFrog CLI](#)
 - [Obtaining a Bucket of Licenses](#)
 - [Adding a License Bucket](#)
 - [Attaching and Detaching Licenses](#)
 - [Working with Artifactory HA](#)
 - [Removing a License Bucket](#)
 - [Bucket Report](#)

Update License



Filter...

☒ ☐ Amsterdam

☐ ☐ Denver

Cancel

< Back

Continue >

Execute All

Add Licenses

Enter the different licenses you want to apply to your Artifactory services.

Simply paste the licenses you want to add into the **Add Licenses** entry field and click "Add Licenses". Click "Continue" to move on to the next step.



Adding multiple licenses at once

You can add licenses one at a time, or paste several licenses, separated by an empty line, into the **Add Licenses** field and add them to the list at once.

Update License

Update License

Match Licenses to Services

In this step, you need to match up the Artifactory services you have selected with the licenses you have added.

First select an Artifactory service. This will enable the list of licenses.

Then, select the license you wish to apply to the selected service and click "Add".

The column on the right displays your matched selections.

To change services and licenses matches, click **Delete** to return the service and license of a matched set to their respective lists.

Once you have matched all of your Artifactory services with their licenses, click "Continue" to move on to the next step.

Update License

Select Artifactory

Add Licenses

Match Licenses

Summary

Select Artifactory

Apply License

Applied Licenses

Amsterdam

Valid Through:
Oct 9, 2018

Type:
ENTERPRISE

Owner:
JFrog

Valid Through:
Oct 9, 2018

Type:
ENTERPRISE

Owner:
JFrog

Add

Cancel

< Back

Continue >

Execute All

Update License

Select Artifactory

Add Licenses

Match Licenses

Summary

Select Artifactory

Apply License

Applied Licenses

Add

Amsterdam

Valid Through:
Oct 9, 2018

Type:
ENTERPRISE

Owner:
JFrog

Valid Through:
Oct 9, 2018

Type:
ENTERPRISE

Owner:
JFrog

Delete

Cancel

< Back

Continue >

Execute All

Apply Licenses

The **Summary** screen displays the Artifactory services that should be updated with the licenses you selected for them.

Wrong screenshot

Update License

Select Artifacts

Add Licenses

Match Licenses

Summary

Filter...

☒ Amsterdam

☐ Denver

Cancel

< Back

Continue >

Execute All

To apply the licenses click "Execute All".



You can retry, but you cannot roll back.

If, for any reason, applying a license does not work in any of the Artifactory services, you can click the corresponding "Retry" button to try again.

However, once a license is applied, you cannot roll back to the previous licenses.

License Bucket Management

JFrog Mission Control uses License Bucket Management which automates, and therefore, greatly simplifies the management of licenses for large numbers of Artifactory services.

JFrog Mission Control offers a convenient way to attach licenses to Artifactory services it manages through the Admin module under **Licenses | Update License**. This works well when managing several services of Artifactory, but may become more challenging as enterprises grow and start managing larger numbers of Artifactory services.



Getting a License Bucket

Contact JFrog Sales at sales@jfrog.com for additional information on how you can get your License Bucket.

Initial Setup

To use JFrog Mission Control license bucket management, you first need to set up the following components on your system:

- JFrog Command Line Interface (CLI)

Installing JFrog CLI

JFrog CLI is a compact and smart client that provides a simple interface to JFrog Mission Control (through its REST API). JFrog CLI can be used by your automation environment for fully automated license management for all your Artifactory services through Mission Control. For details on downloading and installing JFrog CLI, please refer to the [JFrog CLI](#) documentation.

Obtaining a Bucket of Licenses

To obtain your bucket of licenses, please contact your JFrog representative who will create a bucket with the number of licenses you require and send them to you. All licenses in a bucket have exactly the same activation parameters. That means they will all be **enterprise** or **trial** licenses with the same date of expiry.

You will receive your license bucket through two email messages. The first message includes a signed URL that points to the archive containing the bucket of licenses. You should look out for a section similar to the below:

...

Here are the **identifier** and **signed URL** for this license bucket:

Bucket identifier: <bucket_id>

Signed URL: <signed_URL>

...

The second message includes a decryption key for the license bucket. You should look out for a section similar to the below:

...

Here are the **identifier** and **Key** for this license bucket:

Bucket identifier: <bucket_id>

Key: <key>

...

You need to enter the **Bucket Identifier**, **Signed URL** and **Key** in JFrog Mission Control as described in the next section.

Adding a License Bucket

You can view all loaded license buckets in the **Admin** module under **Licenses | Bucket Management**.

License Bucket

⊕ Add new bucket

1 Bucket

Filter by Bucket Name or Bucket ID

< Page 1 of 1 >

Bucket Name	Bucket ID	License Type	Issue Date	Expiry Date	Used / Bucket Size	Customer Name
New-Bucket	415921223	ENTERPRISE	2017-10-30	2018-10-30	0 / 5	JFrog

To load a new bucket, click **Add New Bucket**.

You can give the bucket a logical name in the **Name** field. Enter the signed **URL** and **Key** you received from your JFrog representative and click **Get Bucket**.

License Bucket

New Bucket

Name *

URL *

Key *

Cancel

Get Bucket



Match the bucket identifier to the URL and key

If you have received more than one license bucket, make sure you enter a URL and Key that correspond to the same Bucket Identifier.



Bucket name and identifier are two different things

Don't confuse the bucket Name with the Bucket Identifier. The bucket name is an arbitrary logical name you assign to the bucket. The identifier is a unique identifier assigned to the bucket by JFrog.

Once the license bucket has been loaded into JFrog Mission Control, you can attach licenses from it using JFrog CLI.

Attaching and Detaching Licenses

While each Artifactory license can only activate one Artifactory service at a time, you are free to move any license around to different Artifactory services as long as the license remains valid. For example, you can temporarily attach a license from a bucket to an Artifactory service used for a specific development project. Once the project is complete and the Artifactory service is no longer needed, you can detach the license and return it to the bucket managed by Mission Control.

You can attach and detach licenses with the JFrog CLI.

The `attach-lic` command extracts a license from the bucket specified and attaches it to the specified Artifactory service.

The `detach-lic` command removes the specified license from the specified Artifactory service and returns it to the specified bucket in JFrog Mission Control.



A license always belongs to the same bucket

When you detach a license, it is always returned to the same bucket it was attached from.

For details on using the JFrog CLI to [attach](#) and [detach](#) licenses, please refer to the [JFrog CLI](#) documentation.

Working with Artifactory HA

From version 5.0 of JFrog Artifactory, licensing for all cluster nodes in an Artifactory High Availability configuration are managed through the [Cluster License Manager](#). When attaching or detaching licenses to an HA cluster, Mission Control will detect the Artifactory version and work accordingly with the Cluster License Manager if the version is 5.0 or above, or directly with each node for earlier versions of Artifactory.

Removing a License Bucket

You can view all loaded license buckets in the **Admin** module under **Licenses | Bucket Management**.

To remove a bucket, hover over it and select delete. **Removing the bucket will not remove the licenses from the services.**

Once the license bucket is removed, the services that were attached to it should be assigned to a new bucket.

License Bucket

[+ Add New Bucket](#)

1 Bucket

Filter by Bucket Name or Bucket ID

< Page 1 of 1 >

Bucket Name	Bucket ID	License Type	Issue Date	Expiry Date	Used / Bucket Size	Customer Name	
Bucket2	512583504	ENTERPRISE	2017-12-24	2018-12-24	2 / 5	Jfrog TEST	<div><div>ⓧ</div><div>Delete</div></div>

Bucket Report

A bucket report provides a variety of information on usage of licenses in the bucket. To view a bucket report, click the **Bucket ID** in the list of license buckets.

To detach a license key, right click on it and select "Detach".

License Bucket

Bucket Report: New-Bucket

Bucket identifier: 415921223

Max consumed licenses: 1


Current licenses in use: 1

Current available licenses: 4

Total Licenses: 5

Filter by Artifactory Identifier

< Page 1 of 1 >

License Key	Artifactory Identifier
	Batel : secondary

Close

Bucket identifier	The identifier of this bucket
Max. consumed licenses	The maximum number of licenses that were ever in use concurrently during the validity period of this bucket
Current licenses in use	The number of licenses currently in use
Current available licenses	The number of licenses currently available

Total licenses	The total number of licenses in this bucket
----------------	---



Do you have enough or too many licenses in your bucket?

The value of **Max. Consumed Licenses** shows you that highest number licenses you ever used during the validity period of this bucket. This is something to consider when you renew your JFrog Artifactory licenses. If you never used all the licenses in this bucket concurrently, you may be able to manage with fewer licenses. If you did reach the total number of licenses in this bucket, you may not be meeting the demand for Artifactory services in your organization and should consider purchasing more Artifactory licenses.




Notifications

Overview

Mission Control can issue email notifications for different events that occur. Notifications are generated by defining **Policies** that specify thresholds for values that will trigger the email to be sent.

Currently, events related to how instances and repositories use storage are supported as described in the following sections.

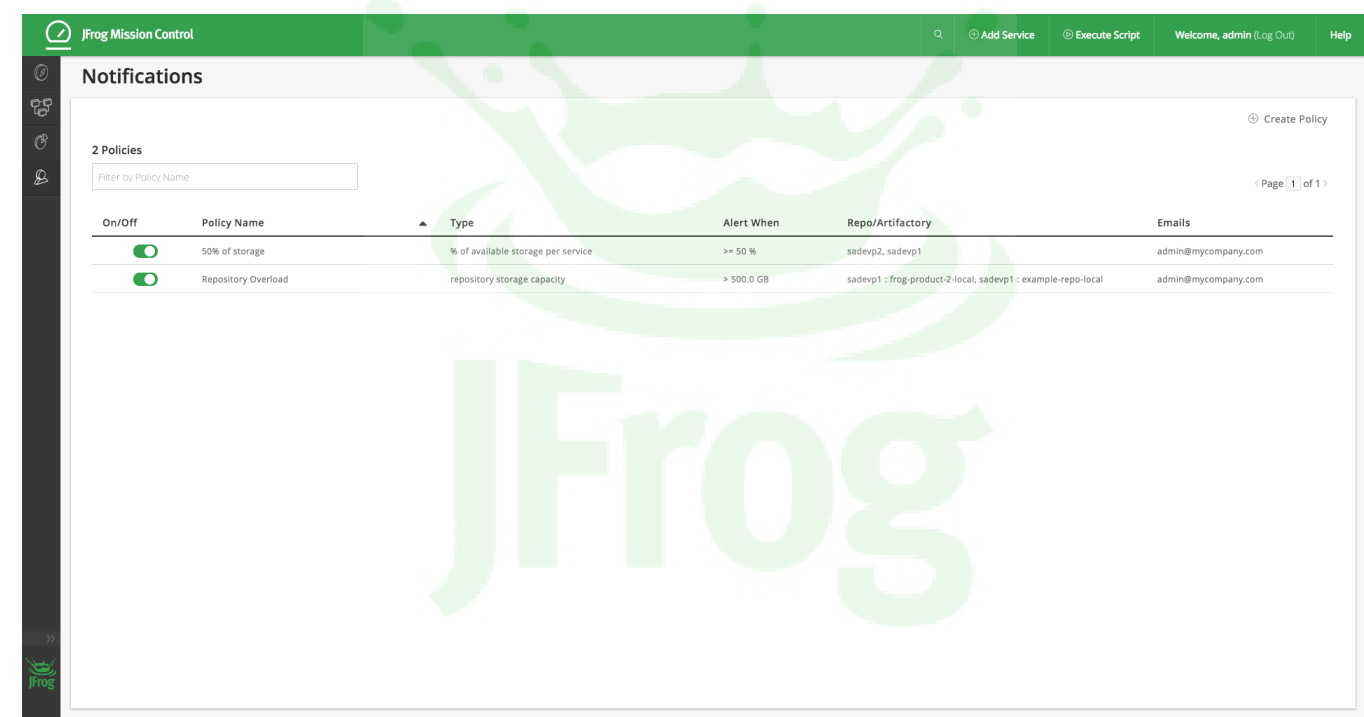
 **Configure an email server**
To allow Mission Control to send out notifications, make sure you configure an [Email server](#)

Page contents

- [Overview](#)
- [Notifications Panel](#)
- [Creating and Editing Policies](#)
 - [Percent of available storage per instance](#)
 - [Repository Storage Capacity Policies](#)

Notifications Panel

The Notifications Panel is available in the **Admin** module under **Notifications Group | Notifications** and displays all the policies you have defined.



On/Off	Enables/disables the policy
Policy Name	A unique name for the policy
Type	There are two types of policy: <ul style="list-style-type: none">• Percent of available storage per instance• Repository storage capacity
Alert When	Specifies the threshold of the policy
Repo/Artifactory	The Artifactory instance or set of repositories on which the policy is applied
Emails	The email address to which notifications will be sent

Creating and Editing Policies

To create a new policy, click the "Create Policy" button, or select an existing policy from the list to edit it.

Percent of available storage per instance



Not available when using cloud a storage provider

You cannot define a policy based on percentage of available storage per instance if you are using a cloud storage provider (e.g. S3, GCP, Microsoft Azure or others)

You can configure an instance policy to send notifications when an instance's **percentage of available storage goes above or below the set threshold**.

To create an instance policy, select **percentage of available storage per instance** in the **Policy Type** field.

Name	A logical name for the policy.
Policy Type	Specifies the policy type - Percent of available storage per instance or Repository storage capacity.
Select Instances	Select the instances to which the policy should apply.
Add Emails	Add email addresses to which notifications should be sent by entering them in this field and clicking the plus sign.
Select Policy	Specify the threshold. Select the comparator (<, <=, >, >=) and the value for the policy.

Repository Storage Capacity Policies

You can configure a repository storage capacity policy to send notifications when a repository's **usage of storage goes above or below the set threshold**.

Select **Repository Storage Capacity** in the **Policy Type** field.

JFrog Mission Control
Add Service
Execute Script
Welcome, admin (Log Out)
Help

Edit Repository Notification

Policy Settings

Name *
Repository Overload

Policy Type
repository storage capacity

Select Repositories

Filter...

sadevp1 : frog-legs-local
sadevp1 : frog-legs-sadevp2
sadevp1 : frog-product-2-assanbox1
sadevp1 : frog-product-2-euchub1
sadevp1 : frog-product-2-eudev1
sadevp1 : frog-product-2-nadevp1
sadevp1 : jump-frog-102
sadevp1 : project1-dev-local-debian-dev-local

Filter...

sadevp1 : example-repo-local
sadevp1 : frog-product-2-local

Add Emails

New Value

admin@mycompany.com

Select Policy

>
500
GB

Cancel
Save

Name	A logical name for the policy.
Policy Type	Specifies the policy type - Percent of available storage per instance or Repository storage capacity.
Select Repositories	Select the repositories to which the policy should apply.
Add Emails	Add email addresses to which notifications should be sent by entering them in this field and clicking the plus sign.
Select Policy	Specify the threshold. Select the comparator (<, <=, >, >=), the value and the units (MB, GB) for the policy.

Graphs

Overview

Mission Control stores historical data about Artifactory services and their repositories, and displays a variety of graphs showing different parameters related to storage and usage of Artifacts in Artifactory services that are managed by Mission Control.



Migrating Data from JFMC 1.x to 2.x

From version 2.0, Mission Control data is stored in Elasticsearch. If you still want your historical data to be accessible, you need to [migrate your data from the previously used InfluxDB](#) as described on this page. However, if you are not interested in your historical data, there is no need to migrate your data. Mission Control will continue to collect data from the installation of version 2.0.

Page contents

- [Overview](#)
- [Storing Graph Data](#)
- [Usage Graphs](#)
 - [Storage / Artifact Usage](#)
 - [Focusing on Artifactory Services and Repositories](#)
 - [Top 5 Repositories \(Max. per Week\)](#)
 - [Top 5 Services \(Max. per Week\)](#)

Storing Graph Data

Data for the Graphs module is stored using an [Elasticsearch](#) database. Mission control runs a service, which collects, stores and manages the historical data in Elasticsearch, and is fully controlled by Mission Control. You should be aware Elasticsearch service is running in the background, however, there should be no need to interact with it directly since it is fully managed by Mission Control.

For full details on using the ElasticSearch database with JFrog Mission Control, please refer to the [Elasticsearch Usage Guide](#).

Usage Graphs

The following graphs are currently available:

- [Storage / Artifact Usage](#) - shows the amount of storage and number of artifacts used by any or all of the services managed by Mission Control
- [Top 5 Repositories \(Max. per Week\)](#) - shows the five repositories whose maximum usage of storage has been highest on a weekly basis for the selected service
- [Top 5 Services \(Max. per Week\)](#) - shows the five services whose maximum usage of storage has been highest on a weekly basis

You can view these graphs in the **Graphs** module by selecting the corresponding tab.

Storage / Artifact Usage	Top 5 Repositories (Max. per Week)	Top 5 Instances (Max. per Week)
--	--	---

Storage / Artifact Usage

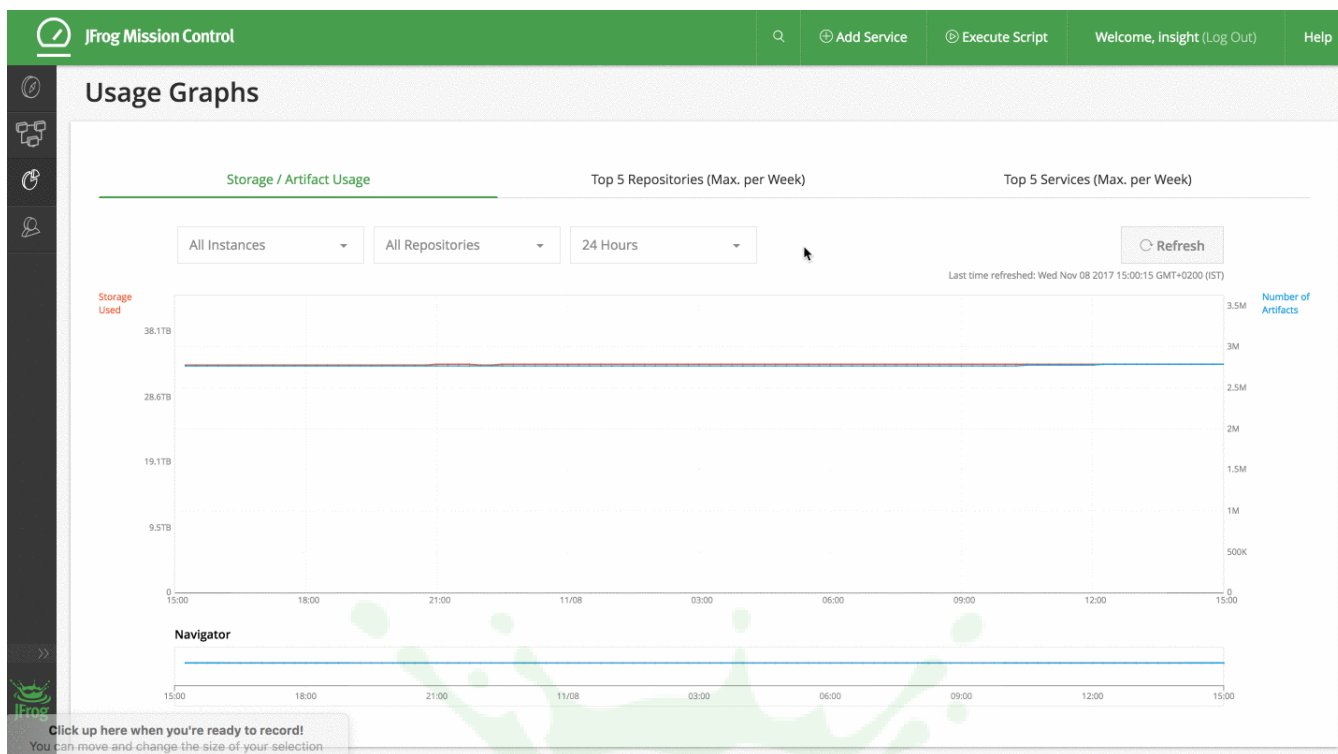
This graph displays the amount of storage used, and the number of artifacts, in the services and repositories selected in the corresponding lists. There are three available filters: **services**, **repositories**, and **time period**.



