DevOps Reference Implementation using a contemporary CI/ CD toolchain

Table of Contents

[1. Abstract 1](#_Toc79661442)

[2. Tools Used 2](#_Toc79661443)

[2.1 Continuous Integration Tools 2](#_Toc79661444)

[2.2 Continuous Deployment Tools 2](#_Toc79661445)

[2.3 Continuous Monitoring Tools 2](#_Toc79661446)

[3. Usage Scenario 2](#_Toc79661447)

[4. DevOps Pipeline Design and Implementation 3](#_Toc79661448)

[4.1 GitHub 3](#_Toc79661449)

[4.2 JFrog Artifactory 3](#_Toc79661450)

[4.3 TeamCity 3](#_Toc79661451)

[4.4 Octopus Deploy 11](#_Toc79661452)

[4.5 Kubernetes 13](#_Toc79661453)

[4.6 Istio 13](#_Toc79661454)

[4.7 Flagger. 14](#_Toc79661455)

[4.8 Splunk 15](#_Toc79661456)

[5. Conclusion: 17](#_Toc79661457)

[6. References: 17](#_Toc79661458)

[7. Appendix: 18](#_Toc79661459)

[7.1 Team City Installation and initial configuration: 18](#_Toc79661460)

[7.2 JFrog Docker Registry setup: 20](#_Toc79661461)

[7.3 Flagger and Istio Setup: 22](#_Toc79661462)

[7.4 Splunk Cloud Setup: 23](#_Toc79661463)

# Abstract

With the advent of microservices architecture, cloud native and hybrid cloud implementation, Continuous Integration, and deployment (CI/CD) has become part of the hygiene of a software application or product. It does not only cover the release management part of a software development lifecycle (SDLC) but also coupled with site reliability engineering, product quality and integrity of the delivered product and resilience of the running application in case of failures. Most of the CI/CD toolchains now-a-days offer comparable capabilities in terms of source code management, build, code coverage and integrity of build through code quality check and continuous testing, monitoring, and collaboration. They also cover various deployment scenarios like blue-green deployment and canary deployment.

**Scope**: This article is intended to provide a reference design and implementation of a DevOps pipeline using a sample Java Microservice based use-case. It does not define any practices, best practices nor comparison of the tools. Here, we have demonstrated how to engineer a DevOps pipeline using GitHub, TeamCity, Octopus Deploy and JFrog Artifactory. Flagger and Istio was used to perform a canary deployment to Google Kubernetes Engine (GKE). Splunk is used as the centralized log monitoring and analysis software. In this exercise the team has used a sample cloud native application, customized and leveraged hybrid instances (local + SaaS) of the toolchain and public cloud account.

# Tools Used

## Continuous Integration Tools

|  |  |
| --- | --- |
| **Tool** | **Description** |
| [Github](https://github.com/) | For Source Code Repository |
| [SonarQube](https://www.sonarqube.org) | Static Code Analysis Tool |
| [TeamCity](https://www.jetbrains.com/teamcity/) | For Build Automation |
| [JFrog](https://jfrog.com) | Artifact Management Tool |

*Table 1: CI Tools*

## Continuous Deployment Tools

|  |  |
| --- | --- |
| **Tool** | **Description** |
| [Octopus](https://octopus.com) | Release Management Tool |
| [Istio](https://istio.io) | Service Mesh |
| [Flagger](https://flagger.app/) | Progressive Delivery Tool which supports Canary deployment |
| [Google Kubernetes Engine](https://cloud.google.com/kubernetes-engine) | Kubernetes Deployment Runtime |
| [Assertible](https://assertible.com/) | Web Application Testing Framework |

*Table 2: CD Tools, cluster platform and continuous test tool*

## Continuous Monitoring Tools

|  |  |
| --- | --- |
| **Tool** | **Description** |
| [Splunk](https://www.splunk.com/) | Centralized log management system |

*Table 3: Continuous Monitoring tools*

# Usage Scenario

The Use Case in consideration for this Reference Implementation covers the following activities, that are typical in a SDLC, post the development activities.

