# Basic Cluster Setup

The following dependencies have to be setup first (from OperatorHub):

* Cluster Logging
* Pipelines
* ArgoCD
* enable prometheus user workload monitoring
  + < 4.3: not available
  + <= 4.5 <https://docs.openshift.com/container-platform/4.5/monitoring/monitoring-your-own-services.html>
  + >= 4.6 <https://docs.openshift.com/container-platform/4.6/monitoring/enabling-monitoring-for-user-defined-projects.html>
  + See https://raw.githubusercontent.com/prft-rh/devopsinabox-operator/main/install/clustermonitoring-config.yaml
* enable log forwarding
  + < 4.3: not available
  + <= 4.5 https://docs.openshift.com/container-platform/4.5/logging/cluster-logging-external.html#cluster-logging-collector-log-forward-enable\_cluster-logging-external
  + >= 4.6 [enabled](https://docs.openshift.com/container-platform/4.6/monitoring/enabling-monitoring-for-user-defined-projects.html) by default

Create an Openshift project to install the DevSecOpsManager resource into.

In that project, install the following operators (if not already installed globally):

* Grafana
* Elasticsearch (from Elastic.co)

Install the provided ClusterRoleBinding object for LogForwarding <https://raw.githubusercontent.com/prft-rh/devopsinabox-operator/main/install/clustermonitoring-config.yaml>

# Accelerator Installation

## From OperatorHub

## From Source

# Gitops Prerequisites

Changes to the cluster configuration should always be tracked in source control. You will need to setup 2 new repositories for this operator to work correctly:

* one git repository for the global cluster(s) configuration. For our purpose, the ManagedNamespace yaml files.
* one separate repo per namespace/project you want managed by the operator. For our purpose, the SpringBootApp yaml files.

You will also need a container images repository to store the application images to be deployed to the cluster(s). There will be only one immutable container image built for each application.

NOTE: the reason why you need separate repos for the cluster and the namespaces is so that you can limit what individual teams can do on the cluster by only giving them access to the namespace specific config repo.

# Manager Deployment

Create a DevSecOpsManager resource using https://raw.githubusercontent.com/prft-rh/devopsinabox-operator/main/install/devsecopsmanager.yaml as a starting point:

In the config section:

**repoUrl**: the git repository URL (HTTP) to use to store the global cluster configuration

**repoPath**: is the path to the files in the repo. Usually . (root of repo)

In the git section:

**server**: the canonical URL for the git server. This must be a prefix of the repoUrl config above

# Managed Namespace

You need to declare the Openshift project/namespace where your application(s) will run. The operator will initialize a number of re-usable components inside that namespace to prepare for your applications.

* Create a new project (optional), different from where you installed the DevSecOpsManager resource.
* On the Operators > Installed Operators page, click Devops-in-a-box
* Click the "ManagedNamespaces" tab
* Click the Create ManagedNamespace button
* Go to the YAML view
* Copy the sample in a yaml file (mynamespace.yaml for example, but the filename doesn't matter)

**Edit with your values...**

Pick a metadata.name, can be the same as the namespace, or anything you want. For example mynamespace-managed

In the config section:

**repoUrl**: the git repository URL (HTTP) to use to store the namespace configuration (the future apps)

**repoPath**: leave blank

**repoBranch**: the branch that holds the staging env configuration, assumed master

In the container and git sections, add your credentials and use the canonical server URLs as shown in the example. DO NOT USE your actual password, instead generate a token (in github and quay) that you can easily revoke in case of security leak. See https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token.

The git creds can be the same as the ones used in the DevSecOpsManager config, but preferably different (cluster admins have access to global cluster config repo while individual teams have a dedicated namespace config repo)

git.token should be the same as git.password.

The manager.namespace value is the name of the project where the DevSecOpsManager resource was installed in the previous steps.

mirrors.maven (optional): the URL of a maven mirror to speed up the builds (artifactory, nexus, etc)

**Commit this file to the global cluster repo** (later on, this should be done by pull requests so the configuration changes can be reviewed)

Wait a few minutes and if everything went well, you should see a new Pipeline under the Pipelines menu in the Openshift Web UI.

## Post-install step:

In order for the pipeline to be allowed to pull application source code and images, you need to allow the "pipeline" service account to access the credentials configured in the previous step. The operator will automatically create two secrets for that purpose:

* [managednamespace-name]-devops-in-a-box-namespace-git-auth
* [managednamespace-name]-devops-in-a-box-namespace-image-repo-creds

In the Web UI, go to "User Management" > "Service Accounts" > YAML, and add the two secrets to the secrets section.

# Managed App

For each app you want to deploy, you will need to create a SpringBootApp resource to tell the operator where to find your application source code and where to deploy it.

* On the Operators > Installed Operators page, click Devops-in-a-box
* Click the "SpringBootApp" tab
* Click the Create SpringBootApp button
* Go to the YAML view
* Copy the sample in a yaml file

The name of the file is important here, it needs to be in the form: [yourappname]-spring-boot.yaml where yourappname is whatever you want but needs to match the name of the resource inside the yaml file.

In the config section:

**repoUrl**: the url of the repo containing this file

**repo**: the name of the repo, without the hostname. Ex: myorg/myapp

In the container section:

**image**: leave bank for now

**registry**: the URL of the registry where your application image will be pushed after the build is complete. Ex: https://quay.io/myorg/myapp

In the git section:

**url**: the URL of the git repository where the application source code is. Must be a http(s):// URL.

**branch**: the branch that you want to build and deploy

**module**: leave blank

***Delete the version section for now.***

**management.port** is the port used by the actuator. It should be different from the service port to avoid exposing actuator data outside of the cluster, but can be anything else.

**service.port** is the port used to serve the APIs. Default to 8080 in Spring Boot.

***Everything else can be ignored for now.***

Commit that file into the managed namespace config repo.

If everything went well, you should soon see a new service in the Networking > Services page of the Web UI

# CI/CD

At this point all you have to do is setup the git webhook to trigger the first build. In the Web UI, go to the Networking > Routes section and look for a route called el-[yourappname]-spring-boot. Copy the Location link.

Add that location as a webhook in your git repo configuration, and make sure you select the JSON format for the webhook payload.

Commit something to your code repo:

*git commit --allow-empty -m "Triggering Pipeline"*

*git push*

On the Pipelines > Pipeline Runs page, you should see a new pipeline starting. Once completed, your application will be deployed automatically.

Go to the Workloads > Deployment Configs page to see the status of the application deployment.

Once complete, your application can be accessed outside of the cluster by going to the route created by the operator. Go to the Networking > Routes section and look for the [yourappname]-spring-boot route location.

# Promoting Builds

To promote a build to production, you will need to setup a second cluster the same way.

* In the global cluster config repo, create a new branch called prod
* For each managed namespace yaml, make sure to specify “prod” as the config.repoBranch as well
* On the second cluster, create the DevSecOpsManager resource and change the config.branch to prod

After a few minutes, your second cluster should be running all the same applications as the staging cluster.

## Promotion Flow

When a build is successful on staging, the pipeline creates a Pull Request in the prod branch of the managed namespace config repo. Simply merge that PR when you’re ready to upgrade the production cluster to run the new version of the app.

## Versioning

If you have Service Mesh installed, you will be able to leverage Istio to managed the multiple versions of your application and use advanced deployment models like blue/green and canary.

Extra Steps:

Install Service Mesh

Create a Gateway

…