Data and services list

After a service is configured in Mission Control, data is captured every 15 minutes of the hour. The graphs will show "No data available yet". The services list will also be empty and will be populated when the first 15 minute interval of data collection window occurs.

Use the navigation bar to zoom in and navigate within the timeline. The blank spaces in the graph demonstrate no data collected, where the Artifactory services were unavailable.



Focusing on Artifactory Services and Repositories

By default, this graph displays readings that are accumulated for all Artifactory services managed by Mission Control, and all their respective repositories. You can, however, focus on any specific service or repository by making the appropriate selection in the corresponding lists, and filter it by the time period as needed.



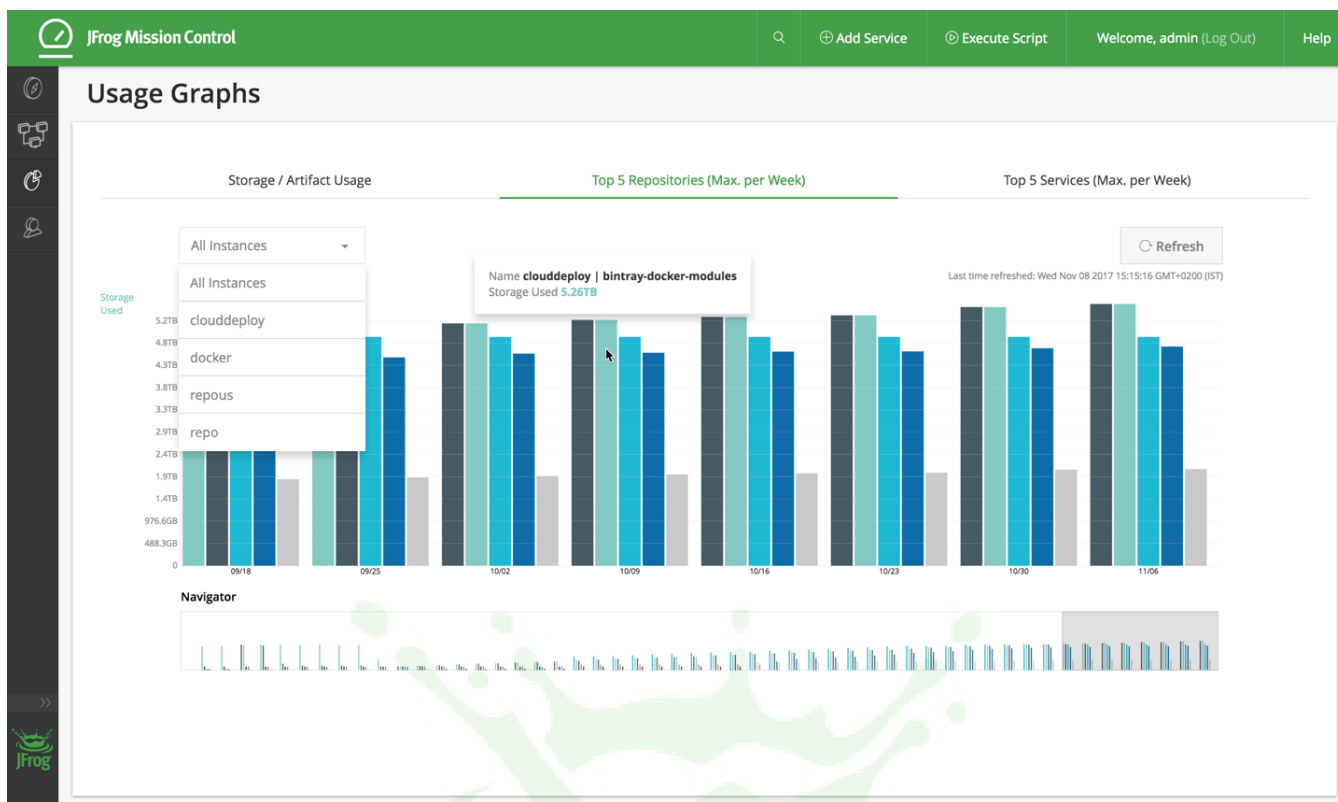
Re-adding services with same name

If a service is deleted, the historical data for the deleted service is not deleted. If a service with the same name is added, the deleted service will be renamed as `<service_name>_old_1` for the first time and `<service_name>_old_2` the second time and so on. The services drop down will show the deleted ones to help with historical debugging.

Top 5 Repositories (Max. per Week)

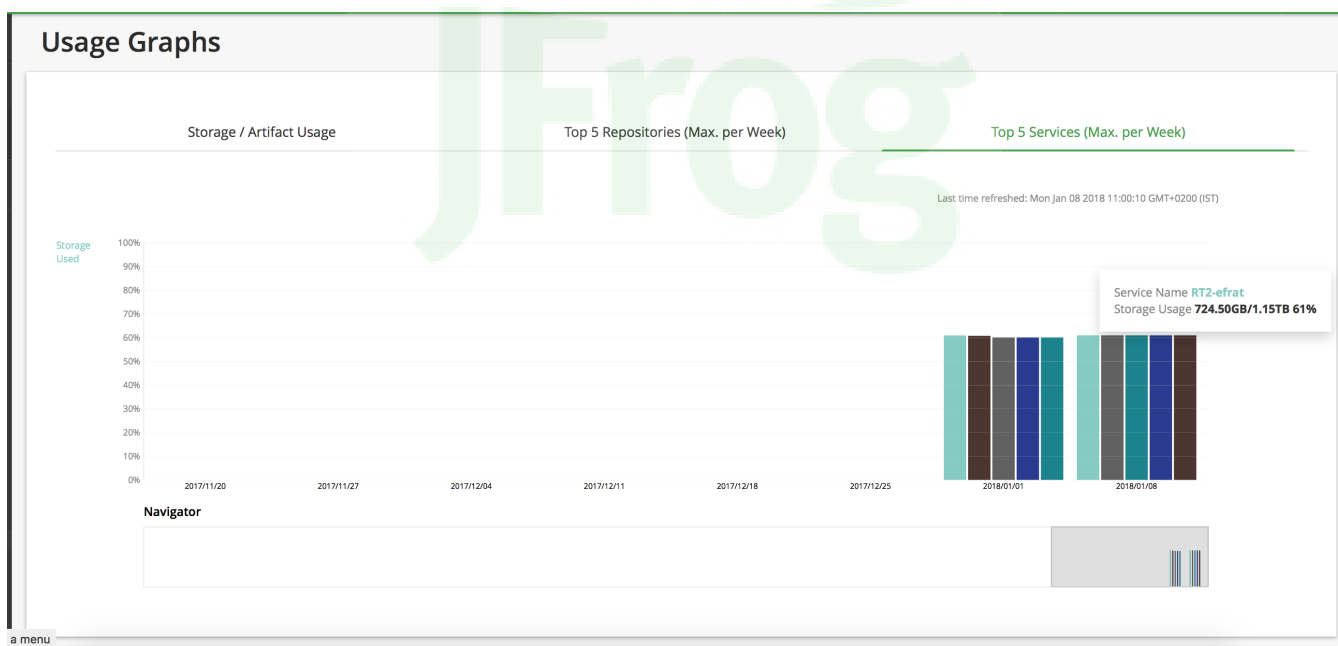
This graph displays the 5 repositories (in all managed Artifactory services) whose maximal storage used over the selected week is greatest. If any repository appears in more than one time period, it is displayed using the same color which makes it convenient to compare that repository's maximal storage used week over week. Selecting any bar or hovering over it displays additional details, including the Artifactory service and repository name.

By default, this graph displays readings that are accumulated for all services managed by Mission Control, for the current week. You can, however, focus on any specific service by making the appropriate selection in the drop-list provided, and slide the navigation bar to focus on the needed time period.



Top 5 Services (Max. per Week)

This graph shows the five Artifactory services whose artifacts storage size over the selected week is greatest. Slide the navigation bar to the time period and hover over the bars to view details.



JMX MBeans

Overview

JFrog Mission Control complies with the [JMX specification](#) for MBeans allowing you to monitor a variety of different parameters of Mission Control using any JMX agent. These include:

- Connected Artifactory instances with data on the storage they consume
- Repositories within the connected instances and the storage they consume
- Replication status for each repository

The rest of this section describes how to set up and use [JConsole](#) to monitor Mission Control, however, any JMX agent should also work.

Page Contents

- [Overview](#)
- [Connecting JConsole](#)
- [Mission Control MBeans](#)

Prerequisites

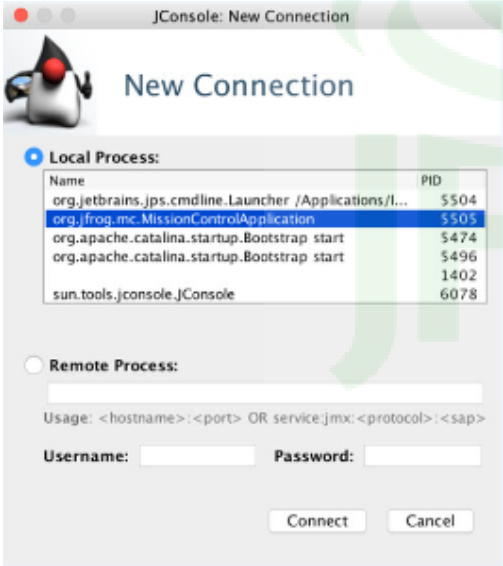
To get started, make sure that:

- Your JAVA_HOME environment variable is correctly configured
- \$JAVA_HOME/bin is registered in your operating system's "path" variable
- Mission Control is configured with one or more Artifactory instances

Connecting JConsole

You can use any JMX compliant agent to connect to the Mission Control MBeans implementation. This section shows how to connect JConsole which should be included in your JDK installation under your `$JAVA_HOME/bin` directory.

- To start up JConsole, enter `jconsole` on your command line
- Scroll the **Local Process** window until you find the **MissionControlApplication** process and click "Connect"



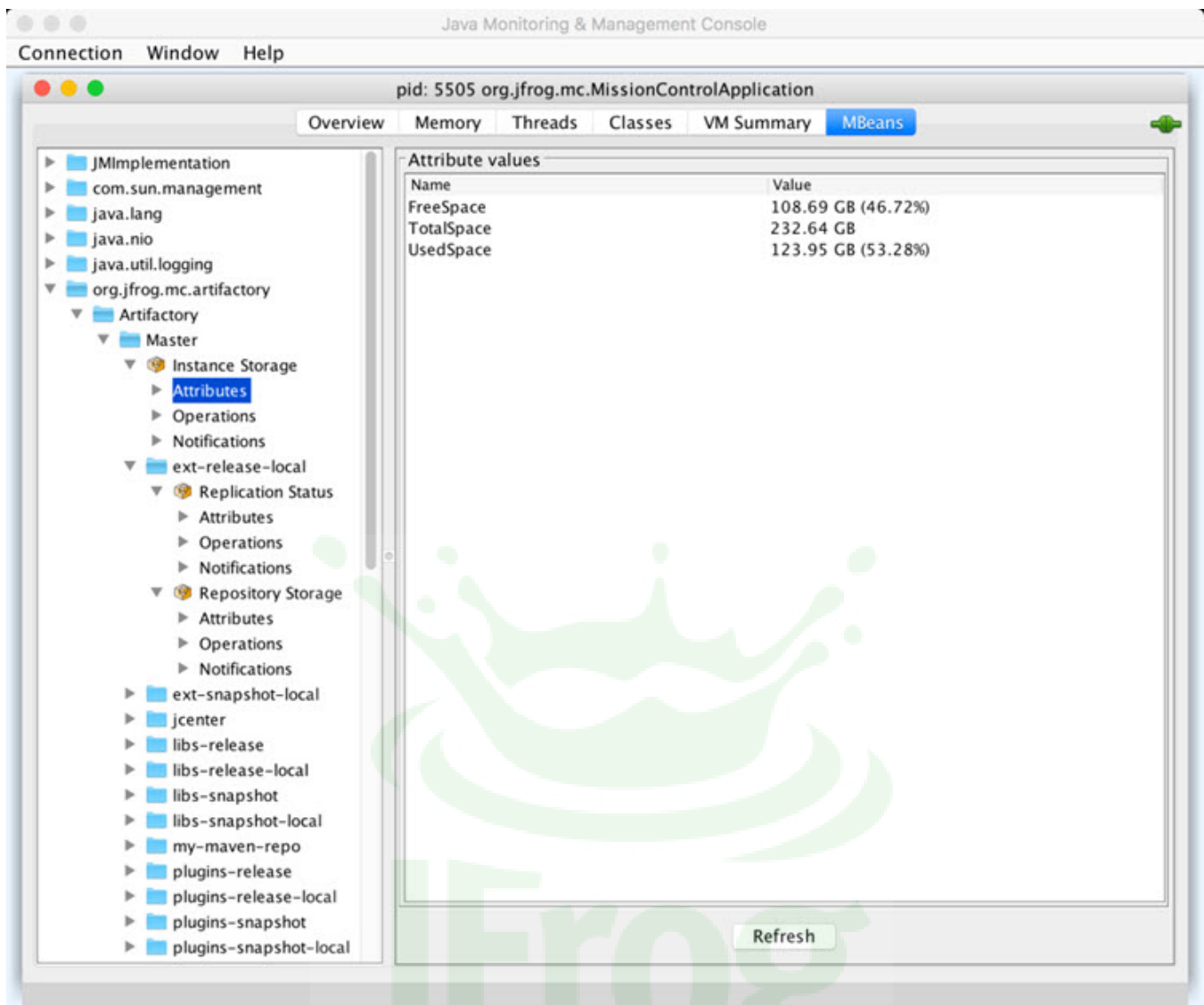
- Select the **MBeans** tab to display the MBeans hierarchy

Mission Control MBeans

Mission Control MBeans are implemented in the following hierarchy

```
org.jfrog.mc.artifactory - the root node
| | -Artifactory          -Root of all Artifactory instances
| | | -<Instances>       - Node representing each instance
| | | | -<Repositories>  - Node representing each repository in the parent instance
```

The screenshot below shows details for an Artifactory instance called "Master" and one of its repositories called "ext-release-local"



The following table describes the different MBeans implemented for Mission Control and the corresponding attributes you can monitor.

MBean	Attributes
Instance storage	<ul style="list-style-type: none"> Free space Used space Total space
Instance status	<ul style="list-style-type: none"> Started up Went down
Repository storage	<ul style="list-style-type: none"> File count Used space
Replication status	<ul style="list-style-type: none"> Last completed replication date/time Replication status

System Monitoring

Overview

Mission Control provides different facilities that allow you to maintain and monitor your system.

Page contents

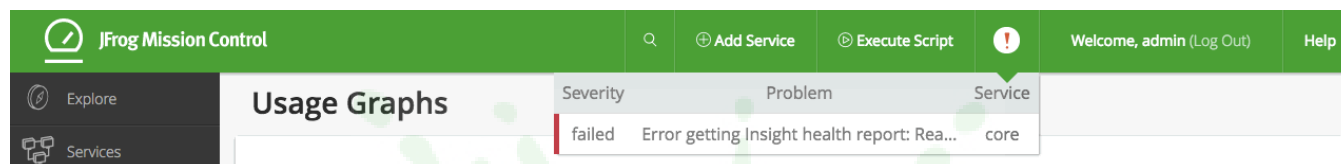
- [Overview](#)
- [System Status](#)

System Status

Mission Control displays your general system status as an icon on the ribbon. The green check mark indicates that there are no issues.



Failed service messages are displayed in the System Status with a red exclamation mark and a short description of the encountered problem.



The following table displays a sample list of failed service messages:

Service	Problem	Severity	Potential Action
core_service	MongoDB URL not configured in settings.	ERROR	Check if the Docker compose file, (Docker installations), or setenv.sh (non-Docker installations), has the appropriate user and password set for Mongo variables for the core and Mongo service, and update as necessary.
core_service	Environment variable JFI_HOME_CORE is not set. Default value is used.	WARNING	No action is required. If the default values are not working consider updating this environment variable in Docker Compose or in setenv.sh.
core_service	Settings could not be retrieved or is empty.	ERROR	Check if the Mongo service is running. If not, consider starting it.
executor_service	Executor status check failed.	ERROR	<ul style="list-style-type: none">• The Executor service is not able to communicate with the core service. Check if the Docker network set up allows communication between them• Executor service is not able to communicate with Elastic search service Check if the Docker network set up allows communication between them.
scheduler_service	Scheduler is not active.	ERROR	Scheduler cannot connect with Postgress. Check the user password set in the Docker Compose file or in setenv.sh.

Mission Control REST API

Overview

Mission Control exposes a rich REST API to allow fully automated management of Artifactory and Xray services under your control.

This provides a convenient and up-to-date self-descriptive API and can be used by various tools to automate the creation of REST calls.

Version

Mission Control REST API is currently in Version 3 and is a major upgrade which includes significant changes from the previous version.

If you are still using Version 2 of the REST API, please refer to [Mission Control REST API v2](#), however note that this has been deprecated.

We strongly recommend that you upgrade your scripts to the latest API version, and to facilitate the required modifications, please refer to [Version Mapping](#) below.



Authentication

All Mission Control REST API endpoints require basic authentication using your username and password except for the [System Health Check](#).

Usage

Mission Control REST API can be invoked in any of the standard ways for a REST API. The following section describes how to use the Mission Control REST API using cURL as an example.

Using and configuring cURL

You can download [cURL](#) here. Learn how to use and configure [cURL](#) here.

Example - Create Site

The example below demonstrates how to invoke the create user REST API.