1. Code Commit to a source-code repository
2. Trigger Build
3. Static Code Analysis
4. Compile the code, Build Deployable and Push to Image Repository
5. Trigger Deployment
6. Pull docker image and configuration
7. Deploy to Runtime Engine using Canary deployment
8. Post-deployment Test
9. Feed the logs to a monitoring system
10. Monitor the logs as dashboards in the monitoring system

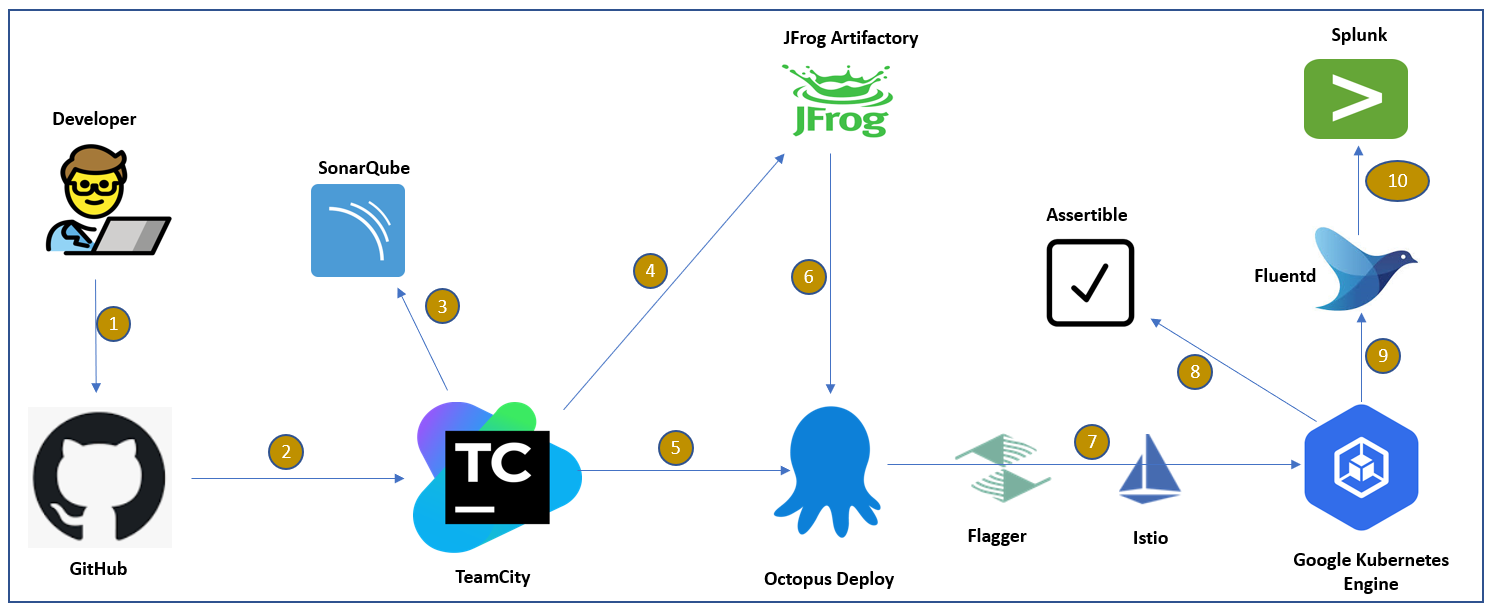


Fig 1: End-To-End CI / CD / CM Toolchain

# DevOps Pipeline Design and Implementation

## GitHub

GitHub free tier has been used as a code repository to store the code base for web application/ microservice.

Repository URL: <https://github.com/DeepakkumarGanesan/devops_reference>

## JFrog Artifactory

There are many flavours available for JFrog Artifactory (including enterprise and cloud version) as listed in <https://jfrog.com/pricing/#sass>. A free cloud instance has been used here which requires just a basic registration, and you are good to go in minutes. In this tutorial, w are using JFrog Artifactory to store Docker Image and configuration file like ConfigMap and Secret.

