Pipeline Example: Helm Blue-Green Deploy

Overview

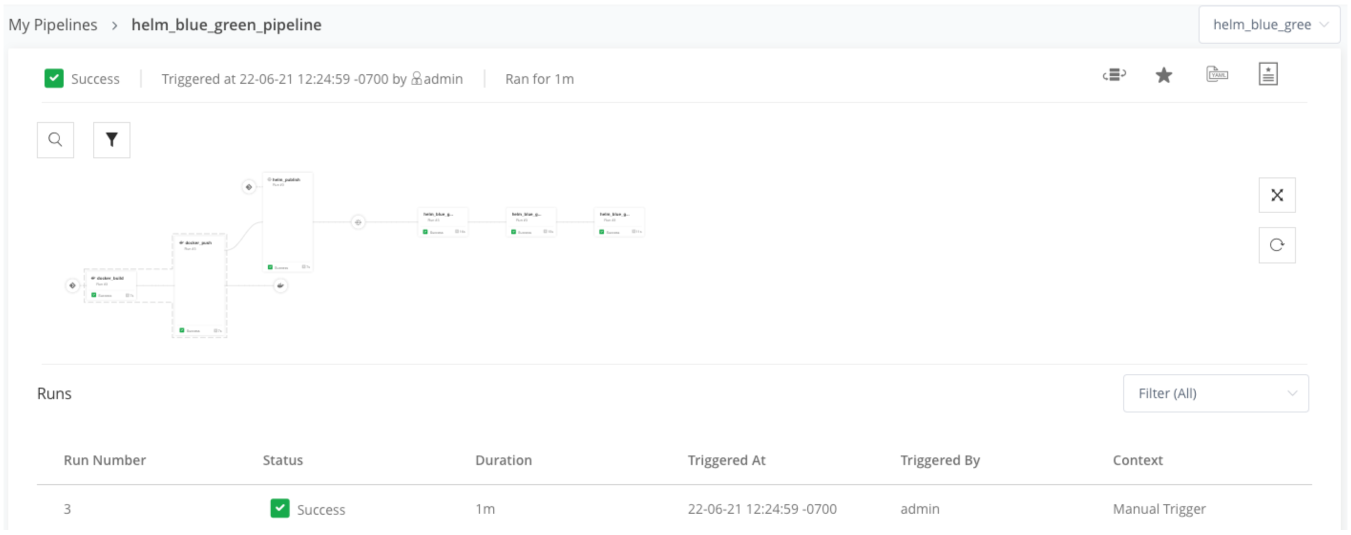
This pipeline demonstrates the definition of a simple pipeline that builds a Docker Image and a Helm Chart and deploys it to a Kubernetes cluster using the Blue/Green strategy. An example Pipelines DSL is used to show how to use integrations, resources, and steps to construct a simple, automated workflow.

The pipeline performs the following sequence of tasks:

- Build and publish a Docker image using DockerBuild and DockerPush native steps
- Publish a Helm Chart using the HelmPublish native step
- Deploy the Helm Chart to Live using the HelmBlueGreenDeploy native step
- Promote Helm Release to Live role using the HelmBlueGreenRoleSwitch native step
- Uninstall previous release using the HelmBlueGreenCleanup native step

Page Contents

A successful run of the pipeline in this Quickstart looks like this:



My Pipelines > helm_blue_green_pipeline
helm_blue_gree
Success | Triggered at 22-06-21 12:24:59 -0700 by admin | Ran for 1m
docker_build
docker_push
helm_publish
Runs
Filter (All)

Run Number	Status	Duration	Triggered At	Triggered By	Context
3	Success	1m	22-06-21 12:24:59 -0700	admin	Manual Trigger

Before you Begin

Before trying this example, ensure that you have:

- A GitHub account. This is required for forking the sample repository.
- A JFrog Platform account or self-hosted JFrog Pipelines.
- At least one node pool. This is the set of nodes that all pipeline steps will execute in. For more information, see Managing Pipelines Node Pools.

  If you have a Cloud account, a node pool will already be available as part of your subscription.