- You have MissionControl running on your local system, on port 8080
- You wish to create a site called "us-west" containing both an Artifactory and Xray service
- You created a file with the site's parameters called **createsite.json**

To use the file to create a new user, you would use the following command:

Using cURL with the REST API

```
$ curl 'http://localhost:8080/api/v3/sites' -i -u 'admin:password' -X POST
-H 'Content-Type: application/json; charset=UTF-8' -T createsite.json
```

The file createsite.json will contain the following :

```
{
  "name": "us-west",
  "description": "US West coast site",
  "city": {
    "name": "Sunnyvale",
    "country_code": "US",
    "latitude": 37.368830,
    "longitude": -122.036350
  },
  "services": ["arti-west", "xray-west"]
}
```

Page Contents

- [Overview](#)
 - [Version](#)
 - [Usage](#)
 - [Using and configuring cURL](#)
 - [Example - Create Site](#)
- [SYSTEM](#)
 - [System Health Check](#)
- [SITES](#)
 - [Create Site](#)
 - [Update Site](#)
 - [Partial Update Site By Name](#)
 - [Get Site](#)
 - [Get Site List](#)
 - [Delete Site](#)
- [SERVICES](#)
 - [Create Service](#)
 - [Update Service](#)
 - [Get Service List](#)
 - [Delete Service](#)
 - [Get Repository List](#)
- [MONITORING](#)
 - [Get Services Status](#)
 - [Get Service Status](#)
- [DISASTER RECOVERY](#)
 - [Create DR Pair](#)
- [SCRIPTS](#)
 - [Get Scripts](#)
 - [Get Script User Input](#)
 - [Execute Script](#)
- [LICENSE BUCKETS](#)
 - [Get Bucket Status](#)
 - [Attach License](#)
 - [Attach License Artifactory 5.x HA](#)
- [AUTHENTICATION](#)
 - [Change Password](#)
- [Version Mappings](#)

Read More

- [Working with User Input](#)
- [Mission Control REST API v2](#)
- [Mission Control REST API v1](#)

SYSTEM

System Health Check

Description: Get an indication if Mission Control is running or not

Since: 2.0

Usage: GET /api/v3/ping

Example:

```
GET /api/v3/ping
```

```
true
```

SITES

These are the relevant fields when configuring sites:

Field	Type	Optional	Description
name	String	false	Site's name
description	String	true	Site's description
city	Object	false	Site's city
city.name	String	true	City's name
city.country_code	String	true	City's country code
city.longitude	Number	false	City's longitude
city.latitude	Number	false	City's latitude
services	Array	true	Names of services

Create Site

Description: Create a new site.

Since: 2.0

Security: Requires an admin user

Usage: POST /api/v3/sites

Return codes:

201 - No Content

400 - Couldn't find service(s) with following name(s); '<Service name>', '<Service name>'

409 - Name '<Site names>' already exists.

Consumes: application/json

```
{
  "name" : "<Site name>",
  "description" : "<Site description>",
  "city" : {
    "name" : "<City name>",
    "country_code" : "<Country code>",
    "latitude" : <City lat coordinate>,
    "longitude" : <City lon coordinate>
  },
  "services" : [ "<Service name>" ]
}
```

Example:

In this example, a new Sunnyvale site is created and services "arti-west" and "xray-west" are associated with the new site. If the services exist, 201 Created will be returned.

```
$ curl 'http://localhost:8080/api/v3/sites' -i -u 'admin:password' -X POST -H 'Content-Type: application
/json; charset=UTF-8' -T createsite.json
```


createsite.json

```
{
  "name": "us-west",
  "description": "US West coast site",
  "city": {
    "name": "Sunnyvale",
    "country_code": "US",
    "latitude": 37.368830,
    "longitude": -122.036350
  },
  "services": ["arti-west", "xray-west"]
}
```

Update Site

Description: Updates an exiting site by name.

Since: since 2.0

Security: Requires an admin user

Usage: PUT /api/v3/sites/{name}

Return codes:

204 - No Content

409 - The entity 'Site' with identifier '<Site-name>' was not found

Consumes: application/json

```
{
  "name" : "<Updated site name>",
  "description" : "<Updated site description>",
  "city" : {
    "name" : "<Updated city name>",
    "country_code" : "<Updated country code>",
    "latitude" : "<Updated city lat coordinate>",
    "longitude" : "<Updated city lon coordinate>"
  },
  "services" : [ {
    "name" : "<Service name>",
    "type" : "<ARTIFACTORY | XRAY>"
  } ]
}
```

Example :

In this example, an existing site named Argentina is being updated to Mexico Data Center with appropriate attributes. If the site 'Argentina' exists, *204 No Content* will be returned

```
$ curl -XPUT 'http://localhost:8080/api/v3/sites/Argentina' -i -u 'admin:password' -H 'Content-Type: application/json; charset=UTF-8' -T updatesite.json
```

updatesite.json

```
{
  "name" : "Mexico Data Center",
  "description" : "Updated site description",
  "city" : {
    "name" : "Mexico City",
    "country_code" : "MX",
    "latitude" : 19.428470,
    "longitude" : -99.127660
  }
}
```

Partial Update Site By Name

Description: Updates a site without updating the attributes.

Since: 2.1

Security: Requires an admin user

Usage: PUT /api/v3/services/{name}

Return codes:

204 - No Content

Consumes: application/json

```
PATCH /api/v3/sites/{name}
```

Example:

```
PATCH /api/v3/sites/Site%20name HTTP/1.1
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 119
```

```
{
  "name" : "Updated site name",
  "description" : "Updated site description",
  "services" : [ "Artifactory name" ]
}
```

```
HTTP/1.1 204 No Content
```

Example:

```
$ curl 'http://localhost:8080/api/v3/sites/Site%20name' -i -u 'admin:password' -X
PATCH -H 'Content-Type: application/json; charset=UTF-8' -d '{
  "name" : "Updated site name",
  "description" : "Updated site description",
  "services" : [ "Artifactory name" ]
}'
```

Get Site

Description: Gets a site by name

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/sites/{name}

Return codes:

200 - Success

409 - The entity 'Site' with identifier '{name}' was not found"

Produces: application/json

```
{
  "name" : "<Site name>",
  "description" : "<Site description>",
  "city" : {
    "name" : "<City name>",
    "country_code" : "CODE",
    "latitude" : <City lat coordinate>,
    "longitude" : <City lon coordinate>
  },
  "services" : [ {
    "name" : "<Service name>",
    "type" : "<ARTIFACTORY | XRAY>"
  } ]
}
```

Example:

In this example, information regarding Site named 'China' is being retrieved. If site 'China' exists *200 Success* will be returned

```
$ curl -XGET 'http://localhost:8080/api/v3/sites/China' -uadmin:password
```

example output

```
{
  "name": "Beijing",
  "description": "Beijing Data Center",
  "city": {
    "name": "Beijing",
    "country_code": "CN",
    "latitude": 39.907500,
    "longitude": 116.397230
  },
  "services": [
    {
      "name": "arti-beijing",
      "type": "ARTIFACTORY"
    }
  ]
}
200 Success
```

Get Site List

Description: Gets a list of all sites

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/sites/

Return codes:

200 - Success

409 - The entity 'Site' with identifier '<Site-name>' was not found

Produces: application/json

```
[{
  "name" : "<Site name>",
  "description" : "<Site description>",
  "city" : {
    "name" : "<City name>",
    "country_code" : "CODE",
    "latitude" : <City lat coordinate>,
    "longitude" : <City lon coordinate>
  },
  "services" : [ {
    "name" : "<Service name>",
    "type" : "<ARTIFACTORY | XRAY>"
  } ]
}]
```

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/sites' -uadmin:password
```

example output

```
[
  {
    "name": "China",
    "description": "",
    "city": {
      "name": "Shanghai",
      "country_code": "CN",
      "latitude": 31.22222,
      "longitude": 121.45806
    },
    "services": [
      {
        "name": "China",
        "type": "ARTIFACTORY"
      }
    ]
  },
  {
    "name": "Argentina",
    "description": "",
    "city": {
      "name": "Buenos Aires",
      "country_code": "AR",
      "latitude": -34.61315,
      "longitude": -58.37723
    },
    "services": [
      {
        "name": "Source Local",
        "type": "ARTIFACTORY"
      }
    ]
  }
]

200 Success
```

Delete Site

Description: Delete a site

Since: 2.0

Security: Requires an admin user

Usage: DELETE /api/v3/sites/{name}

Return codes:

200 - Success

409 - Cannot delete site {name}, it has non-empty service(s): {name of the service}

Example:

```
$ curl -XDELETE 'http://localhost:8080/api/v3/sites/China' -uadmin:password -H "Content-Type: application/json"
```

SERVICES

Create Service

Description: Creates a new service.

Since: 2.0

Security: Requires an admin user

Usage: POST /api/v3/services

Return codes:

201 - Created

409 - Failed to connect to the service. Please verify that the service information provided is correct

Consumes: application/json

```
{
  "name" : "<Service name>",
  "description" : "<Service description>",
  "url" : "<Service URL>",
  "username" : "<Service admin username>",
  "password" : "<Service admin password>",
  "type" : "<ARTIFACTORY | XRAY>"
}
```

Example:

```
$ curl 'http://localhost:8080/api/v3/services' -i -u 'admin:password' -X POST -H 'Content-Type: application/json; charset=UTF-8' -T createservice.json
```

createservice.json

```
{
  "name" : "dev-west",
  "description" : "Artifactory serving development in West region",
  "url" : "https://artifactory-west.acme.com/artifactory",
  "username" : "admin",
  "password" : "password",
  "type" : "ARTIFACTORY"
}
```

201 Created

Update Service

Description: Updates a service

Since: 2.0

Security: Requires an admin user

Usage: PUT /api/v3/services/{name}

Return codes:

204 - No Content

409 - Url <Service-url> already exists

Consumes: application/json

```
{
  "name" : "<Service name>",
  "description" : "<Service description>",
  "url" : "<Service URL>",
  "username" : "<Service admin username>",
  "password" : "<Service admin password>"
}
```

Example:

```
$ curl 'http://localhost:8080/api/v3/services/dev-west' -i -u 'admin:password' -X PUT -H 'Content-Type: application/json; charset=UTF-8' -T updateservice.json
```

updateservice.json

```
{
  "name" : "dev-east",
  "description" : "Artifactory serving development in East region",
  "url" : "https://artifactory-east.acme.com/artifactory",
  "username" : "admin",
  "password" : "password"
}

204 No Content
```

Get Service List

Description: Get a list of all services

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/services/

Produces: application/json

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/services' -uadmin:password
```

example output

```
[
  {
    "name": "Argentina",
    "description": "Artifactory serving development in Argentina",
    "url": "http://10.0.0.8:8082/artifactory",
    "type": "ARTIFACTORY"
  },
  {
    "name": "China",
    "description": "Artifactory serving development in China",
    "url": "http://10.0.0.8:8081/artifactory",
    "type": "ARTIFACTORY"
  }
]

200 Success
```

Delete Service

Description: Deletes a service

Since: 2.0

Security: Requires an admin user

Usage: DELETE /api/v3/services/{name}

Return codes:

200 - Success

Example:

```
$ curl -XDELETE 'http://localhost:8080/api/v3/services/Argentina' -uadmin:password -H "Content-Type: application/json"
```

Get Repository List

Description: Gets the list of repositories in the specified Artifactory service

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/services/artifactory/{name}/repositories

Return codes:

200 - Success

409 - Could not find Artifactory instance with name <Service-name>

Consumes: application/json

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/services/artifactory/{name}/repositories' -uadmin:password
```

example output

```
[
  {
    "repository_key": "bower-local",
    "description": "",
    "type": "local",
    "package_type": "bower"
  },
  {
    "repository_key": "generic-local",
    "description": "",
    "type": "local",
    "package_type": "generic"
  },
  {
    "repository_key": "libs-release-local",
    "description": "",
    "type": "local",
    "package_type": "maven"
  },
  ...
  {
    "repository_key": "npm",
    "description": "",
    "type": "virtual",
    "package_type": "npm"
  }
]
```

MONITORING

Get Services Status

Description: Get status of all services

Since: 2.0

Usage: GET /api/v3/services/monitoring/status

Return codes:

200 - Success

Produces: application/json

```
[
  {
    "service_name": "<Service name>",
    "up_time_in_sec": <Time in seconds that the service has been up>,
    "service_state": "< ONLINE | OFFLINE >"
  }
]
```

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/services/monitoring/status' -uadmin:password
```


example output

```
[
  {
    "service_name": "China",
    "up_time_in_sec": 29282,
    "service_state": "ONLINE"
  },
  {
    "service_name": "Argentina",
    "up_time_in_sec": 131,
    "service_state": "ONLINE"
  }
]

200 Success
```

Get Service Status

Description: Get status of the specified service

Since: 2.0

Usage: GET /api/v3/services/{name}/monitoring/status

Return codes:

200 - Success

409 - Could not find service with name <Service-name>

Produces: application/json

```
{
  "service_name": "<Service name>",
  "up_time_in_sec": <Time in seconds that the service has been up>,
  "service_state": "< ONLINE | OFFLINE >"
}
```

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/services/China/monitoring/status' -uadmin:password
```

example output

```
{
  "service_name": "China",
  "up_time_in_sec": 46182,
  "service_state": "ONLINE"
}

200 Success
```

DISASTER RECOVERY

Create DR Pair

Description: Matches up a Master and Target Artifactory service as a DR pair.

Since: 2.0

Security: Requires an admin user

Usage: POST /api/v3/dr-configs

Consumes: application/json

```
{
  "source" : "<Source artifactory instance>",
  "target" : "<Target artifactory instance>"
}
```

Produces: application/json

```
{
  "active": "NONE",
  "dr_replications_enabled": <true | false>,
  "state": "NONE"
}
```

Example:

```
$ curl -X POST 'http://localhost:8080/api/v3/dr-configs' -i -u 'admin:password' -H 'Content-Type: application/json; charset=UTF-8' -T createdr.json
```

createdr.json

```
{
  "source" : "Mexico",
  "target" : "China"
}
```

Example return:

Example return

```
{
  "active": "NONE",
  "dr_replications_enabled": false,
  "state": "NONE"
}
```

SCRIPTS

Get Scripts

Description: Get a list of all scripts

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/scripts

Return codes:

200 - Success

Produces: application/json

```
[
    { "name" : "<script name>" }
]
```

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/scripts' -uadmin:password
```

```
[
  {
    "name": "Create_repository"
  },
  {
    "name": "Delete_repository"
  },
  {
    "name": "ldap"
  },
  {
    "name": "Create_service"
  }
]

200 Success
```

Get Script User Input

Description: Get a list of required script user inputs

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/scripts/{name}/user_inputs

Return codes:

200 - Success

404 - The entity 'Script' with identifier '<Script-name>' was not found

Produces: application/json

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/scripts/{name}/user_inputs' -uadmin:password
```

example output

```
{
  "ArtifactoryDsl#0#LocalRepositoryDsl#0#description#0": {
    "name": "Enter the required user input value here",
    "description": "Please provide a description",
    "value": "This is a generic description",
    "type": "STRING",
    "multivalued": false
  }
}
```

Execute Script

Description: Executes the specified scripts on the specified service

Since: 2.0

Security: Requires an admin user

Usage: PUT /api/v3/execute_script/{name}

Return codes:

200 - Success

404 - Errors based on the input provided.

Consumes: application/json (only when user input is required by the script)

When the script requires user input, this is the JSON object describing it as returned by [Get Script User Input](#) endpoint.

Produces: application/json

```
[
  {
    "instance":{
      "name":"<instance name>",
      "url":"<instance URL>",
      "type":"<instance type>"
    },
    "status":"< OK | ERROR >",
    "execution_duration":<Duration in seconds>
  }
]
```

In case of error, output is:

```
[
  {
    "instance":{
      "name":"<instance name>",
      "url":"<instance URL>",
      "type":"<instance type>"
    },
    "status":"ERROR",
    "error": {
      "type": "<Error type>",
      "message" : "<Error message>",
      "details" : [ "<Additional details>" ],
    },
    "execution_duration":<Duration in seconds>
  }
]
```

Example:

```
$ curl -uadmin:password -XPUT http://localhost:8080/api/v3/execute_script/{Script_Name} -d '{}' -H 'Content-Type: application/json'
```

example output

```
[
  {
    "instance":{
      "name":"Mexico",
      "url":"http://172.31.61.159:8081/artifactory",
      "type":"ARTIFACTORY"
    },
    "status":"OK",
    "execution_duration":1622,
    "operation":"UPDATE_REPOSITORY"
  }
]
```

LICENSE BUCKETS

Get Bucket Status

Description: Get the report for a specified bucket.

Since: 2.0

Security: Requires an admin user

Usage: GET /api/v3/buckets/{identifier}/report

Produces: application/json

```
{
  "id": "<bucket ID>",
  "size": <Number of licenses in the bucket>,
  "licenses": {
    "used": <Number of licenses that are in use>,
    "available": <Number of licenses that are in available>,
    "max_used": <The maximum number of licenses ever used concurrently>
  }
}
```

Return codes:

200 - Success

Example:

```
$ curl -XGET 'http://localhost:8080/api/v3/buckets/415921223/report' -i -u 'admin:password'
```

example output

```
{
  "id": "2435",
  "size": 5,
  "licenses": {
    "used": 0,
    "available": 5,
    "max_used": 0
  }
}
```

200 Success

Attach License

Description: Attaches a license from the specified bucket to the specified Artifactory service.

Since: 2.0

Security: Requires an admin user

Usage: POST /api/v3/attach_lic/buckets/{name}

Consumes: application/json

```
{
  "node_id" : "<nodeId of the cluster node to receive the license>",
  "service_name" : "Artifactory service name",
  "deploy" : <true | false>
}
```

Produces: application/json

```
{
  "license_key" : "<license-key>"
}
```

Return codes:

200 - Success

Example:

```
$ curl -POST 'http://localhost:8080/api/v3/attach_lic/buckets/{name}' -i -u 'admin:password' -H 'Content-Type: application/json; charset=UTF-8' -T attachlicense.json
```

attachlicense.json

```
{
  "node_id" : "nodeId",
  "service_name" : "ServiceName",
  "deploy" : false
}
```

example output

```
{
  "license_key" : "<cHJVZHV...jdHM6CiAgY>"
}
```

Attach License Artifactory 5.x HA

Description: Attaches a number of licenses from the specified bucket to an Artifactory 5.x HA cluster.

Since: 2.0

Security: Requires an admin user

Usage: POST /api/v3/attach_lic/buckets/{name}

Consumes: application/json

```
{
  "number_of_licenses" : 5,
  "service_name" : "<Service name>",
  "deploy" : < false | true >
}
```

Return codes:

200 - Success

409 - "Deployment of multiple license is supported for cluster only"

Example:

```
$ curl 'http://localhost:8080/api/v3/attach_lic/buckets/{name}' -i -u 'admin:password' -X POST -H 'Content-Type: application/json; charset=UTF-8' -T attachlicense.json
```

attachlicense.json

```
{
  "number_of_licenses" : 5,
  "service_name" : "ServiceName",
  "deploy" : false
}
```

AUTHENTICATION

Change Password

Description: Changes a user's password.

Since: 2.1

Security: Users can change their own password. Requires an admin user to change all user passwords.

Usage: PUT /api/v3/auth/changePassword

Return codes:

204 - No Content

Consumes: application/json

```
PUT /api/v3/auth/changePassword HTTP/1.1
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
Content-Type: application/json; charset=UTF-8
Host: localhost:8080
Content-Length: 56

