# System YAML Configuration File

## Overview

One of the important requirements of modern-day applications is the ability to manage **configuration as code**. Configuration as code means that the file being used to configure your system is immutable to the application, except in cases where the password and secrets are in plain-text, in which case the application will encrypt and correct the encrypted string back to the file.

Configuration as code can be used in “cloud native” setups, where the configuration file can be “injected” when the application starts up. This means that you can modify the configuration externally and without having the application running, and simply “provide” it to the application at startup.

The modified file is then placed in the correct filesystem path (for Linux, Docker, RPM, Debian installations) or as a secret in Helm installations using the Helm `values.yaml` file.

Page contents



## Implementing Configuration as Code in JFrog

JFrog provides a flexible way to configure your system using a single `system.yaml` configuration file found in the `$JFROG_HOME/<product>/var/etc` folder for each product.

With the JFrog `system.yaml` file, you can set up your system configuration in a single config file and provide that file as an input to your server as you deploy the server. The configuration files allow you to manage several aspects of your system, including resources, security settings, databases, and external connections. All possible configurations are provided in the template YAML file, available under `$JFROG_HOME/<product>/var/etc/`.

Each JFrog component has its own `system.yaml` file, which you can use to configure:

- JFrog Artifactory
- JFrog Xray
- JFrog Insight
- JFrog Distribution
- JFrog Pipelines

After installing the JFrog Platform or product, simply use the relevant YAML file to set your configurations before starting up your server.

**YAML Best Practices**

To learn more about best practices when using YAML, see best practices for using YAML files.

## Using System YAMLs

### YAML File Format

The YAML file is constructed with **keys** and **entities**, using the following format:

```
key: [entity]
```

For example:

```
configVersion: 1
...
# common database used by all mission_control services
shared:
  database:
    type: postgresql
    host: localhost
    port: 5432
    name: mission_control
    ...
  ...
...
```

### Configuration Value Hierarchy

Configuration values are applied according to the following hierarchy:

1. `System.yaml` service section
   if nothing is found there
2. `System.yaml` shared section
   if nothing is found there
3. Environment variables.
   if there are no variables, move to the next values
4. Etc.

If the user does not set any values in any of the files above, the application will use the default settings of the application.



The system yaml templates (`system.full-template.yaml`) represent the default value as they are configured in the application.

## Applying Configuration Changes

Once you have configured your YAML file to include all the configuration changes needed, you can apply them by restarting the server.



**Take care when modifying Artifactory configurations**

Modifying the system configurations is an advanced feature. Since it is easy to overwrite configurations, it is strongly recommended backing up the configuration before making any direct changes, and taking great care when doing so.

## Shared Configuration Versus Operation Microservices

As explained above, each product in the JFrog Platform has its own `system.yaml` file that enables you to configure its behavior. Each YAML file, however, includes both shared configurations - which apply to all of the JFrog Platform products - and dedicated operational microservices configurations, which are relevant only to that product.