Running This Example

Follow the steps below to build your Go binary:

1. **Fork the repository**
   This Pipelines sample is available in the jfrog-pipelines-helm-blue-green-sample repository in the JFrog GitHub account. The configuration is included in YAML files at the root of the repository:

- `pipelines.yml`, which contains the declarations for all the resources and steps required to run the pipeline. This configuration is written in template format and you do not have to change anything in this file.
- `values.yml`, which contains custom values that will be populated into the template to create your pipeline.

Fork this repository to your account or organization. This is important since you need admin access to the repositories that are used as in your pipelines to enable Pipelines to add webhooks to these repositories and listen for change events.

2. **Sign in to Artifactory**

Sign in to JFrog Platform with your Artifactory credentials.

3. **Create the required repositories**

Create the following repositories that will be used in your pipeline configuration:

- **docker-local**: A local Docker repository where your image will be published.
- **docker-remote**: A remote Docker repository that proxies https://registry-1.docker.io/.
- **docker**: A virtual Docker repository that aggregates local and remote repositories and is used in your pipeline definition to resolve dependencies. Ensure that you select your local and remote repositories in the **Repositories** section while creating this virtual repo. Also, ensure that you select your local repository as the **Default Deployment Repository**.
- **helm-local**: A local Helm repository where your chart will be published.
- **helm**: A virtual Helm repository that aggregates local and remote repositories and is used in your pipeline definition to resolve dependencies. Ensure that you select your local repository in the **Repositories** section while creating this virtual repo. Also, ensure that you select your local repository as the **Default Deployment Repository**.

4. **Add Integrations**

a. Go to **Administration | Pipelines | Integrations** to add three integrations:

- **GitHub Integration**: This integration is used to add the Pipeline source, as well as the GitRepo resource.
- **Artifactory Integration**: This integration is used to authenticate with Artifactory to resolve dependencies and to publish the built binary to Artifactory.
- **Kubernetes Integration**: This integration is used to authenticate with the Kubernetes cluster to deploy the chart and create role Services.

b. Write down the names of all integrations as these are required for the next step. Ensure that the names are unique and easy to remember.

5. **Update pipeline definitions**

Since your `pipelines.yml` config file is templatized, you can just update `values.yml` in your forked repository by following instructions below.

Tag	Description	Example
`gitRepo.gitProvider`	Provide the name of the Github integration you added in step 4.	`gitProvider: my_github`
`gitRepo.path`	Provide the path to your fork of this repository.	`path: myuser/jfrog-pipelines-helm-blue-green-sample`
`gitRepo.branch`	Provide the branch of your fork to use to resolve the source code.	`branch: main`
`appImage.registry`	Provide the name of the Artifactory integration you added in step 4.	`registry: my_artifactory`
`appImage.registryUrl`	Provide the url to your Artifactory environment.	`registryUrl: myartifactory.myorg.com`
`appChart.sourceArtifactory`	Provide the name of the Artifactory integration you added in step 4.	`sourceArtifactory: my_artifactory`
`runtime.k8s_integration`	Provide the name of the Kubernetes integration you added in step 4.	`k8s_integration: my_k8s`
`runtime.namespace`	Provide the name of the Kubernetes namespace where the Helm chart will be deployed.	`namespace: my_namespace`

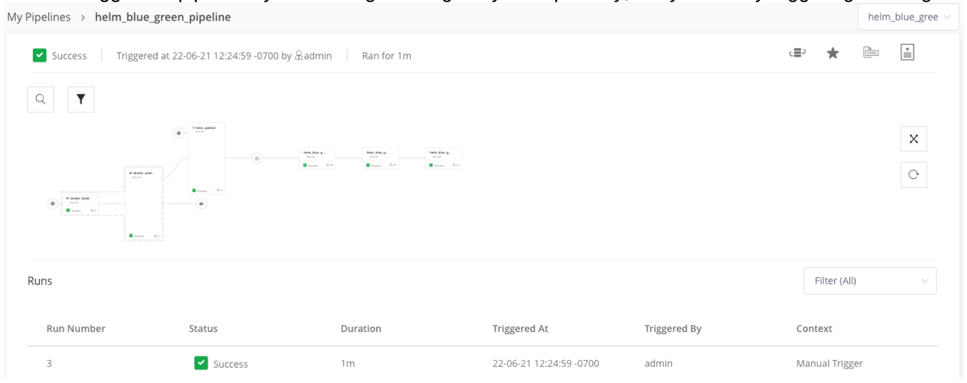
And that's it. Your configuration is ready to go!