{
  "username" : "username",
  "password" : "pa$1word"
}
```

Version Mappings

To facilitate updating your scripts to use the latest API, the following table presents a mapping between endpoints in V1 and the corresponding endpoints in V2 of the REST API.

Category	Description	Method	V1 Endpoint	V2 Endpoint	V3 Endpoint
Services	Get list of Artifactory services	GET	/api/v1/instances	/api/v2/instances	N/A

	Add service	POST	/api/v1/instances	/api/v2/instances	/api/v3/services
	Update service	PUT	Not available	/api/v2/instances/{name}	/api/v3/services/{name}
	Get repositories for service	GET	/api/v1/instances/{name}/repositories	/api/v2/instances/{name}/repositories	/api/v3/services/artifactory/{name}/repositories
	Delete service by name	DELETE	/api/v1/instances/{name}	/api/v2/instances/{name}	/api/v3/services/{name}
	Get all services	GET			/api/v3/services/
Authentication	Update password	PUT	N/A	N/A	/api/v3/auth/changePassword
Security	Create user	POST	/api/v1/users	/api/v2/security/users	N/A
	Update user	PUT	/api/v1/users/{name}	/api/v2/security/users/{name}	N/A
	Create user group	POST	/api/v1/userGroups	/api/v2/security/user_groups	N/A
	Update user group	PUT	/api/v1/userGroups/{name}	/api/v2/security/user_groups/{name}	N/A
	Create permission target	POST	/api/v1/permissionTargets	/api/v2/security/permission_targets	N/A
	Update permission target by name	PUT	/api/v1/permissionTargets/{name}	/api/v2/security/permission_targets/{name}	N/A
License Buckets	Get bucket status	GET	/api/v1/buckets/{id}/status	/api/v2/buckets/{id}/report	/api/v3/buckets/{identifier}/report
	Attach a license	POST	/api/v1/buckets/{id}/licenses	/api/v2/attach_lic/buckets/{id}	/api/v3/attach_lic/buckets/{name}
	Detach a license	DELETE	/api/v1/buckets/{id}/licenses	/api/v2/detach_lic/buckets/{id}	/api/v3/attach_lic/buckets/{name}
	Attach a license to Artifactory v5.5 and above	POST			/api/v3/attach_lic/buckets/{name}
Execute Scripts	Create repository	POST	/api/v1/repositories	/api/v2/execute_scripts/repositories	N/A
	Update repository	PUT	/api/v1/repositories	/api/v2/execute_scripts/repositories	N/A
	Execute scripts on service	PUT	/api/v1/instances	/api/v2/execute_scripts/instances	/api/v3/execute_script/{name}
Scripts	Get scripts	GET	/api/v1/scripts	/api/v2/scripts	/api/v3/scripts
	Get script user inputs	GET	/api/v1/userInputs	/api/v2/scripts/user_inputs	/api/v3/scripts/{name}/user_inputs
System	System health check (ping)	GET	/api/v1/ping	/api/v2/ping	/api/v3/ping
Sites	Create Site	POST	N/A	N/A	/api/v3/sites
	Update Site	POST	N/A	N/A	/api/v3/sites/{name}
	Partial update site by Name	PUT	N/A	N/A	/api/v3/services/{name}
	Get Site	GET	N/A	N/A	/api/v3/sites/{name}
	Get Site List	GET	N/A	N/A	/api/v3/sites/
	Delete Site	DELETE	N/A	N/A	/api/v3/sites/{name}
Monitoring	Get Service Status	GET	N/A	N/A	/services/{name}/monitoring/status
	Get All services status	GET	N/A	N/A	/api/v3/services/monitoring/status
Disaster Recovery	Create DR Artifactory services pair	POST	N/A	N/A	/api/v3/dr-configs

Working with User Input

Overview

Mission Control provides you the flexibility of using placeholders in configuration scripts so that the user can enter user input when the scripts are applied.

The REST API supports this capability through the [Get Script User Input](#) REST API call.

Running a Script with User Input

Running a script with user input through the REST API is done through the following main steps:

1. Run the [Get Script User Input](#) REST API endpoint on the script.
This will return a JSON object specifying the user input required
2. Modify the **name** property of the required user input fields in the JSON object
3. Run the [Execute Script](#) REST API endpoint passing in the modified user input JSON object as input

Page Contents

- [Overview](#)
- [Running a Script with User Input](#)
- [Examples](#)
 - [Example 1 - Modifying a Repository](#)
 - [Example 2 - Selecting the Artifactory Service to Modify](#)

Examples

The following examples show how to run scripts through the REST API with user input.

Example 1 - Modifying a Repository

The following script modifies the **description** field of a local repository called "my-repository" in an Artifactory service called "Denver" according to user input. Assume the script is called **modify-description**

```
artifactory('Denver'){
    localRepository("my-repository") {
        description userInput (
            type : "STRING",
            value : "This is a generic description",
            description : "Please provide a description"
        )
    }
}
```

1. Call Get Script User Input

To get the user input required for the script, you need to make the following REST API call:

```
$ curl -XGET 'http://localhost:8080/api/v3/scripts/modify-description/user_inputs' -uadmin:password
```

This will return the following JSON object:

```
{
  "ArtifactoryDsl#0#LocalRepositoryDsl#0#description#0": {
    "name": "description",
    "description": "Please provide a description",
    "value": "This is a generic description",
    "type": "STRING",
    "multivalued": false
  }
}
```

2. Execute the script with the modified user input value

```
$ curl -XPUT 'http://localhost:8080/api/v3/execute_script/modify-description' -uadmin:password -H 'Content-Type: application/json; charset=UTF-8' -T scriptinput.json
```

where we pass in the script input JSON object as the following file:

scriptinput.json

```
{
  "ArtifactoryDsl#0#LocalRepositoryDsl#0#description#0": "*** This is my new description ***"
}
```

3. This produces the following output:

```
[
  {
    "instance": {
      "name": "Denver",
      "url": "http://artifactory-adi.jfrogdev.co/artifactory",
      "type": "ARTIFACTORY"
    },
    "status": "OK",
    "execution_duration": 514,
    "operation": "UPDATE_REPOSITORY"
  }
]
```

Example 2 - Selecting the Artifactory Service to Modify

In this example, we will modify the description of a local repository as in the previous example, however we will also receive the Artifactory service on which to make the change as user input. Assume the script is called **modify-description-select-service**

```
name = userInput (
  type : "STRING",
  value : "Insert Artifactory Name",
  description : "Please provide Service name"
)
artifactory(name){
  localRepository("my-repository") {
    description userInput (
      type : "STRING",
      value : "This is a generic description",
      description : "Please provide a description"
    )
  }
}
```

1. Call Get Script User Input

```
$ curl -XGET 'http://localhost:8080/api/v3/scripts/modify-description-select-service/user_inputs' -uadmin:password
```

This will return the following JSON object:

```
{
  "McDsl#0#name#0": {
    "name": "name",
    "description": "Please provide Service name",
    "value": "Insert Artifactory Name",
    "type": "STRING",
    "multivalued": false
  },
  "ArtifactoryDsl#0#LocalRepositoryDsl#0#description#0": {
    "name": "description",
    "description": "Please provide a description",
    "value": "This is a generic description",
    "type": "STRING",
    "multivalued": false
  }
}
```

2. Execute the script with the modified user input value

```
$ curl -XPUT 'http://localhost:8080/api/v3/execute_script/modify-description-select-service' -uadmin:password -H 'Content-Type: application/json; charset=UTF-8' -T scriptinput.json
```

where we pass in the script input JSON object as the following file:

scriptinput.json

```
{
  "McDsl#0#name#0": "Denver",
  "ArtifactoryDsl#0#LocalRepositoryDsl#0#description#0": "description for local repository"
}
```

Mission Control REST API v2

Overview

Mission Control exposes a rich REST API to allow fully automated management of Artifactory instances under your control.

Version



Deprecated

This version of Mission Control REST API v2 has been deprecated. We strongly recommend updating your scripts to the latest version. For details, please refer to [Mission Control REST API](#).

We strongly recommend that you upgrade your scripts to the latest API version, and to facilitate the required modifications, please refer to [V1 to V2 Mapping](#).

Authentication

All Mission Control REST API endpoints require basic authentication using your username and password.



Exception

The [System Health Check](#) endpoint does not require authentication.

Error Handling

Mission Control REST API returns error responses in different formats depending on where the error occurred.

Top Level Errors

Top level errors are returned if the request cannot be executed, and have the following format:

```
{
  "errors" : [
    {
      "message" : <message>,    // a descriptive error message
      "type" : <type>           // The error
    }
  ]
}
```

For example:

```
{
  "errors": [
    {
      "message": "selected script has wrong type: REPOSITORY",
      "type": "Template processing"
    }
  ]
}
```

Page Contents

- [Overview](#)
 - [Version](#)
 - [Authentication](#)
 - [Error Handling](#)
 - [Top Level Errors](#)
 - [Instance Errors](#)
 - [Repository Errors](#)
 - [Working with User Inputs](#)
 - [Mandatory Fields](#)
 - [Failure Policies](#)
 - [Supported Endpoints](#)
- [REST Resources](#)
 - [SCRIPT MAPPINGS](#)
 - [Get Script List](#)
 - [List User Inputs](#)
 - [INSTANCES](#)
 - [Get Instances](#)
 - [Add Instance](#)
 - [Update Instance](#)
 - [Delete Instance](#)
 - [Execute Scripts on Instance](#)
 - [Get All Instances Status](#)
 - [Get Instance Status](#)
 - [REPOSITORIES](#)
 - [Get Repositories](#)
 - [Create Repository](#)
 - [Update Repository](#)
 - [SECURITY](#)
 - [Create User](#)
 - [Update User](#)
 - [Create User Group](#)
 - [Update User Group](#)
 - [Create Permission Target](#)
 - [Update Permission Target](#)
 - [LICENSE BUCKETS](#)
 - [Bucket Status](#)
 - [Attach License](#)
 - [DISASTER RECOVERY](#)
 - [Create a DR Pair](#)
 - [SYSTEM](#)
 - [System Health Check](#)
- [V1 to V2 Mapping](#)

Instance Errors

Instance errors are returned for API endpoints that act on instances, such as [Create User](#), [Create User Group](#) or [Update Instance](#) and others.

```
{
  "data": [
    {
      "success": "false",                                // "false" indicates an error occurred
      "message": <message>,                             // A descriptive error message string
      "instanceName": <instance name>                   // The instance on which the error occurred
    }
  ]
}
```

For example:

```

{
  "data": [
    {
      "success": true,
      this instance succeeded //The action on
      "instanceName": "localhost:8091/artifactory"
    },
    {
      "success": false,
      this instance failed //The action on
      "message": "Connection refused",
      "instanceName": "localhost:8081/artifactory"
    }
  ]
}

```

Repository Errors

Repository errors are returned for API endpoints that act on repositories, such as [Update Repository](#) and others.

```

{
  "data": [
    {
      "success": "false",
      "message": <message>,
      "instanceName": <instance name>,
      "repositoryKey": "maven-local"
    }
  ]
}

```

// "false" indicates an error occurred
 // A descriptive error message string
 // The instance on which the error occurred
 // The repository on which the error occurred

For example:

```

{
  "data": [
    {
      "success": true,
      succeeded // This operation
      "instanceName": "localhost:8091/artifactory",
      "repositoryKey": "maven-local"
    },
    {
      "success": false,
      operation failed // This
      "message": "Connection refused",
      "instanceName": "localhost:8081/artifactory",
      "repositoryKey": "maven-local"
    }
  ]
}

```

Working with User Inputs

Mission control configuration scripts offer the flexibility of letting you provide input just before the script is applied. When working with the Mission Control UI, the User Input screen allows you to enter all user input values required for the scripts you are about to apply.

When working with the REST API, another mechanism is provided that allows you to fully automate your management of Artifactory instances using [Script Mappings](#).

For a details on how to work with script mappings and user input with the Mission Control REST API, please refer to [Working with User Input](#).

Mandatory Fields

For all endpoints, mandatory input fields are indicated by a plus sign (+).

Failure Policies

Some of the endpoints in the Mission Control REST API perform multiple operations. For example [Create Repository](#) may create several repositories on several Artifactory instances for a single API call. For [supported endpoints](#), you may specify how the system should behave if a one of the operations in a call fails by adding a "failurePolicy" tag to the JSON payload.

The options are:

Failure Policy	Description
ignore	[default] Ignore the failure and continue with the remaining operations
rollback	Roll back completed operations and cancel remaining ones
fail	Cancel all remaining operations immediately upon any failure
retry	Any failed operation should be retried once. If failure persists, all remaining operations should be cancelled

For example,

```
PUT /api/v2/security/permission_targets/{permission_target_name}
{
  "instancesIds":["art1","art2"],
  ...
  "failurePolicy" : "rollback" | "retry" | "fail" | "ignore"
}
```

Supported Endpoints

- [Create Repository](#)
- [Update Repository](#)
- [Execute Scripts on Instance](#)
- [Create User](#)
- [Update User](#)
- [Create User Group](#)
- [Update User Group](#)
- [Create Permission Target](#)
- [Update Permission Target](#)

REST Resources

JFrog Mission Control is currently in its second version which is significantly different from version 1. The following sections describe the endpoints for REST API v2

If you are still the REST API v1, please refer to [Mission Control REST API v1](#).

If you are ready to upgrade to the latest REST API (v2), please refer to [V1 to V2 Mapping](#).

SCRIPT MAPPINGS

Get Script List

Description: Gets the list of instance and repository configuration scripts available on Mission Control

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v2/scripts

Consumes: None

Produces: application/json

```

{
  "data": [
    {
      "name" : "string",
      "description" : "string",
      "target" : "INSTANCE" | "REPOSITORY"
    }
  ]
}

```

repository scripts

// The script name
// The script description
// Specifies whether this is an instance or

Example:

```

GET /api/v2/scripts
{
  "data": [
    {
      "name": "local-default",
      "target": "REPOSITORY"
    },
    {
      "name": "remote-default",
      "target": "REPOSITORY"
    },
    {
      "name": "propertySet_prodready",
      "target": "INSTANCE"
    }
  ]
}

```

List User Inputs

Description: Gets the list of Artifactory user inputs needed for scripts that are being applied

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/scripts/user_inputs

Consumes: application/json

```

{
+   "scriptMappings" : [ { //An array of scriptMapping objects
+       "instanceName" : <instance name>, // The instance on which you want to
apply a script
       "repositoryKey" : <string>, // The repository on which you want to apply the script
// Mandatory if the operationType is UPDATE_REPOSITORY.
// *** Not applicable and should be omitted *** if the operationType is
CREATE_REPOSITORY or UPDATE_INSTANCE
       "scriptNames" : [<script name>, <script name>, ...] //The names of the scripts you want
to apply
    }
  ]
  "operationType": "CREATE_REPOSITORY" | "UPDATE_REPOSITORY" | "UPDATE_INSTANCE" //The type of
operation you want to perform in the subsequent call with the user inputs returned
}

```

Produces: application/json


```

{
  "data":
  [
    {
      "success" : true,
      "instanceName" : "localhost:8091/artifactory",
      "repositoryKey" : "maven-local",          // The repository on which the script
was applied
                                                // Only present if operationType was
UPDATE_REPOSITORY.
      "scriptUserInputs" : [
        {
          "multiple" : "boolean",          // Whether this user input item
can take multiple values
          "type" : "STRING" | "BOOLEAN" | "INTEGER" | "INSTANCE" |
"REPOSITORY", // The type of this user input item
          "value" : "object",              // A default value for this
user input item
          "description" : "string", // A description for this user input
item
          "name" : "string",              // A logical name for this
user input item
          "id" : "string"                  // The identifier of this user
input item. This is the identifier that must be used in any subsequent API call
        } ]
      ]
    }
  ]
}

```

Sample usage:

Assume the following scripts have been defined:

Script Name	Script Body
local-string-userinput	<pre> localRepository('local-userInput') { description = userInput (name : "User Friendly Name", // Optional type : "STRING", // "BOOLEAN", "INTEGER", "INSTANCE", "REPOSITORY" value : "default value", description : "please provide a value") } </pre>
local-instance-userInput	<pre> test = userInput (type : "INSTANCE") localRepository('repo-variables') { description test.name } </pre>
local-repo-userInput	<pre> test = userInput (type : "REPOSITORY") localRepository('repo-variables') { description test.name } </pre>

```

POST /api/v2/scripts/user_inputs
{
  "scriptMappings": [{
    "instanceName": "Master",
    "scriptNames": ["local-string-userInput", "local-instance-userInput", "local-repo-userInput"]
  }],
  "operationType" : "CREATE_REPOSITORY"
}

Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "Master",
      "scriptUserInputs": [
        {
          "id": "RepositoryMapper#0#description#0",
          "name": "User Friendly Name",
          "description": "please provide a value",
          "value": "default value",
          "type": "STRING",
          "multiple": false
        },
        {
          "id": "TemplateExecutor#0#test#0",
          "name": "test",
          "type": "INSTANCE",
          "multiple": false
        },
        {
          "id": "TemplateExecutor#0#test#1",
          "name": "test",
          "type": "REPOSITORY",
          "multiple": false
        }
      ]
    }
  ]
}

```

INSTANCES

Get Instances

Description: Gets the list of Artifactory instances managed by Mission Control

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v2/instances

Consumes: None

Produces: application/json

```

{
  "data": [
    {
      "name" : <string>           //The Artifactory instance name
      "url"  : <string>,         //The Artifactory instance URL
    }
  ]
}

```

Example:

```
GET /api/v2/instances
{
  "data": [
    {
      "name": "DRTarget",
      "url": "http://hostname-target/artifactory"
    },
    {
      "name": "Master",
      "url": "http://hostname-master/artifactory"
    },
    {
      "name": "AOL-ClusterNNN-100",
      "url": "https://mycompany.artifactoryonline.org/mycompany"
    }
  ]
}
```

Add Instance

Description: Adds an Artifactory instance

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/instances

Consumes: application/json

```
{
  "name" : "<instance name>",
  "description": "<description text>",
  "url" : "<instance URL>",
  "username" : "<admin username>",
  "password" : "<admin password>",
  "location" : "<location text>"
}
```

Produces: application/json

Example: Add a node to an Artifactory HA cluster

```
POST /api/v2/instances
{
  "name" : "Master",
  "description": "description text",
  "url" : "http://artifactory-hostname/artifactory",
  "username" : "admin",
  "password" : "password",
  "location" : "node location"
}
```

Update Instance

Description: Updates an Artifactory instance

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v2/instances/{instance}

Consumes: application/json

```
{
  "description": "<description text>",
  "url" : "<instance URL>",
  "username" : "<admin username>",
  "password" : "<admin password>",
  "location" : "<location text>"
}
```

Produces: application/json

Example: Update Artifactory instance called "Master"

```
POST /api/v2/instances/Master
{
  "description": "new description text",
  "url" : "http://new-artifactory-hostname/artifactory",
  "username" : "new-admin",
  "password" : "new-password",
  "location" : "new node location"
}
```

Delete Instance

Description: Removes an Artifactory instance from Mission Control (doesn't do anything to the instance itself)

Since: 1.0

Security: Requires an admin user

Usage: DELETE /api/v2/instances/{instance}

Example: Delete Artifactory instance called "AOL-ClusterNNN-100"

```
DELETE /api/v2/instances/AOL-ClusterNNN-100

204
```

Execute Scripts on Instance

Description: Executes a set of scripts on a set of Artifactory instances

Since: 1.3

Security: Requires an admin user

Usage: PUT /api/v2/execute_scripts/instances

Consumes: application/json

```
{
  "scriptMappings": [
    {
      "instanceName": <Artifactory instance>,
      "scriptNames": [<script names>]
    }
  ]
}
```

Example: Apply scripts ldapSettings1 and propertySets1 to Artifactory instance called "Master".