Follow the below link for setting up JFrog Artifactory as Private Docker Registry:

<https://www.jfrog.com/confluence/display/JFROG/Docker+Registry>

Follow the below link to understand how to integrate JFrog Artifactory with TeamCity:

https://www.jfrog.com/confluence/display/JFROG/TeamCity+Artifactory+Plug-in

## TeamCity

TeamCity comes with two flavours: Server version and a Cloud version. The Cloud version has a free trial, does not allow you to install any external plugins and you have a limitation of data. But server version will allow to integrate with external databases (hsql, mysql, postgreys, oracle etc) and can be integrated with any of the supported plugins as needed.

**Please refer to Appendix 7.1 to install TeamCity and create initial build configuration.**

If you have setup your Teamcity and initial project and build configuration, you can now follow the additional configuration steps with respect to Maven, Docker, SonarQube, Assertible and JFrog Artifactory as mentioned below.

Install plugins:

The following plugins are to be installed

* JFrog Artifactory
* Octopus Deploy
* Sonar Runner

Go to administration ->plugins and install below plugins.

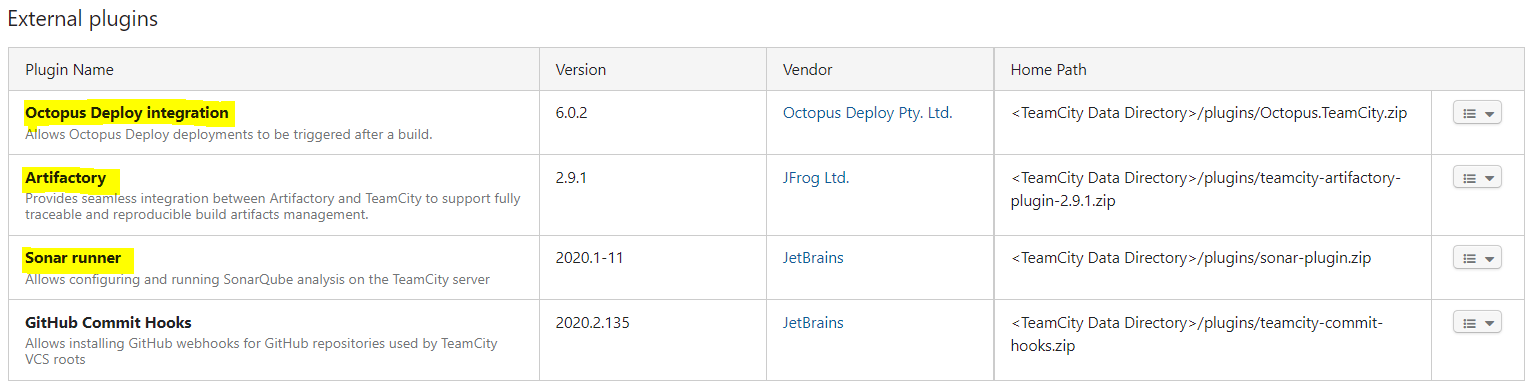


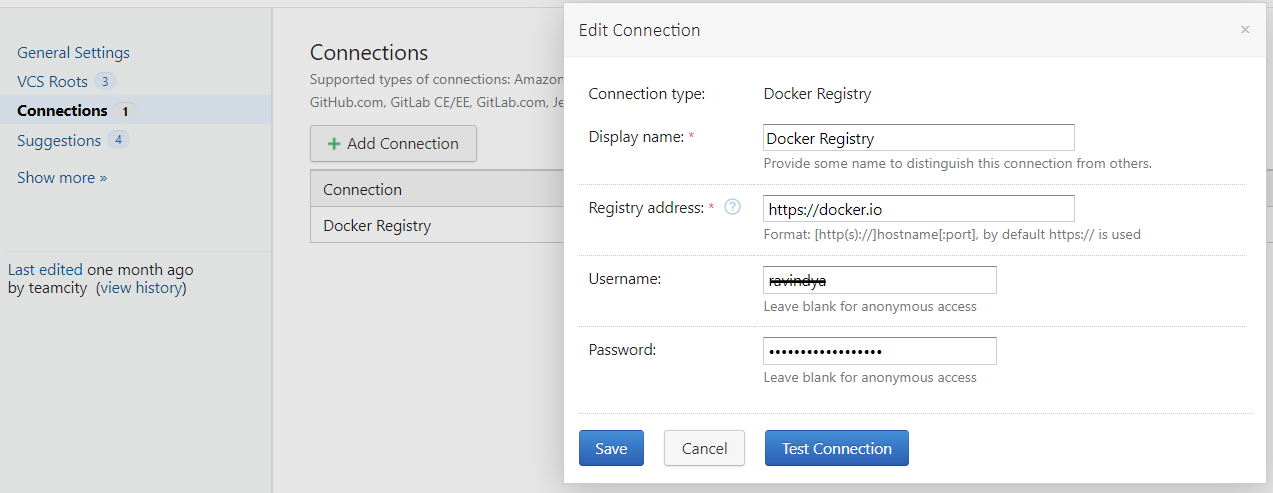
Fig2: TeamCity plugins

Configuring Docker Registry:

**If you need to create docker registry in jfrog artifactory, please refer Appendix 7.2.**

1. Establish a Docker Registry connection:
   1. Setting Docker Hub as Public Docker Registry:

Click on your project name, in the left panel click on connection and then click on “**Add Connection**” and fill the connection details as mentioned below:

Fig3:TeamCity to docker configuration

* 1. Setting JFrog Artifactory as Private Docker Registry:

Go to Administration and under Integrations section on left panel, click on **Artifactory** and then click on “**Create new Artifactory server configuration**” and fill the details as mentioned below:

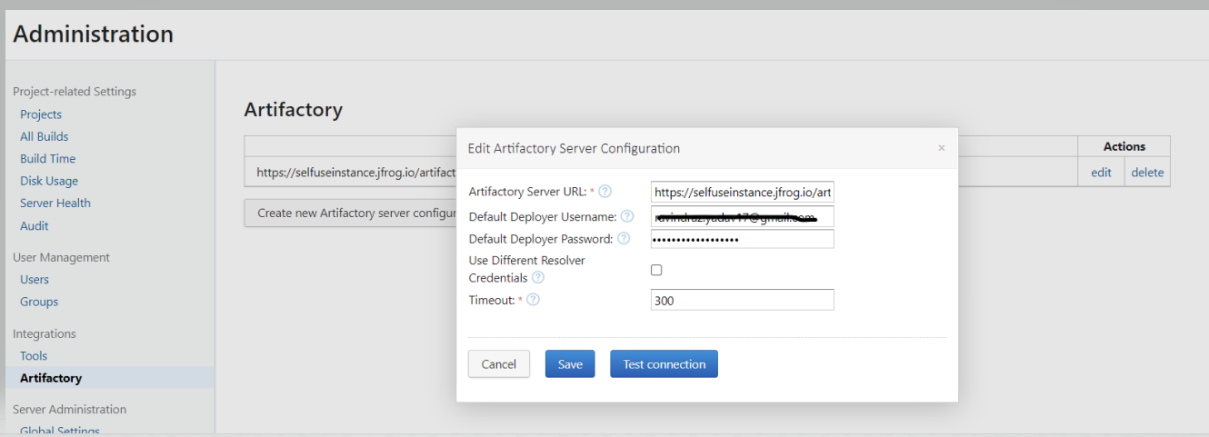


Fig4: JFrog Artifactory configuration

Create Build configurations

If you click on edit project settings on your project, you will see an existing build configuration already there, we have to click on build configuration and add below build steps.