All pipeline definitions are global across JFrog Pipelines within a Project. The names of your pipelines and resources need to be unique within the Project in JFrog Pipelines.

6. ## Add Pipeline Sources
   A Pipeline Source represents the git repository where our pipelines definition files are stored. A pipeline source connects to the repository through an integration, which we added in step 4.
   a. In your left navigation bar, go to **Administration | Pipelines | Pipeline Sources**. Click **Add a Pipeline Source** and then choose **From YAML**. Follow instructions to add a Pipeline Source. This automatically adds your configuration to the platform and pipelines are created based on your YAML.

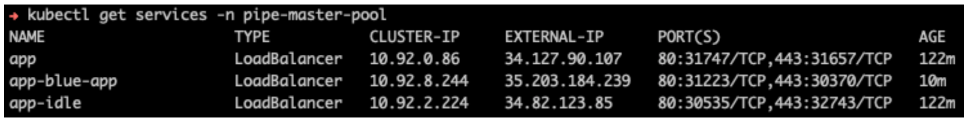
      Ensure that the **Repository Full Name** points to your forked repository and the **Pipeline Config File Filter** is entered as `(pipelines|values).yml` so that both your config files are included.
   b. After your pipeline source syncs successfully, navigate to **Pipelines | My Pipelines** in the left navbar to view the newly added pipeline. In this example, `helm_blue_green_pipeline` is the names of your pipeline.
   c. Click the name of the pipeline. This renders a real-time, interactive, diagram of the pipeline and the results of its most current run.

7. ## Execute the Pipeline
   You can trigger the pipeline by committing a change to your repository, or by manually triggering it through the UI.


My Pipelines > helm_blue_green_pipeline
helm_blue_gree
Success | Triggered at 22-06-21 12:24:59 -0700 by admin | Ran for 1m
Runs
Filter (All)

Run Number	Status	Duration	Triggered At	Triggered By	Context
3	✓ Success	1m	22-06-21 12:24:59 -0700	admin	Manual Trigger

8. ## Success!
   You have successfully executed the Helm Blue/Green pipeline! You can verify the results by viewing the Services deployed to your Kubernetes cluster.



```
→ kubectl get services -n pipe-master-pool
NAME                 TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)                      AGE
app                  LoadBalancer   10.92.0.86    34.127.90.107    80:31747/TCP,443:31657/TCP   122m
app-blue-app         LoadBalancer   10.92.8.244   35.203.184.239   80:31223/TCP,443:30370/TCP   10m
app-idle             LoadBalancer   10.92.2.224   34.82.123.85     80:30535/TCP,443:32743/TCP   122m
```

Explanation of Pipeline Definition

Let us now take a look at the pipeline definition files and what each section means.

The **pipelines.yml** file contains the templatized definition of your pipeline. This consists of the following:

- Resources are entities that contain information that is consumed or generated by pipeline steps. In our example, we use the following resources:
  - A GitRepo resource pointing to the source control repository where your application code is present. You can configure this resource to trigger dependent steps on specific events.
  - An Image resource that adds a reference to a Docker image to your pipeline.
  - A HelmChart resource that adds a reference to a Helm Chart to your pipeline.
- Steps are executable units that form your pipeline. In our example, the pipeline consists of the following steps:
  - A DockerBuild native step that builds your Docker image based on a Dockerfile present in the git repo.
  - A DockerPush native step that published the built image to Artifactory.
  - A HelmPublish native step that packages and publishes your Helm Chart to Artifactory.
  - A HelmBlueGreenDeploy native step that deploys the Helm Chart to your Kubernetes cluster using the Blue/Green deployment strategy. The chart will be deployed using the details of the release playing the Idle role and the configured Idle role Services will be create or updated. Any final stage validation to ensure the new version is ready to be available to users could be added between this step and the next one.
  - A HelmBlueGreenRoleSwitch native step that promotes the deployed Helm release to the Live role in the Blue/Green strategy. The configured Live role Services will be created or updated. Any after release validation to ensure the new release is successful could be added after this step. A copy of this step could be added downstream to rollback the release in case the validation is not successful.
  - A HelmBlueGreenCleanup native step the uninstalls any previously deployed Helm release.