```
{
  "scriptMappings": [
    {
      "instanceName": "Master",
      "scriptNames": [
        "ldapSettings1", "propertySets1"
      ]
    }
  ]
}
```

Get All Instances Status

Description: Gets the status of all managed Artifactory instances

Since: 1.6

Security: Requires an admin user

Usage: GET /api/v2/instances/monitoring/status

Consumes: None

Produces: application/json

```
{
  "data": [
    {
      "instanceName": <instance name>, //String
      "upTimeInSec": <Number of seconds instance has been up>, //Integer (when instanceState is ONLINE)
      "instanceState": <instance state> //String: ONLINE | OFFLINE | UNAUTHORIZED | ERROR
    }
  ]
}
```

Example

```
{
  "data": [
    {
      "instanceName": "test-instance-2",
      "instanceState": "OFFLINE"
    },
    {
      "instanceName": "test-instance-1",
      "upTimeInSec": 1744204,
      "instanceState": "ONLINE"
    }
  ]
}
```

Get Instance Status

Description: Gets the status of a specific managed Artifactory instance

Since: 1.6

Security: Requires an admin user

Usage: GET /api/v2/instances/[instance-name]/monitoring/status

Consumes: None

Produces: application/json

```
{
  "data": {
    {
      "instanceName": <instance name>, //String
      "upTimeInSec": <Number of seconds instance has been up>, //Integer (when instanceState is ONLINE)
      "instanceState": <instance state> //String: ONLINE | OFFLINE | UNAUTHORIZED | ERROR
    }
  }
}
```

Example

```
{
  "data": {
    {
      "instanceName": "test-instance-1",
      "upTimeInSec": 1744204,
      "instanceState": "ONLINE"
    }
  }
}
```

REPOSITORIES

Get Repositories

Description: Gets a list of repositories in an Artifactory instance

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v2/instances/{instance name}/repositories

Consumes: None

Produces: application/json

Example: Get a list of repositories in instance AOL-ClusterNNN-100.

```
GET /api/v2/instances/cluster-126-10100/repositories
{
  "data": [
    {
      "repositoryKey": "libs-release-local",
      "description": "",
      "type": "local",
      "packageType": "maven"
    },
    {
      "repositoryKey": "libs-snapshot-local",
      "description": "",
      "type": "local",
      "packageType": "maven"
    },
    {
      "repositoryKey": "plugins-release-local",
      "description": "Local repository for plugins",
      "type": "local",
      "packageType": "maven"
    }
  ]
}
```

Create Repository

Description: Creates a repository in a set of Artifactory instances. For more information, please refer to [Configuring Repositories](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/execute_scripts/repositories

Consumes: application/json

```
{
  "scriptMappings": [
    {
      "instanceName" : <string>,           //Artifactory instance on which to create
      "scriptNames" : [<string>],         //scripts to apply when creating the
      "scriptUserInputs":                 //user inputs required for the scripts
      {
        <userId> : <user input value> // userId obtained from
      }
    }
  ]
}
```

Produces: application/json

Example: Use the following script, to create a local repository with default values.

```

localRepository('local-default') {
    description "Public description"
    notes "Some internal notes"
    includesPattern "**/*" // default
    excludesPattern "" // default
    repoLayoutRef "maven-2-default"
    packageType "generic" // "maven" | "gradle" | "ivy" | "sbt" | ... | "generic"
    debianTrivialLayout false
    checksumPolicyType "client-checksums" // default | "server-generated-checksums"
    handleReleases true // default
    handleSnapshots true // default
    maxUniqueSnapshots 0 // default
    snapshotVersionBehavior "unique" // "non-unique" default | "deployer"
    suppressPomConsistencyChecks false // default
    blackedOut false // default
    propertySets // ([ "ps1", "ps2" ])
    archiveBrowsingEnabled false
    calculateYumMetadata false
    yumRootDepth 0
}

```

```

POST /api/v2/execute_scripts/repositories
{
    "scriptMappings": [
        {
            "instanceName": "DRTarget",
            "scriptNames": [
                "local-default"
            ]
        }
    ]
}

```

Update Repository

Description: Updates a repository in a set of Artifactory instances. For more information, please refer to [Configuring Repositories](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v2/execute_scripts/repositories

Consumes: application/json

```

{
    "scriptMappings" : [
        {
+           "instanceName" : <string>,    //Artifactory instance on which to update the
repository
+           "repositoryKey" : <string>,  //Name of the repository to update
            "scriptNames" : [<string>],  //scripts to apply when updating the repository.
The user inputs must be provided in an order that corresponds to the script names.
            "scriptUserInputs" : //user inputs required for the scripts applied
            {
                <userId> : <user input value> // string or object, depending on
the user input type
            }
        }
    ]
}

```

Produces: application/json

Example: Use the following script called local-repo-userinput to update the description field on local repository ext-release-local on instance M aster.

```
test = userInput (
  type : "REPOSITORY"
)
localRepository('repo-variables') {
  description test.name
}
```

```
PUT /api/v2/execute_scripts/repositories
{
  "scriptMappings":[
    {
      "instanceName": "Master",
      "repositoryKey": "ext-release-local",
      "scriptNames":[
        "local-repo-userInput"
      ],
      "scriptUserInputs" : {
        "TemplateExecutor#0#test#0": {"instanceName": "Master", "repositoryKey": "ext-release-
local" }
      }
    }
  ]
}

Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "Master",
      "repositoryKey": "ext-release-local"
    }
  ]
}
```

SECURITY

Create User

Description: Creates a user on a set of Artifactory instances. For more information, please refer to [Managing Users](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/security/users

Consumes: application/json

```
{
+   "instanceNames":[<string>], // The Artifactory instances on
which to create this user
+   "user" :
    {
+       "name" : <string>, // The user's name
+       "email" : <string>, // The
user's email
+       "password" : <string>, // The user's password
in clear-text
+       "admin" : <boolean>, // If true,
this is an admin user
+       "profileUpdatable" : <boolean>, // If true, this user can update
their profile
+       "internalPasswordDisabled" : <boolean> // If true, this user cannot use internal
password when external authentication (such as LDAP) is enabled.
    }
}
```


Produces: application/json

Example: Create user "johns" with the below parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```
POST /api/v2/security/users
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "user": {
    "name": "johns",
    "email": "johns@somewhere.com",
    "password": "12345678",
    "admin": false,
    "profileUpdatable": false,
    "internalPasswordDisabled": false
  }
}
```

Response body:

```
{
  "data": [
    {
      "success": true,
      "instanceName": "cluster-121-10100"
    },
    {
      "success": true,
      "instanceName": "cluster-126-10100"
    }
  ]
}
```

Update User

Description: Updates a user on a set of Artifactory instances. For more information, please refer to [Managing Users](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v2/security/users/{username}

Consumes: application/json

```
{
+   "instanceNames":<string>,           // The Artifactory instances on which to create
this user
+   user:
+   {
      "email" : <string>,                // The user's email
      "password" : <string>,            // The user's password in clear-text
      "admin" : <boolean>, // If true, this is an admin user
      "profileUpdatable" : <boolean>, // If true, this user can update their profile
      "internalPasswordDisabled" : <boolean> // If true, this user cannot use internal password
when external authentication (such as LDAP) is enabled.
    }
}
```

Produces: application/json

Example: Update the parameters of user "johns" in Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```

PUT /api/v2/security/users/johns
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "user": {
    "name" : "johns"
    "email": "johns@newdomain.com"
    "password": "changed",
    "admin": false,
    "profileUpdatable": true,
    "internalPasswordDisabled": false
  }
}

```

```

Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "cluster-121-10100"
    },
    {
      "success": true,
      "instanceName": "cluster-126-10100"
    }
  ]
}

```

Create User Group

Description: Creates a user group on a set of Artifactory instances. For more information please refer to [Creating and Editing Groups](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/security/user_groups

Consumes: application/json

```

{
+   "instanceNames":<string>],           // The Artifactory instances on which to create
this user
+   "userGroup" : {
+       "name" : <string>,                // The group's name
        "description" : <string>,        // A description for this group
        "autoJoin" : <boolean>,         // If true, new users created in the target Artifactory instance
will automatically be added to this group
        "users" : [<string>]             // The list of users (by user name) to include in this group
    }
}

```

Produces: application/json

Example: Creates a user group called "developers" along with the specified parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```
POST /api/v2/security/user_groups
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "userGroup": {
    "name": "developers",
    "description": "The developer group",
    "autoJoin": false,
    "users": ["johns", "ronaldrm"]
  }
}
```

```
Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "cluster-121-10100"
    },
    {
      "success": true,
      "instanceName": "cluster-126-10100"
    }
  ]
}
```

Update User Group

Description: Updates a user group on a set of Artifactory instances. For more information please refer to [Creating and Editing Groups](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v2/security/user_groups/{group name}

Consumes: application/json

```
{
+   "instanceNames":<string>,           // The Artifactory instances on which to update
this group
+   "userGroup:{
      "name" : <string>
      "autoJoin" : <boolean>,           // If true, new users created in the target Artifactory instance
will automatically be added to this group
      "description" : <string>           // A description for this group
      "users" : [<string>],             // The new list of users (by user name) to include in this group.
This list replaces the current set of users in the group
    }
}
```

Produces: application/json

Example: Update the "developers" user group with the specified parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```

PUT /api/v2/security/user_groups/developers
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "userGroup": {
    "name" : "developers",
    "description": "The changed developer group",
    "autoJoin": false,
    "users": []
  }
}

Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "cluster-121-10100"
    },
    {
      "success": true,
      "instanceName": "cluster-126-10100"
    }
  ]
}

```

Create Permission Target

Description: Creates a permission target on a set of Artifactory instances. For more information, please refer to [Managing Permissions](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v2/security/permission_targets

Consumes: application/json

```

{
+   "instanceNames" : [<string>],           // The Artifactory instances to which this
permission target should be applied
+   "permissionTarget" : {
+       "name" : <string>,                  // A name for this permission target
+       "repositories" : [<string>],        // The repositories to which this permission
target applies
        "anyRemote" : <boolean>,           // If true, applies to any remote repository
        "anyLocal" : <boolean>,           // If true, applies to any local repository
        "excludesPattern" : <string>,      // Excludes pattern to filter out certain
repositories
        "includesPattern" : <string>,      // Includes pattern to filter in certain
repositories
        "principals":                     // The principles to which this permission target should be applied
        {
            "users" :
            {
                <userName> : [{permission}] // The users and corresponding permissions they are
given where m=admin; d=delete; w=deploy; n=annotate; r=read
            },
            "groups" :
            {
                <groupName> : [{permission}] // The groups and corresponding permissions they
are given where m=admin; d=delete; w=deploy; n=annotate; r=read
            }
        }
    }
}

```

Produces: application/json

Example: Creates a permission target called "releasers" on repository "ext-release-local" for users "johns" and "colonels" and groups "developers" and "itmanagers" (each with corresponding permissions) on Artifactory instances "cluster-121-10100" and "cluster-126-10100".

```

POST /api/v2/security/permission_targets
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "permissionTarget": {
    "name": "releasers",
    "includesPattern": "***",
    "excludesPattern": "",
    "anyLocal": true,
    "anyRemote": false,
    "repositories": ["ext-release-local"],
    "principals": {
      "users": {
        "johns": ["r", "w", "m"],
        "colonels": ["d", "w", "n", "r"]
      },
      "groups": {
        "developers": ["m", "r", "n"],
        "itmanagers": ["r"]
      }
    }
  }
}

```

```

Response:
{
  "data": [
    {
      "success": true,
      "instanceName": "cluster-121-10100"
    },
    {
      "success": true,
      "instanceName": "cluster-126-10100"
    }
  ]
}

```

Update Permission Target

Description: Updates a permission target on a set of Artifactory instances. Any permissions previously set for users or groups are replaced with the specified permissions. For more information, please refer to [Managing Permissions](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v2/security/permission_targets/{permission target name}

Consumes: application/json

```

{
+   "instanceNames" : [<string>],           // The Artifactory instances to which this
permission target should be applied
+   "permissionTarget" :
    {
+       "repositories" : [<string>],         // The repositories to which this permission
target applies
        "anyRemote" : <boolean>,           // If true, applies to any remote repository
        "anyLocal" : <boolean>,           // If true, applies to any local repository
        "excludesPattern" : <string>,      // Excludes pattern to filter out certain
repositories
        "includesPattern" : <string>,      // Includes pattern to filter in certain
repositories
        "principals" :                    // The principles to which this permission target should be applied
        {
            "users" :
            {
                <userName> : [{permission}] // The users and corresponding permissions they are
given where m=admin; d=delete; w=deploy; n=annotate; r=read
            },
            "groups" :
            {
                <groupName> : [{permission}] // The groups and corresponding permissions they
are given where m=admin; d=delete; w=deploy; n=annotate; r=read
            }
        }
    }
}

```

Produces: application/json

Example: Updates the permission target called "releasers" on repository "ext-release-local" for "itmanagers" awarding them full permissions on any remote repository on Artifactory instances "cluster-121-10100" and "cluster-126-10100".

```

PUT /api/v2/permissionTargets/releasers
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "permissionTarget" : {
    "anyRemote" : true,
    "repositories" : ["ext-release-local"],
    "principals": {
      "users" : {
        "colonels" : ["d", "w", "n", "r"]
      },
      "groups" : {
        "itmanagers" : ["d", "w", "n", "r"]
      }
    }
  }
}

```

LICENSE BUCKETS

Bucket Status

Description: Gets the status of the specified license bucket .

Since: 1.3

Security: None

Usage: GET /api/v2/buckets/{bucket-name}/report

Consumes: None

Produces: application/json

```
{
  "data": {
    "id": <bucket-id>,
    "size": <number of licenses in the bucket>,
    "licenses": {
      "used": <number of licenses currently being used>,
      "available": <number of licenses currently available>,
      "maxUsed": <max number of licenses that were ever used concurrently>,
    }
  }
}
```

Sample usage: Get the status of bucket with ID abcdefg

```
GET /api/v2/buckets/abcdefg/report
{
  "data": {
    "id": "abcdefg",
    "size": 14,
    "licenses": {
      "used": 1,
      "available": 13,
      "maxUsed": 1
    }
  }
}
```

Attach License

Description: Attaches a license from the specified bucket, or a number of licenses to an Artifactory 5.x HA cluster.

Since: 1.3

Security: None

Usage: POST /api/v2/attach_lic/buckets/{bucket-name}

Consumes: application/json

```
{
  "instanceName" : <Artifactory instance to which the license should be attached>,
  "deploy" : true <If true, the license is actually deployed to the instance>,
  "numberOfLicenses" : < number of licenses to deploy to an Artifactory 5.x HA cluster. Optional, default
value is 1>,
}
```

Sample usage:

```
POST /api/v2/attach_lic/buckets/abcdefg
{
  "instanceName" : "Master",
  "deploy" : true,
  "numberOfLicenses" : 7
}

Response:
{
  "data": {
    "success": true,
    "message": "License deployed to instance Master"
  }
}
```

Detach License

Description: Detaches a license from an Artifactory instance and returns it to the specified bucket

Since: 1.3

Security: None

Usage: DELETE /api/v2/detach_lic/buckets/{bucket-name}

Consumes: application/json

```
{
  "InstanceName" : <Artifactory instance from which to detach the license>
}
```

Sample usage:

```
DELETE /api/v2/detach_lic/buckets/abcdefg
```

```
{
  "InstanceName" : "Master"
}
```

Response (success)

2014

Response (error - instance is online)

```
{
  "errors": [
    {
      "type": "Exception",
      "message": "Instance `Master' is still using the license."
    }
  ]
}
```

DISASTER RECOVERY

Create a DR Pair

Description: Matches up a Master and Target instance as a DR pair.

Since: 1.5

Security: None

Usage: POST /api/v2/dr-configs

Consumes: application/json

```
{
  "source": "master_instance_name",
  "target": "target_instance_name"
}
```

Sample usage:


```
POST /api/v2/dr-configs
{
  "source": "corp-west",
  "target": "corp-west-dr"
}
```

```
Response:
{
  "data": {
    "id": "0be405a8-2713-4ec6-a775-d34072e1b2d5",
    "sourceId": "276dd14f-8579-4f64-967e-46214fc7eafe",
    "targetId": "dac1f570-096d-4104-9b06-881588e0adc0",
    "active": "NONE",
    "drReplicationsEnabled": false,
    "state": "NONE"
  }
}
```

SYSTEM

System Health Check

Description: Simple ping to Mission Control to see if it is running.

Since: 1.0

Security: None

Usage: GET /api/v2/ping

Consumes: None

Produces: application/json

```
{
  "data" : true
}
```

V1 to V2 Mapping

To facilitate updating your scripts to use the latest API, the following table presents a mapping between endpoints in V1 and the corresponding endpoints in V2 of the REST API.

Category	Description	Method	V1 Endpoint	V2 Endpoint
Instances	Get list of artifactory instances	GET	/api/v1/instances	/api/v2/instances
	Add instance	POST	/api/v1/instances	/api/v2/instances
	Update Instance	PUT	Not available	/api/v2/instances/{name}
	Get repositories for instance	GET	/api/v1/instances/{name}/repositories	/api/v2/instances/{name}/repositories
	Delete instance by name	DELETE	/api/v1/instances/{name}	/api/v2/instances/{name}
Security	Create user	POST	/api/v1/users	/api/v2/security/users
	Update user	PUT	/api/v1/users/{name}	/api/v2/security/users/{name}
	Create user group	POST	/api/v1/userGroups	/api/v2/security/user_groups
	Update user group	PUT	/api/v1/userGroups/{name}	/api/v2/security/user_groups/{name}
	Create permission target	POST	/api/v1/permissionTargets	/api/v2/security/permission_targets
	Update permission target by name	PUT	/api/v1/permissionTargets/{name}	/api/v2/security/permission_targets/{name}
License Buckets	Get bucket status	GET	/api/v1/buckets/{id}/status	/api/v2/buckets/{id}/report
	Attach a license	POST	/api/v1/buckets/{id}/licenses	/api/v2/attach_lic/buckets/{id}
	Detach a license	DELETE	/api/v1/buckets/{id}/licenses	/api/v2/detach_lic/buckets/{id}
Execute Scripts	Create repository	POST	/api/v1/repositories	/api/v2/execute_scripts/repositories
	Update repository	PUT	/api/v1/repositories	/api/v2/execute_scripts/repositories
	Execute scripts on instance	PUT	/api/v1/instances	/api/v2/execute_scripts/instances

Scripts	Get scripts	GET	/api/v1/scripts	/api/v2/scripts
	Get script user inputs	GET	/api/v1/userInputs	/api/v2/scripts/user_inputs
System	System health check (ping)	GET	/api/v1/ping	/api/v2/ping



Mission Control REST API v1

Overview

Mission Control exposes a rich REST API to allow fully automated management of Artifactory instances under your control.

Version



Deprecated

This version of Mission Control REST API has been deprecated. We strongly recommend updating your scripts to the latest version. For details, please refer to [Mission Control REST API](#).

Authentication

All Mission Control REST API endpoints require authentication using user/password.



Exception

The [System Health Check](#) endpoint does not require authentication.

Error Handling

Mission Control REST API returns error responses in different formats depending on where the error occurred.

Top Level Errors

Top level errors are returned if the request cannot be executed, and have the following format:

```
{
  "errors" : [
    {
      "message" : <message>,    // a descriptive error message
      "type" : <type>           // The error
    }
  ]
}
```

For example:

```
{
  "errors": [
    {
      "message": "selected script has wrong type: REPOSITORY",
      "type": "Template processing"
    }
  ]
}
```

Page Contents

- [Overview](#)
- [Version](#)
 - [Error Handling](#)
 - [Top Level Errors](#)
 - [Instance Errors](#)
 - [Repository Errors](#)
 - [Working with User Inputs](#)
 - [Mandatory Fields](#)
- [REST Resources](#)
 - [SCRIPT MAPPINGS](#)
 - [Get Script List](#)
 - [List User Inputs](#)
 - [INSTANCES](#)
 - [Get Instances](#)
 - [Update Instance](#)
 - [REPOSITORIES](#)
 - [Get Repositories](#)
 - [Create Repository](#)
 - [Update Repository](#)
 - [SECURITY](#)
 - [Create User](#)
 - [Update User](#)
 - [Create User Group](#)
 - [Update User Group](#)
 - [Create Permission Target](#)
 - [Update Permission Target](#)
 - [SYSTEM](#)
 - [System Health Check](#)

Instance Errors

Instance errors are returned for API endpoints that act on instances, such as [Create User](#), [Create User Group](#) or [Update Instance](#) and others.

```
{
  "data": [
    {
      "success": "false",           // "false" indicates an error occurred
      "message": <message>,       // A descriptive error message string
      "instanceName": <instance name> // The instance on which the error occurred
    }
  ]
}
```

For example:

```
{
  "data": [
    {
      "success": true,
      "message": "this instance succeeded",
      "instanceName": "localhost:8091/artifactory"
    },
    {
      "success": false,
      "message": "Connection refused",
      "instanceName": "localhost:8081/artifactory"
    }
  ]
}
```

Repository Errors

Repository errors are returned for API endpoints that act on repositories, such as [Update Repository](#) and others.

```

{
  "data": [
    {
      "success": "false",
      "message": <message>,
      "instanceName": <instance name>,
      "repositoryKey": "maven-local"
    }
  ]
}

```

// "false" indicates an error occurred
 // A descriptive error message string
 // The instance on which the error occurred
 // The repository on which the error occurred

For example:

```

{
  "data": [
    {
      "success": true,
      "instanceName": "localhost:8091/artifactory",
      "repositoryKey": "maven-local"
    },
    {
      "success": false,
      "message": "Connection refused",
      "instanceName": "localhost:8081/artifactory",
      "repositoryKey": "maven-local"
    }
  ]
}

```

// This operation succeeded
 // This operation failed

Working with User Inputs

Mission control configuration scripts offer the flexibility of letting you provide input just before the script is applied. When working with the Mission Control UI, the User Input screen allows you to enter all user input values required for the scripts you are about to apply.

When working with the REST API, another mechanism is provided that allows you to fully automate your management of Artifactory instances using [Script Mappings](#).

For a details on how to work with script mappings and user input with the Mission Control REST API, please refer to [Working with User Input](#).

Mandatory Fields

For all endpoints, mandatory input fields are indicated by a plus sign (+).

REST Resources

The following sections provide a comprehensive list of resources exposed by the Mission Control REST API.

SCRIPT MAPPINGS

Get Script List

Description: Gets the list of instance and repository configuration scripts available on Mission Control

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v1/scripts

Consumes: None

Produces: application/json

```

{
  "data": [
    {
      "name" : "string",
      "description" : "string",
      "target" : "INSTANCE" | "REPOSITORY"
    }
  ]
}

```

repository scripts

// The script name
// The script description
// Specifies whether this is an instance or

Example:

```

GET /api/v1/scripts
{
  "data": [
    {
      "name" : "QA Property Sets",
      "description" : "Applies property sets used by QA",
      "target" : "INSTANCE"
    },
    {
      "name" : "LDAP Dev",
      "description" : "Applies development group LDAP settings",
      "target" : "INSTANCE"
    },
    ...
    {
      "name" : "Docker Local",
      "description" : "Creates a local Docker repo called docker-local",
      "target" : "REPOSITORY"
    },
    {
      "name" : "Replicate releases-local",
      "description" : "Replicates releases-local repository to a target repo",
      "target" : "REPOSITORY"
    }
  ]
}

```

List User Inputs

Description: Gets the list of Artifactory user inputs needed for scripts that are being applied

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v1/userInputs

Consumes: application/json

```

{
+   "scriptMappings" : [ { //An array of scriptMapping objects
+       "instanceName" : <instance name>, // The instance on which you want to
apply a script
      "repositoryKey" : <string>, // The repository on which you want to apply the script
      // Mandatory if the operationType is UPDATE_REPOSITORY.
      // *** Not applicable and should be omitted *** if the operationType is
CREATE_REPOSITORY or UPDATE_INSTANCE
      "scriptNames" : [<script name>, <script name>, ...] //The names of the scripts you want
to apply
    }
  ]
  "operationType": "CREATE_REPOSITORY" | "UPDATE_REPOSITORY" | "UPDATE_INSTANCE" //The type of
operation you want to perform in the subsequent call with the user inputs returned
}

```

Produces: application/json

```
{
  "data": [
    {
      "success" : true,
      "instanceName" : "localhost:8091/artifactory",
      "repositoryKey" : "maven-local",          // The repository on which the script
was applied
                                                    // Only present if operationType was
UPDATE_REPOSITORY.
      "scriptUserInputs" : [
        {
          "multiple" : "boolean",          // Whether this user input item
can take multiple values
          "type" : "STRING" | "BOOLEAN" | "INTEGER" | "INSTANCE" |
"REPOSITORY", // The type of this user input item
          "value" : "object",              // A default value for this
user input item
          "description" : "string", // A description for this user input
item
          "name" : "string",                // A logical name for this
user input item
          "id" : "string"                    // The identifier of this user
input item. This is the identifier that must be used in any subsequent API call
        } ]
      ]
    }
  ]
}
```

INSTANCES

Get Instances

Description: Gets the list of Artifactory instances managed by Mission Control

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v1/instances

Consumes: None

Produces: application/json

```
{
  "data": [
    {
      "url" : <string>,          //The Artifactory instance URL
      "name" : <string>          //The Artifactory instance name
    }
  ]
}
```

Example:

```
GET /api/v1/instances
{
  "data": [
    {
      "url": "http://10.0.0.110:8080/artifactory",
      "name": "QA"
    },
    {
      "url": "http://10.0.0.120:8080/artifactory",
      "name": "DEV"
    }
  ]
}
```

Update Instance

Description: Updates an array of Artifactory instances with corresponding scripts

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v1/instances

Consumes: application/json

```
{
  "scriptMappings": [
+    {
+      "instanceName" : <string>,           // The name of the instance being updated
+      "scriptNames" : [<string>],         // Scripts to apply
+      "scriptUserInputs":                 // The user inputs needed for the
+        {
+          <userInputId> : <user input value> // string or object,
+        }
+      }
+    ]
+  }
}
```

Produces: application/json

Example: Update Artifactory instances "DEV" and "QA" by applying script "LDAP Settings" to each of them. When applying this script, I need to provide a name for the LDAP settings with user input

```
PUT /api/v1/instances
{
  "scriptMappings" : [
    {
      "instanceName": "DEV",
      "scriptNames": ["LDAP Settings"],
      "scriptUserInputs" :
        {
          "InstanceMapper#0#name#0" : "DEV-LDAP"
        }
    },
    {
      "instanceName": "QA",
      "scriptNames": ["LDAP Settings"],
      "scriptUserInputs" :
        {
          "InstanceMapper#0#name#0" : "QA-LDAP"
        }
    }
  ]
}
```


REPOSITORIES

Get Repositories

Description: Gets a list of repositories in an Artifactory instance

Since: 1.0

Security: Requires an admin user

Usage: GET /api/v1/instances/{instance name}/repositories

Consumes: None

Produces: application/json

Example: Get a list of repositories in instance cluster-126-10100.

```
GET /api/v1/instances/cluster-126-10100/repositories
```

Response:

```
{
  "data": [
    {
      "repositoryKey": "ext-release-local",
      "description": "Local repository for third party libraries",
      "type": "local",
      "packageType": "maven"
    },
    {
      "repositoryKey": "ext-snapshot-local",
      "description": "Local repository for third party snapshots",
      "type": "local",
      "packageType": "maven"
    },
    ...
    {
      "repositoryKey": "plugins-snapshot",
      "type": "virtual",
      "packageType": "maven"
    },
    {
      "repositoryKey": "remote-repos",
      "type": "virtual",
      "packageType": "maven"
    }
  ]
}
```

Create Repository

Description: Creates a repository in a set of Artifactory instances. For more information, please refer to [Configuring Repositories](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v1/repositories

Consumes: application/json

```

{
    "scriptMappings":[
+
+ the repository
+ repository
+ applied. One of these should refer to the repositoryKey field
+ previous call to userInputs
    {
        "instanceName" : <string>,           //Artifactory instance on which to create
        "scriptNames" : [<string>],          //scripts to apply when creating the
        "scriptUserInputs":                  //user inputs required for the scripts
        {
            <userId> : <user input value> //  userId obtained from
        }
    }
]
}

```

Produces: application/json

Example: Use a script called create-docker-local to create a local Docker registry. From a previous call to [userInputs](#) we found that we need to supply the repository key as user input in the RepositoryMapper#0#repositoryKey#0 parameter. We are creating the same repository on instances DEV-EAST and QA-EAST.

```

POST /api/v1/repositories
{
    "scriptMappings":[
        {
            "instanceName" : "DEV-EAST",
            "scriptNames" : ["create-docker-local"],
            "scriptUserInputs":
            {
                "RepositoryMapper#0#repositoryKey#0" : "dev-docker-local"
            }
        },
        {
            "instanceName" : "QA-EAST",
            "scriptNames" : ["create-docker-local"],
            "scriptUserInputs":
            {
                "RepositoryMapper#0#repositoryKey#0" : "qa-docker-local"
            }
        }
    ]
}

```

Update Repository

Description: Updates a repository in a set of Artifactory instances. For more information, please refer to [Configuring Repositories](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v1/repositories

Consumes: application/json

```

{
    "scriptMappings" : [
+       {
+           "instanceName" : <string>,    //Artifactory instance on which to update the
repository
+           "repositoryKey" : <string>,  //Name of the repository to update
+           "scriptNames" : [<string>],  //scripts to apply when updating the repository.
The user inputs provided be provided in an order that corresponds to the script names.
+           "scriptUserInputs" : //user inputs required for the scripts applied
+           {
+               <userId> : <user input value> // string or object, depending on
the user input type
+           }
+       }
    ]
}

```

Produces: application/json

Example: Use a script called update-local to update the description field on local repository dev-docker-local on DEV-EAST, and local repository qa-docker-local on QA-EAST.

```

POST /api/v1/repositories
{
    "scriptMappings": [
+       {
+           "instanceName" : "DEV-EAST",
+           "repositoryKey" : "dev-docker-local",
+           "scriptNames" : ["update-local"],
+           "scriptUserInputs":
dev"
+           {
+               "RepositoryMapper#0#description#0" : "Local Docker registry for
+           }
+       },
+       {
+           "instanceName" : "QA-EAST",
+           "repositoryKey" : "qa-docker-local",
+           "scriptNames" : ["update-local"],
+           "scriptUserInputs":
qa"
+           {
+               "RepositoryMapper#0#description#0" : "Local Docker registry for
+           }
+       }
    ]
}

```

SECURITY

Create User

Description: Creates a user on a set of Artifactory instances. For more information, please refer to [Managing Users](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v1/users

Consumes: application/json

```

{
+     "instanceNames":<string>],           // The Artifactory instances on
which to create this user
+     "user" :
+     {
+         "name" : <string>,                // The user's name
+         "email" : <string>,                // The
user's email
+         "password" : <string>,            // The user's password
in clear-text
+         "admin" : <boolean>,              // If true,
this is an admin user
+         "profileUpdatable" : <boolean>,   // If true, this user can update
their profile
+         "internalPasswordDisabled" : <boolean> // If true, this user cannot use internal
password when external authentication (such as LDAP) is enabled.
+     }
}

```

Produces: application/json

Example: Create user "johns" with the below parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```

POST /api/v1/users
{
  "instanceNames": ["cluster-121-10100","cluster-126-10100"],
  "user": {
    "name": "johns",
    "email": "johns@somewhere.com",
    "password": "12345678",
    "admin": false,
    "profileUpdatable": false,
    "internalPasswordDisabled": false
  }
}

```

Update User

Description: Updates a user on a set of Artifactory instances. For more information, please refer to [Managing Users](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v1/users/{username}

Consumes: application/json

```

{
+     "instanceNames":<string>],           // The Artifactory instances on which to create
this user
+     user:
+     {
+         "email" : <string>,                // The user's email
+         "password" : <string>,            // The user's password in clear-text
+         "admin" : <boolean>,              // If true, this is an admin user
+         "profileUpdatable" : <boolean>,   // If true, this user can update their profile
+         "internalPasswordDisabled" : <boolean> // If true, this user cannot use internal password
when external authentication (such as LDAP) is enabled.
+     }
}

```

Produces: application/json

Example: Update the email address of user "johns" Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```
PUT /api/v1/users/johns
{
  "instanceNames": ["cluster-121-10100","cluster-126-10100"],
  "user": {
    "email": "johns@newdomain.com"
  }
}
```

Create User Group

Description: Creates a user group on a set of Artifactory instances. For more information please refer to [Creating and Editing Groups](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v1/userGroups

Consumes: application/json

```
{
+   "instanceNames": [<string>],           // The Artifactory instances on which to create
this user
+   "userGroup" : {
+       "name" : <string>,                 // The group's name
        "description" : <string>,         // A description for this group
        "autoJoin" : <boolean>,          // If true, new users created in the target Artifactory instance
will automatically be added to this group
        "users" : [<string>]             // The list of users (by user name) to include in this group
    }
}
```

Produces: application/json

Example: Creates a user group called "developers" along with the specified parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```
POST /api/v1/userGroups
{
  "instanceNames": ["cluster-121-10100","cluster-126-10100"], // The Artifactory
instances on which to create this group
  "userGroup": {
    "name": "developers",
    "description": "The developer group",
    "autoJoin": false,
    "users": ["johns", "ronaldm"]
  }
}
```

Update User Group

Description: Updates a user group on a set of Artifactory instances. For more information please refer to [Creating and Editing Groups](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v1/userGroups/{group name}

Consumes: application/json

```

{
+     "instanceNames": [<string>],           // The Artifactory instances on which to update
this group
+     "userGroup": {
        "users" : [<string>],           // The new list of users (by user name) to include in this group.
This list replaces the current set of users in the group
        "autoJoin" : <boolean>,       // If true, new users created in the target Artifactory instance
will automatically be added to this group
        "description" : <string> // A description for this group
    }
}

```

Produces: application/json

Example: Update the "developer" user group with the specified parameters on Artifactory instances "cluster-121-10100" and "cluster-126-10100"

```

PUT /api/v1/userGroups/developers
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "userGroup": {
    "description": "The developer group",
    "autoJoin": false,
    "users": ["janes", "colonels"]
  }
}

```

Create Permission Target

Description: Creates a permission target on a set of Artifactory instances. For more information, please refer to [Managing Permissions](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: POST /api/v1/permissionTargets

Consumes: application/json

```

{
+     "instanceNames" : [<string>],           // The Artifactory instances to which this
permission target should be applied
+     "permissionTarget" : {
+         "name" : <string>,                 // A name for this permission target
+         "repositories" : [<string>],       // The repositories to which this permission
target applies
        "anyRemote" : <boolean>,           // If true, applies to any remote repository
        "anyLocal" : <boolean>,           // If true, applies to any local repository
        "excludesPattern" : <string>,     // Excludes pattern to filter out certain
repositories
        "includesPattern" : <string>,     // Includes pattern to filter in certain
repositories
        "principals":                     // The principles to which this permission target should be applied
        {
            "users" :
            {
                <userName> : [{permission}] // The users and corresponding permissions they are
given where m=admin; d=delete; w=deploy; n=annotate; r=read
            },
            "groups" :
            {
                <groupName> : [{permission}] // The groups and corresponding permissions they
are given where m=admin; d=delete; w=deploy; n=annotate; r=read
            }
        }
    }
}

```

Produces: application/json

Example: Creates a permission target called "releasers" on repository "ext-release-local" for users "johns" and "colonels" and groups "developers" and "itmanagers" (each with corresponding permissions) on Artifactory instances "cluster-121-10100" and "cluster-126-10100".

```
POST /api/v1/permissionTargets
{
  "instanceNames": ["cluster-121-10100", "cluster-126-10100"],
  "permissionTarget": {
    "name": "releasers",
    "includesPattern": "***",
    "excludesPattern": "",
    "anyLocal": true,
    "anyRemote": false,
    "repositories": ["ext-release-local"],
    "principals": {
      "users": {
        "johns": ["r", "w", "m"],
        "colonels": ["d", "w", "n", "r"]
      },
      "groups": {
        "developers": ["m", "r", "n"],
        "itmanagers": ["r"]
      }
    }
  }
}
```

Update Permission Target

Description: Updates a permission target on a set of Artifactory instances. Any permissions previously set for users or groups are replaced with the specified permissions. For more information, please refer to [Managing Permissions](#) in the Artifactory documentation.

Since: 1.0

Security: Requires an admin user

Usage: PUT /api/v1/permissionTargets/{permission target name}

Consumes: application/json

```
{
+   "instanceNames" : [<string>],           // The Artifactory instances to which this
permission target should be applied
+   "permissionTarget" :
+   {
+       "repositories" : [<string>],         // The repositories to which this permission
target applies
+       "anyRemote" : <boolean>,           // If true, applies to any remote repository
+       "anyLocal" : <boolean>,            // If true, applies to any local repository
+       "excludesPattern" : <string>,       // Excludes pattern to filter out certain
repositories
+       "includesPattern" : <string>,       // Includes pattern to filter in certain
repositories
+       "principals" :                     // The principles to which this permission target should be applied
+       {
+           "users" :
+           {
+               <userName> : [{permission}] // The users and corresponding permissions they are
given where m=admin; d=delete; w=deploy; n=annotate; r=read
+           },
+           "groups" :
+           {
+               <groupName> : [{permission}] // The groups and corresponding permissions they
are given where m=admin; d=delete; w=deploy; n=annotate; r=read
+           }
+       }
+   }
}
```

Produces: application/json

Example: Updates the permission target called "releasers" on repository "ext-release-local" for "itmanagers" awarding them full permissions on any remote repository on Artifactory instances "cluster-121-10100" and "cluster-126-10100".

```
PUT /api/v1/permissionTargets/releasers
{
  "instanceNames": ["cluster-121-10100","cluster-126-10100"],
  "permissionTarget" : {
    "anyRemote" : true,
    "repositories" : ["ext-release-local"],
    "principals": {
      "groups" : {
        "itmanagers" : ["d","w","n","r"]
      }
    }
  }
}
```

SYSTEM

System Health Check

Description: Simple ping to Mission Control to see if it is running.

Since: 1.0

Security: None

Usage: GET /api/v1/ping

Consumes: None

Produces: application/json

```
{
  "data" : true
}
```


JFrog CLI

JFrog CLI documentation has moved to a dedicated site that describes how to use the CLI with all JFrog Products.

Please refer to [JFrog CLI User Guide](#).





Disaster Recovery

Overview

As the centralized dashboard for all your Artifactory services, JFrog Mission Control is the perfect place to configure disaster recovery. The purpose of configuring disaster recovery is to prevent the loss of critical data in the event that one of your Artifactory services experiences an event that causes irreversible damage and loss of data, or if it needs to be taken down gracefully for any other reason (e.g. hardware maintenance on the server machine). JFrog Mission Control lets you configure disaster recovery in the **Admin** module under **DR | DR Configuration**.


Disaster Recovery is implemented by configuring complete system replication between **Master** services and corresponding **Target** services where a Master service holds critical data you want to protect against irreversible damage, and the corresponding DR Target is the replication target of the Master.

Managing DR involves the following basic steps:

1.	Sync Artifactory Encryption Key	If your Master service has Artifactory Key Encryption enabled, you need to sync over the Artifactory Encryption Key to your Target service so that all passwords can be properly decrypted once your security settings are replicated to the Target. For details, please refer to Artifactory Key Encryption in the JFrog Artifactory User Guide. From version 5.5, new installations of JFrog Artifactory will have Artifactory Key Encryption enabled by default. To sync the Artifactory Encryption Key to the target, you can perform a system import of the master, or just run the DR steps with encryption disabled on both master and target.
2.	Configure	Configuring your DR replication Master and Target pairs.
3.	External Sync (Optional)	<p>You may work with the relevant department in your organization to manually sync between Master and Target services outside of both Mission Control or Artifactory before you initialize DR (step 3, Init, below).</p> <p>This optional, external synchronization can avoid lengthy and resource intensive synchronization (step 4 below) if the storage on your Master service contains large amounts of data.</p>
4.	Init	<p>Establishing the replication relationships between all local repositories on the Master service and the corresponding repositories on the Target service.</p> <p>Backing up security settings and various configuration files from the Master service to Mission Control. These are later pushed to the Target service.</p> <div> No data transfer Note that at this stage repository data is not yet transferred from the Master to the Target service.</div>
5.	Synchronize	<p>Invoking replication from the Master service to the Target service so that each local repository on the Target service is synchronized with the corresponding repository on the Master service.</p> <div> Now you're protected Once all repositories on your Target service are synchronized with your Master service, your system is DR protected. This means you can instantly invoke failover from your Master to your Target service so that your users may transparently continue to get service from Artifactory.</div>

Page Contents

- [Overview](#)
- [Configuring Master and Target Pairs](#)
 - [Authentication with Access Tokens](#)
- [Initializing DR](#)
 - [Updating Configuration Files](#)
- [Synchronizing Repositories](#)
 - [Manual Sync](#)
 - [Automatic Sync](#)
 - [Sync Status](#)
- [Activating DR](#)
- [Restoring the Master Service](#)
- [Deleting a DR Configuration](#)
- [DR Configuration Properties](#)

6.	Activate	Invoking failover from the Master service to the Target. Once this operation is complete, all requests targeted at the Master service will automatically be rerouted to the Target service.
		<div>  DR is now in effect </div> <p>Once you have activated your DR setup, and failover is in effect, your Artifactory users will continue to get service transparently without having to make any changes to their environments.</p>
7.	Restore	Restoring a Master service from its Target once the event that spawned DR is remediated.



DR is not HA

Don't confuse setting up Artifactory in a High Availability configuration with setting up Disaster Recovery.

A high availability configuration uses two or more redundant Artifactory servers to ensure users continue to get service if one or more of the Artifactory servers goes down (whether due to some hardware fault, maintenance, or any other reason) as long as at least one of the servers is operational. Once HA is set up, service continues automatically with no intervention required by the administrator.

Disaster recovery is designed to continue providing service as quickly as possible if a **whole Artifactory installation** goes down (for example, all servers in an HA installation go down due to an electrical malfunction in the site). In this case, requests can be rerouted to the Target service, and, while this is not automatic, it is achieved with a single click of a button in Mission Control.

Configuring Master and Target Pairs

The Master and Target pairs you have configured are displayed in the **Manage** module under **DR Configuration**.



Master	Target
sadevp1	eudevp1

To configure a new Master / Target pair click **Create DR**.

Create DR

Select Master
Filter...
☐ assanbox1
☐ assanbox2
☐ assanbox3
☐ assanbox4
☐ assanbox5
☐ assanbox6
☐ euchub1
☐ euchub2
☐ euchub3
☐ euchub4
☐ eudevp1
☐ eudevp2
☐ nadevp1
☒ sadepv1
☐ sadevp2

Select Target
Filter...
☐ assanbox1
☐ assanbox2
☐ assanbox3
☐ assanbox4
☐ assanbox5
☐ assanbox6
☐ euchub1
☐ euchub2
☐ euchub3
☐ euchub4
☒ eudevp1
☐ eudevp2
☐ nadevp1
☐ sadevp1
☐ sadevp2

Cancel Save

Select the Artifactory services you wish to define as the Master and Target pair and click "Save".



Master and Target Artifactory version

Master and Target pairs must both be running Artifactory v4.7.2 or later.

Once you have configured the Artifactory services of a Master and Target pair, you can not change them. If you need to change a Master and Target pair, you need to delete the pair from the list and configure a new pair.

To proceed with the DR process, click the **Control Panel** icon to display the Disaster Recovery screen for the selected Master and Target pair.

The Disaster Recovery screen displays several panels in two separate tabs.

The **Master-Target** tab displays:

- **Master:** Provides details about the Master service
- **Target:** Provides details about the Target service
- **Log:** Displays log file output

DR configurations

Create DR

< Page 1 of 1 >

Master	Target
sadepv1	eudevp1

The **Repositories** tab displays the synchronization status of the corresponding repositories on the Master and Target services.

The **Actions** menu lets you refresh the **Repositories** table, perform a sync test, or proceed through the phases of DR (Init, Activate etc.)

Disaster Recovery

DR phase: Initiated

Master-Target

Repositories

Actions

Filter by Repository Key

Filter by:

Fully synced

Out of sync

Missing repositories

Page 2 of 2

Repository Key	Master						Target					
	Type	Space	Folders	Files	Metadata Files	Metadata Size	Type	Space	Folders	Files	Metadata Files	Metadata Size
libs-release	VIRTUAL	0 bytes	0	0	N/A	N/A	VIRTUAL	0 bytes	0	0	N/A	N/A
libs-release-local	LOCAL	8.45 KB	3	2	1	400 bytes	LOCAL	5.06 KB	3	2	1	348 bytes
some-helm	LOCAL	0 bytes	0	0	N/A	N/A	LOCAL	0 bytes	0	0	N/A	N/A
bintray-docker-remote...	CACHE	0 bytes	0	0	N/A	N/A	CACHE	0 bytes	0	0	N/A	N/A
toto-npm-prod-local	LOCAL	0 bytes	0	0	N/A	N/A	LOCAL	0 bytes	0	0	N/A	N/A

Repository Key	A logical name for the service.
Type	The service type. Artifactory or Xray.
Space	Total used space for the repository.
Folders	Number of folders in the repository.
Files	Number of files in the repository.
Metadata Files	Number of metadata files. (Applicable only for Maven repositories)
Metadata Size	Size of metadata files. (Applicable only for Maven repositories)

Authentication with Access Tokens

Any [access tokens](#) issued by the Master service for authentication will not work with the DR Target. For clients to work with the DR target once DR has been activated, the Target must issue new access tokens.

Initializing DR



Using Master Key Encryption

If your Master service has Master Key Encryption enabled, you need to sync over the Master key to your Target service so that all passwords can be properly decrypted once your security settings are replicated to the Target. For details, please refer to [Artifactory Key Encryption](#) in the JFrog Artifactory User Guide.

Once your Master and Target pairs are [Configured](#), to initialize DR, select **Init** from the **Actions** menu.

Initializing DR essentially means setting up all the replication configurations that are needed for DR, meaning:

- Every repository on the Master service has a corresponding repository on the Target service
- Replication from each local repository on the Master to the corresponding local repository on the Target is configured. At this stage, replication is not yet enabled.
- Replication from each local repository from the Target to the corresponding local repository on the Master is configured but **disabled**. (This configuration will later be used during the [Restore](#) operation.)
- If the Master had additional replications, not related to DR, configured, these are duplicated in the Target, but are also **disabled**.



Make sure your Master and Target services have corresponding repositories with enough storage

It is up to your JFrog Mission Control administrator to ensure that each repository on the Master service has a corresponding repository on the Target service for replication. Before initializing DR, Mission Control will also verify that the target service has enough storage available.

Example

Consider that repository maven-local on service **Master** has a replication configured to my-local-maven on service **Other**. For maven-local to be DR protected:

- There must be a **Target** service with a corresponding maven-local repository defined
- Replication is configured from maven-local on **Master** to maven-local on **Target**, however, it is currently **disabled**.
- Replication from maven-local on the **Target** back to **Master** is also configured, but this replication is also currently **disabled**.
- Replication from maven-local on the **Target** to my-local-maven on **Other** is also configured, but this replication is currently **disabled**.

Updating Configuration Files

Initializing DR performs different actions on configuration files in the Target service.

The following entities are **overridden** (replaced by their counterparts from the Master service)

- Existing repositories having the same name (e.g. a repository called "maven-local" on the Target is replaced with a repository of the same name that exists on the Master)
- Property sets

The following entities are **updated**

- Layouts
- LDAP settings
- LDAP groups
- SSO configuration

In addition, any user plugins existing on the Master but missing on the Target are created on the Target service.

Synchronizing Repositories

When invoking DR, or restoring your Master service, at some point, you will need to synchronize repositories between your Master and Target services:

- When invoking DR, you need to synchronize repositories once initialization is complete and you will be moving data from your Master to your Target service.
- When doing a Restore, you need to synchronize repositories after invoking **Restore** from the **Actions** menu and you will be moving data from your Target back to your Master service.

Depending on the amount of data in your filestore, this may be a resource intensive operation. Therefore, to avoid overloading your systems which may cause performance issues, Mission Control lets you manage synchronization of your repositories either manually or automatically.

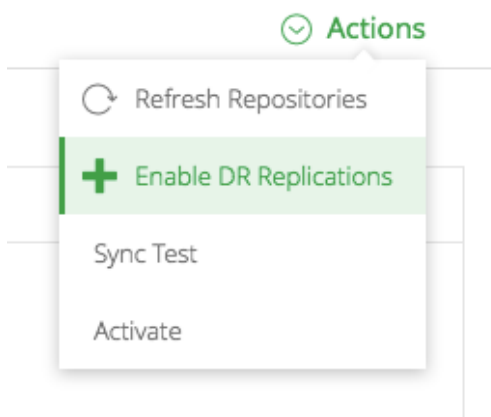
Manual Sync

You can invoke replication directly on the Artifactory service (the Master service when invoking DR, the Target service when doing a restore) in a gradual manner according to your IT policies and available bandwidth. For details, please refer to [Replication](#) in the Mission Control User Guide, or to [Scheduling and Configuring Replication](#) in the JFrog Artifactory User Guide. In this way, using your knowledge of how much data is hosted in each of your repositories, you may implement a gradual synchronization process with minimal or no impact to your system performance.

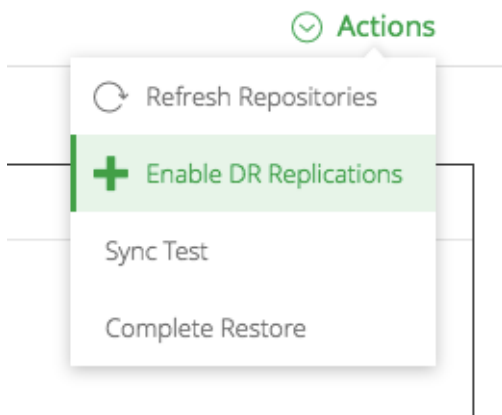
Automatic Sync

If the amount of data in your system does not pose any performance risk during synchronization, you may invoke a full synchronization automatically through Mission Control from the **Actions** menu by selecting **Enable DR Replications**. Mission Control invokes replication indirectly by setting the cron expression that determines the timing of replication for each repository being replicated.

When invoking DR, it looks like this:



For a Restore, it looks like this:



Even so, Mission Control avoids risking a performance hit and ensures that replications are not all invoked simultaneously. All replications are configured to run sequentially, at 5 minute intervals (default) to ensure that replication is initiated in a staggered manner to avoid a heavy load burst when data transfer begins. You can modify the time interval through the `drconfig.replication.cronIncrement` parameter in your `$MC_HOME/etc/mission-control.properties` file.

Sync Status

At any time, the **Repositories** panel gives you a clear picture of the synchronization status of your repositories. Click any header in the **Legend** to view only those repositories in the selected status. For example, click **Missing Repo** to view those repositories that exist in the Master service but were missed in the Target service. The table in the Repositories panel displays the following information for both the Master and Target service:

Repository Key	The repository key
Space	The amount of storage occupied by files in the repository
Files	The number of physical files in the repository
Folders	The number of folders in the repository
Items	The number of items in the repository. Note that while a file is stored only once, it may appear as several items in different locations in the repository.

Activating DR

Activating DR is the process of bringing the Target service into operation when the Master service is down or not available. Once activation is complete, an administrator needs to update the DNS or load balancer to point to the Target service.

To activate DR, from the **Actions** menu, click "**Activate**"

JFrog Mission Control

Add Service
Execute Script
Welcome, admin (Log Out)
Help

Disaster Recovery

DR phase: Initiated

Master-Target
Repositories

Master

Status: ONLINE
Name: nadevdr1
Location:
Url: <http://104.196.248.15:8010/artifactory>
Version: 5.5.2

Target

Status: ONLINE
Name: nadevdr2
Location:
Url: <http://130.211.213.126:8010/artifactory>
Version: 5.5.2

Refresh Repositories
+ Enable DR Replications
Sync Test
Activate

```

=====
20-11-17 08:58:32 UTC
SYNC Results:
=====
On nadevdr1 there is 163.29 GB (82.96%) of available storage.
On nadevdr2 there is 55.41 GB (28.15%) of available storage.
Config Descriptor of the service nadevdr1 cached in MC
Security Descriptor of the service nadevdr1 cached in MC
Config Descriptor of the service nadevdr2 cached in MC
Security Descriptor of the service nadevdr2 cached in MC
Security Descriptor of the service nadevdr1 pushed to the service nadevdr2
=====

```

Close

The following actions that occur during the activation process depend on whether the Master service is up or not:

Master service is up

- Replication is globally disabled.

Master is down or was gracefully turned off

- Mission Control issues a notification in the UI that the Target service is up and running; the Master service is displayed as being offline.
- Replication is globally enabled on the Target service.

At this point, your administrator needs to redirect traffic from the Master service by pointing your DNS or load balancer to the Target service on the DR environment. This change should not have any effect on the IP/DNS records that are configured on your Mission Control DR configurations.

This brings you to the **Failover** phase in which network traffic goes to your Target.

JFrog Mission Control

Add Service
Execute Script
Welcome, admin (Log Out)
Help

Disaster Recovery

DR phase: Failover

Master-Target
Repositories

Master

Status: ONLINE
Name: nadevdr1
Location:
Url: <http://104.196.248.15:8010/artifactory>
Version: 5.5.2

Target

Status: ONLINE
Name: nadevdr2
Location:
Url: <http://130.211.213.126:8010/artifactory>
Version: 5.5.2

```

=====
20-11-17 09:07:56 UTC
ACTIVATE Results:
=====
Push replications from service nadevdr2 to service nadevdr1 were disabled
Global replication on service nadevdr1 was set to FALSE
Security Descriptor of the service nadevdr1 cached in MC
Push replications from service nadevdr1 to service nadevdr2 were disabled
Security Descriptor of the service nadevdr1 pushed to the service nadevdr2
Global replication on service nadevdr2 was set to TRUE
=====

```

Close

Restoring the Master Service

Once the Master service can be brought back into normal operation, you need to restore your system and data back from the Target service. This process mirrors the process of setting up and activating DR.

Before you proceed with the restore operation, you need to ensure the following pre-requisites are met:

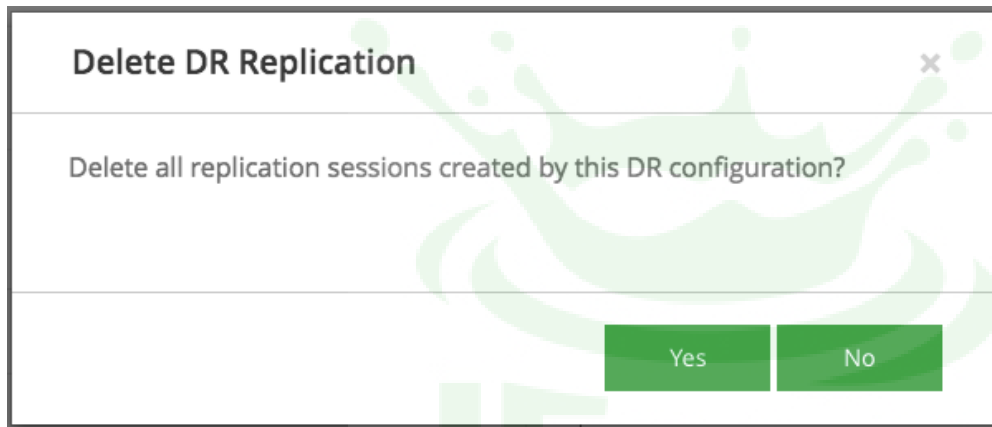
- Both the Master and Target services are up and running
- Replication in the Master service is [globally disabled](#) (otherwise you run the risk of losing data).
- No changes should be made to the Target service configuration during the restore operation

Once these pre-requisites are met, invoke **Restore** by selecting it from the **Actions** menu. During the restore process, Mission Control performs the following actions:

- Duplicate all non-DR replications on the DRTarget service back to the Master service.
- Create all repositories that exist on the Target service but not on the Master service

Now you are ready to synchronize your repositories from your Target service back to your Master service as described in [Synchronizing Repositories](#).

Once you have finished synchronizing your repositories you need to finalize the restore operation by selecting **Actions | Complete Restore**. This action gives you the option of removing the DR replication sessions between the corresponding Target and Master once the restore is complete to avoid having redundant replications configured for the Target.



Finally, once the restore operation is complete, your administrator needs to redirect traffic Target service back to your Master service by manually updating your load balancer or DNS to point to the Master service.

Deleting a DR Configuration

To delete a DR configuration, simple click the corresponding "Delete" icon in the **DR Configurations** list.



Note that since any DR configuration that has been invoked will create replication sessions between the corresponding Master and Target services, you have the option of leaving those replication sessions defined or removed when you delete the DR configuration.

Confirm Delete

✕

When deleting the DR configuration, you may also delete the replication sessions it created.

Cancel

Delete with Replication Sessions

Delete

DR Configuration Properties

Your `$MC_HOME/etc/mission-control.properties` includes a number of properties related to your DR configuration:

drconfig.replication.cronExpression	When performing an automatic sync of repositories, this parameter determines when the first repository will be synchronized.
drconfig.replication.cronIncrement	<p>Default: 5 min</p> <p>When performing an automatic sync of repositories, this parameter determines the time interval between the successive initiation of replication for the local repositories on the Master service. For example, if set to 5 minutes, Mission Control will set the cron expression in the first repository in the Master service to invoke replication as specified in the cron expression, the next one will start 5 minutes later, the one after that, 5 minutes later again, and so forth.</p> <div> <div>✓</div> <div> When should you change this? If several repositories on your Master service that you are replicating have large amounts of data, we recommend increasing this value to avoid excessive loads on your system. </div> </div>
drconfig.sync.period	<p>Default: 300,000 ms</p> <p>Mission Control periodically synchronizes the repository definitions and replication settings between the Master and Target service (config descriptor) . This parameter sets period between these sync job executions.</p>
artifactory.client.socketTimeout	<p>Default: 45 seconds</p> <p>Sets the timeout period for a socket opened to Artifactory.</p> <div> <div>✓</div> <div> If your Artifactory service services thousands of users and has thousands of repositories, the security and configuration descriptors may be too big to complete transfer before the socket times out. Increasing the socketTimeout parameter should solve this issue. </div> </div>
drconfig.replication.socketTimeout	<p>Default: 15000 milliseconds</p> <p>Sets the Artifactory replication socket timeout for DR.</p>
drconfig.space.detection.disabled	<p>Default: true</p> <p>Before initializing DR, Mission Control verifies that the target service has enough storage available. When true, this parameter specifies that Mission Control should not perform this check for available space. This is the desired behavior when your target service uses cloud storage in which case the check for available space is not needed.</p>

If you modify your `$MC_HOME/etc/mission-control.properties` file, we recommend restarting Mission Control to make sure your changes take effect.



System Backup and Rapid Recovery

Overview

As the centralized command and control center for all your Artifactory instances, we recommend maintaining a dormant copy of Mission Control that can quickly take over in case your main installation goes down for any reason. To facilitate rapid recovery capabilities, this page offers system backup procedures that keep your dormant copy updated in the background and ready to take over at a moment's notice. These procedures are based on frequently synchronizing the `MC_HOME/etc` and `MC_HOME/data` folders from your main Mission Control instance to your recovery instance during normal operation.

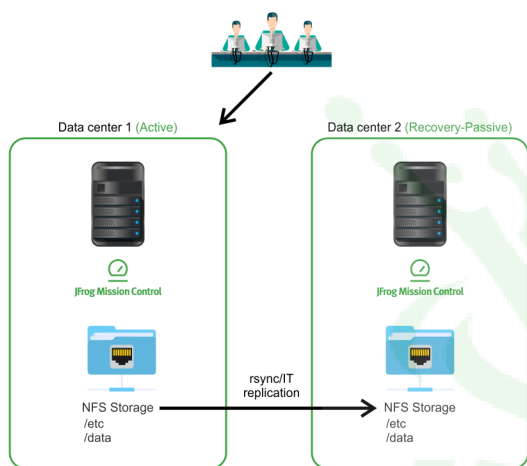


We recommend backing up the MC_HOME/etc folder

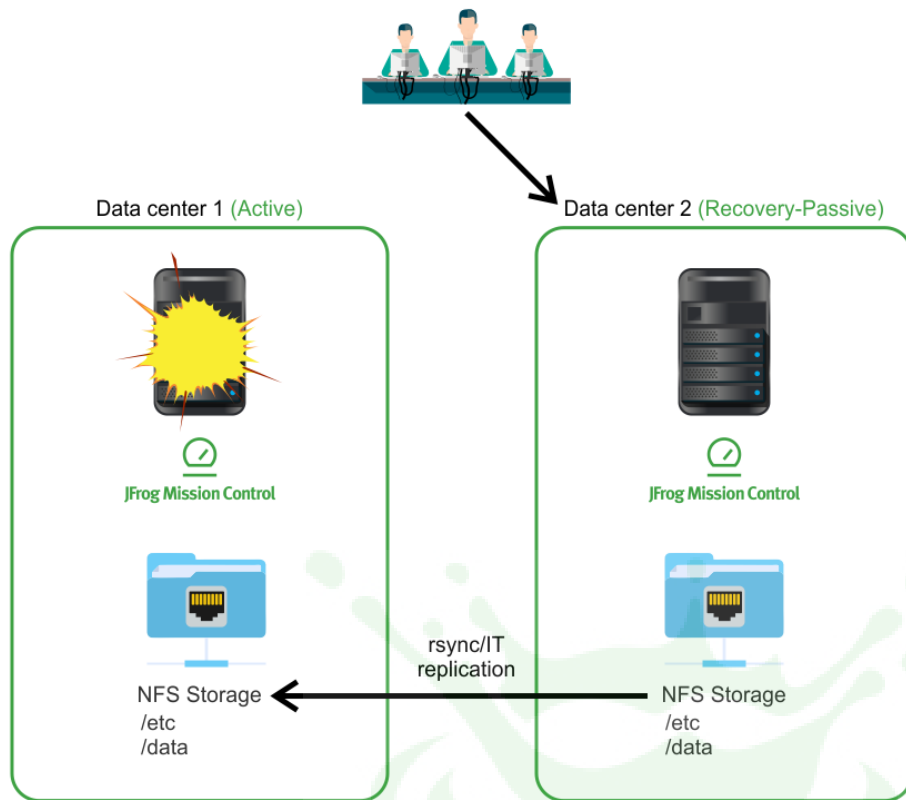
While not strictly required, we strongly recommend backing up the `MC_HOME/etc` folder so that Mission Control properties are also backed up and can be applied to the recovery instance.

Page Contents

- [Overview](#)
- [Backing Up for Rapid Recovery](#)
 - [Running with Docker](#)
 - [Non-Docker Installations](#)



Then, in case of a failure on your main Mission Control installation, your administrator can quickly implement failover to the recovery instance by updating your DNS record, or modifying your load balancer configuration to point to your recovery instance. Once failover is complete and the recovery instance is operational, it then synchronizes any further changes in configuration back to the storage of the main instance so it can take over once the fault is remediated.



Backing Up for Rapid Recovery

Running with Docker

For an installation of Mission Control as a [Docker image](#), data and configuration files are stored under the `$MC_HOME` folder (whose default location is `$HOME/.jfrog/jfmc`).

To enable fully functional recovery following a failure in Mission Control, you need to back up all content in this folder.

Non-Docker Installations

In non-Docker installations:

- A data folder is set up (default location is `/var/opt/jfrog/mission-control`).
- User choices made during the installation are also stored in: `/opt/jfrog/jfmc/scripts/setenv.sh`.

To enable fully functional recovery following a failure in Mission Control, you need to back up all content in the above `mission-control` folder and the `setenv.sh` file.

Troubleshooting

Installation

Cause	From version 2.0 standalone ZIP installations have been deprecated from Mission Control.
Resolution	For a full list of installation options, please refer Installation and Upgrade

- [Installation](#)
- [Disaster Recovery](#)
- [Scripting](#)
- [Graphs](#)
- [Groups](#)
- [REST API](#)
- [Services](#)

Disaster Recovery

Cause	The DR target is not able to decrypt passwords since it does not have the same Master Key as the DR Master service.
Resolution	Sync over the Artifactory Master Key from the DR Master to your Target Artifactory service. This will allow all passwords to be properly decrypted once your security settings are replicated to the Target. For details, please refer to Master Key Encryption in the JFrog Artifactory User Guide.

Scripting

Cause	In version 2.0, Mission Control made significant changes in how configuration scripts are written. These changes are not backwards compatible, so any scripts written for Mission Control version 1.x will not work. For details, please refer to Configuration Scripts . <TBD update link>.
Resolution	You need to migrate your scripts to work with Mission Control 2.0 and above. While the migration process is not automatic, it is quite simple. For details please refer to Migrating Scripts from Version 1.x to Version 2.x . <TBD update link>

Cause	Previous to Mission Control version 2.0, implementing a Star Toplogy was very complex, cumbersome and error-prone.
Resolution	A new configuration block was introduced in Mission Control version 2.0 to make implementing a Star Toplogy, using either push or pull replication, very simple. For details, please refer to Star Toplogy . <TBD update link>

Cause	Previous to Mission Control version 2.0, implementing a Full Mesh Toplogy was very complex, cumbersome and error-prone.
Resolution	A new configuration block was introduced in Mission Control version 2.0 to make implementing a Star Toplogy, using either push or pull replication, very simple. It is now much easier to implement a Full Mesh by configuring each node with its own Star Toplogy. For details, please refer to Star Toplogy . <TBD update link>

Cause	The default layout for any new repository is maven-2-default.
Resolution	When creating a new repository in a script, modify the <code>repoLayoutRef</code> field to the correct layout for your repository type.

Cause	When updating a repository using a script, any parameter that is not explicitly specified is updated to the default value of that parameter.
Resolution	When updating a repository using a script, specify all the parameter for that repository that don't take the default value, even those that shouldn't change.

Cause	Mission Control scripts are maintained in one of its internal databases which cannot be accessed from outside of Mission Control.
Resolution	To access Mission Control scripts, configure a Git repository to which Mission Control will synchronize all its scripts as described under Git Integration <TBD update link>. This also allows you to create and modify your scripts using any external editor.

Graphs

Cause	After adding a service to the system, it can take up to 15 minutes before Mission Control displays any data samples.
-------	--

Resolution	Just wait up to 15 minutes, and you should then see data for the new added service.
------------	---

Cause	To preserve historical data for deleted services, Mission Control stores the data under a fictitious service named <deleted_service_name>_old_xx (e.g. art1_old_01, art1_old_02. XX will be incremented if you delete and re-add a service with the same name.
Resolution	When viewing graphs for all services, these deleted will still appear, however, you may also focus on a specific service.

Groups

Cause	From version 2.0, Groups have been deprecated in Mission Control.
Resolution	Previous to version 2.0, creating Groups was a convenient way to run scripts on a number of Artifactory services at once since each script could only perform one action at a time. From version 2.0, scripting is much more flexible and you can specify any number of services on which a script should act.

REST API

Cause	In version 2.0, Mission Control implemented a major upgrade of the REST API which is not backwards compatible with previous versions.
Resolution	You need to update your scripts to use the new REST API. For details, please refer to the updated Mission Control REST API <TBD update link> page, especially the Version Mappings <TBD update link> section that describes how to map endpoints from previous versions to version 2.0 and above.

Services

Cause	Mission Control extracts storage information about managed Artifactory services using a REST API call.
Resolution	In some cases, the REST API call may take time to return. Wait a few moments and refresh the screen.

Release Notes

Overview

This page presents release notes for JFrog Mission Control describing the main fixes and enhancements made to each version as it is released.


Download

Click to download the latest version of [JFrog Mission Control](#).

Installation

For installation instructions please refer to [Installing Mission Control](#).

To upgrade to the latest version, please refer to [Upgrading Mission Control](#).

 To receive automatic notifications whenever there is a new release of Mission Control, please watch us on [Bintray](#).

Page Contents

- [Overview](#)
 - [Download](#)
 - [Installation](#)
- [Mission Control 2.1](#)
 - [Mission Control 2.1.1](#)
- [Mission Control 2.0](#)

Mission Control 2.1

January 8, 2018

Highlights

Ubuntu and Red Hat Installation

This release introduces support for additional flavors of Linux and adds Ubuntu 16.x and Red Hat 7.x as supported platforms. For details on how to install Mission Control on these platforms, please refer to [Installing Mission Control](#).

Using External Databases

Mission Control uses Elasticsearch, MongoDB and PostgreSQL databases for its different functions, and until now, would install dedicated instances of each of these databases. This version gives you more control of your resources and lets you direct Mission Control to use instances of these databases you may already have installed and in use, rather than creating new ones. Offering full flexibility, during the installation or upgrade process, Mission Control lets you select which databases should be externalized and which Mission Control should create for its own dedicated use.

Maven Metadata in DR Repositories Tab

In addition to the replication status, for Maven packages, the [DR Repositories](#) tab now also includes the metadata file count and size for each repository.

System Monitoring

Mission Control now displays your general system status as an icon on the top ribbon of the UI. Services notifications are automatically updated at all times.

Feature Enhancements

1. In both the UI and the REST API, the option to execute a script on a non-online service is blocked.
2. JFrog Mission Control blocks creating a DR pair when the master version is higher than target version.
3. In the REST API, two new commands have been added. The [Change Password](#) command allows users to change their own password and enables the admin to change passwords for all users. The [Partial Update Site By Name](#) command updates site information without updating the attributes.
4. When setting DR, you can manually set the replication socket timeout in the [DR configuration properties file](#).
5. Mission Control allows you to add a commit message when updating a script.
6. The [Top 5 Services graph](#) now displays the percentage of used storage.

Mission Control 2.1.1

January 23, 2018

Highlights

Helm Chart Repositories

This release adds full support in Mission Control for [Helm Chart repositories](#) which were introduced in JFrog Artifactory 5.8.

Mission Control 2.0

November 20, 2017

JFrog is pleased to release Mission Control 2.0.

This release introduces many changes from version 1.x to improve workflow and efficiency in managing your global Artifactory and Xray services. Yes, you read correctly, Mission Control now also manages your Xray services. In addition, Mission Control 2.0 introduces significant changes in installation and upgrade procedure, workflow for adding and managing services, a new concept of [Sites](#) that associate services to a geographic location, improvements in [Usage Graphs](#) and more.

Note that some of the new features and enhancements are breaking changes that are not compatible with version 1.x. In these cases, we offer a migration path to version 2.0.

For details about the changes introduced by Mission Control 2.0, please read the sections below.

Highlights

Artifactory and Xray Services

In addition to managing your Enterprise Artifactory instances, Mission Control can now also manage the JFrog Xray instances attached to them. This allows you to do things like configure connections between Artifactory and Xray services, use scripts to create [Watches](#) and more.

To manage Artifactory or Xray instances through Mission Control, they must be added as [Services](#) and be assigned to [Sites](#).

Scripting

Scripting has undergone significant changes in Mission Control 2.0. Most importantly, configuration scripts are now much more flexible allowing you to operate on as many Artifactory or Xray services, and on any number of repositories as you want in a single script (previously, each script could only perform one action on a single service). To support this capability, Mission Control 2.0 introduces service closures in configuration scripts. These define the Artifactory and Xray instances on which the script operates and enclose the different configuration blocks that create or implement changes on Artifactory and Xray services.

The scripting DSL has also been significantly enhanced with new configuration blocks. These include configuration blocks to create and configure Xray services, a security block to configure users, groups and permissions in Artifactory services and more. Make sure to check out the [Star Topology](#) configuration blocks that make it very easy to set up a complex one-to-many replication relationship using three lines of code. Not also that applying a Star Topology configuration to all members of a star actually implements a full mesh topology.



Breaking Change - You need to migrate your scripts

The new scripting mechanism is a breaking change which means that scripts written for JFrog Mission Control 1.x will not work in version 2.0 and above. While there is no way to migrate your scripts automatically, the process is not very complicated. For guidelines and best practices for migrating your scripts please refer to [Migrating Scripts from Version 1.x to Version 2.x](#).

Graphs

The Graphs UI has been enhanced to give you an easy way to focus on different Artifactory instances and repositories and also zoom in on specific time periods on historical usage graphs. From version 2.0, Mission Control uses a new Elasticsearch database to store historical usage data. Upon installation of the new version, the previous InfluxDB database will no longer be used, and usage data will only be collected in the new database.



[Optional] You may migrate historical usage data from InfluxDB to Elastic Search

If you wish to continue viewing historical usage data collected in the InfluxDB database of versions 1.x, you can migrate this data to the new Elasticsearch database using the process described in [Migrating Scripts from Version 1.x to Version 2.x](#).

Installation and Upgrade

Instructions for installing and upgrading to version 2.0 have changed, but remain simple procedures, with support for **CentOS**, **Debian** and **Docker** installations.

To install Mission Control 2.0 as CentOS or Debian distribution, please refer to [Installing Mission Control](#).

To install and run the Mission Control Docker image, please refer to [Running with Docker](#).



ZIP installation is deprecated

From version 2.0, Mission Control is no longer available as a ZIP installation.

REST API

Mission Control 2.0 introduces a completely new REST API that accommodates all the new functionality introduced in this version. The new REST API adds several new endpoints, but also removes some, including endpoints related to configuring users, groups and permissions in Artifactory since this functionality is now available through JFMC scripting. From Mission Control 2.0, the REST API v3 is active, while the REST API v2 is deprecated.



Breaking Change - You need to migrate your REST API calls

The new REST API is a breaking change which means any scripts that use the previous REST API version will not work. To learn how to migrate your scripts to the new REST API, please refer to [Version Mappings](#) in the new Mission Control REST API page.

Sites

Sites are a new concept in JFrog Mission Control. They represent physical locations (cities) into which you can aggregate the different Artifactory and Xray services serving them. Any service defined in Mission Control must be assigned to a site. Sites are displayed in the [Explore](#) module (which replaces the old Dashboard module) and can display sites as a list view or on a map.



Existing Artifactory services are automatically assigned to a site

When you upgrade to Mission Control 2.0, any Artifactory services already managed by your current version of Mission Control will be assigned to new Sites that will be created according to the location of your Artifactory instances. For example, an Artifactory service located in San Francisco, will be assigned to a new site in Mission Control that's located in San Francisco.

Artifactory services that do not have a location will be placed in the "Unassigned Services" site by default.



"Groups" feature is deprecated

From version 2.0, collecting managed Artifactory services into "Groups" is deprecated. All services should be placed in the context of a Site.

Feature Enhancements

1. Mission Control startup time has been greatly improved.
2. Mission Control's DSL has been enhanced with new configuration blocks to allow extremely easy configuration of replication relationships that implement Star Topologies for geographically distant Artifactory services. [starPush](#) configures a multi-push replication relationship while [starPull](#) configures pull replication. As a result, the `multipushReplication` configuration block, which is now redundant, has been deprecated.
3. The [Execute Script](#) REST API endpoint has been enhanced to provide more detailed error messages if script execution fails

Issues Resolved

1. Fixed an issue that prevented Mission Control from implementing DR when the Master Artifactory instance was using [encrypted passwords](#).