1. Maven Build:

It will checkout the code from GitHub, compile it and create a package. Package created will be present in the working directory of the project (check the build logs to identify the current working directory of the project)

Graphical user interface, text, application

Description automatically generated

Fig5: Maven build step

1. Code Inspection/Analysis :

Go to project configuration and add sonar server as follows.

Graphical user interface, application, Word

Description automatically generated

Fig6: SonarQube server build step

Now you go to project build configuration, add next build step with runner type as sonar runner and add below details.

A picture containing text

Description automatically generated

Fig7: SonarQube runner build step

Once your build will run, you will be able to see your code analysis report at your sonar dashboard.

1. Building Docker Image from Dockerfile:

We need Dockerfile to build Docker image containing the package built by Maven. Each project has its own Dockerfile, hence it is stored in the GitHub repository as part of the code base.

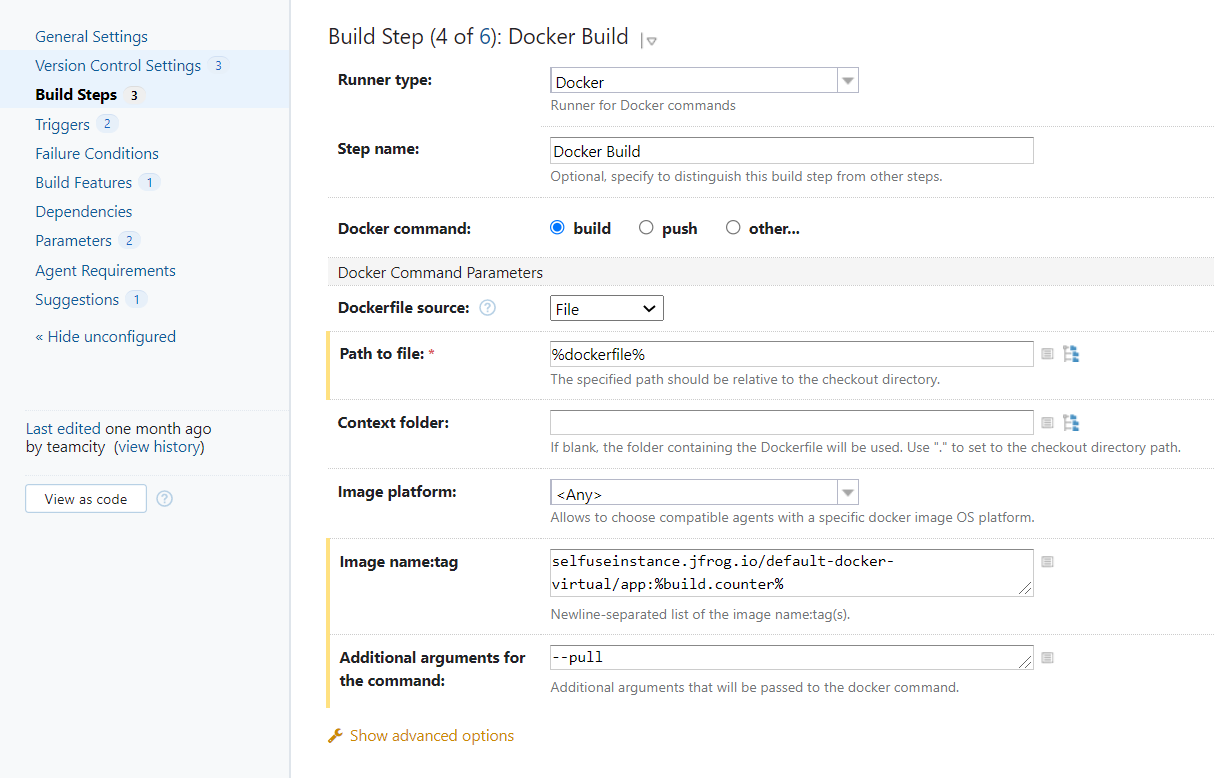


Fig8: Docker build step

1. Pushing Docker Image in Private Docker Registry:

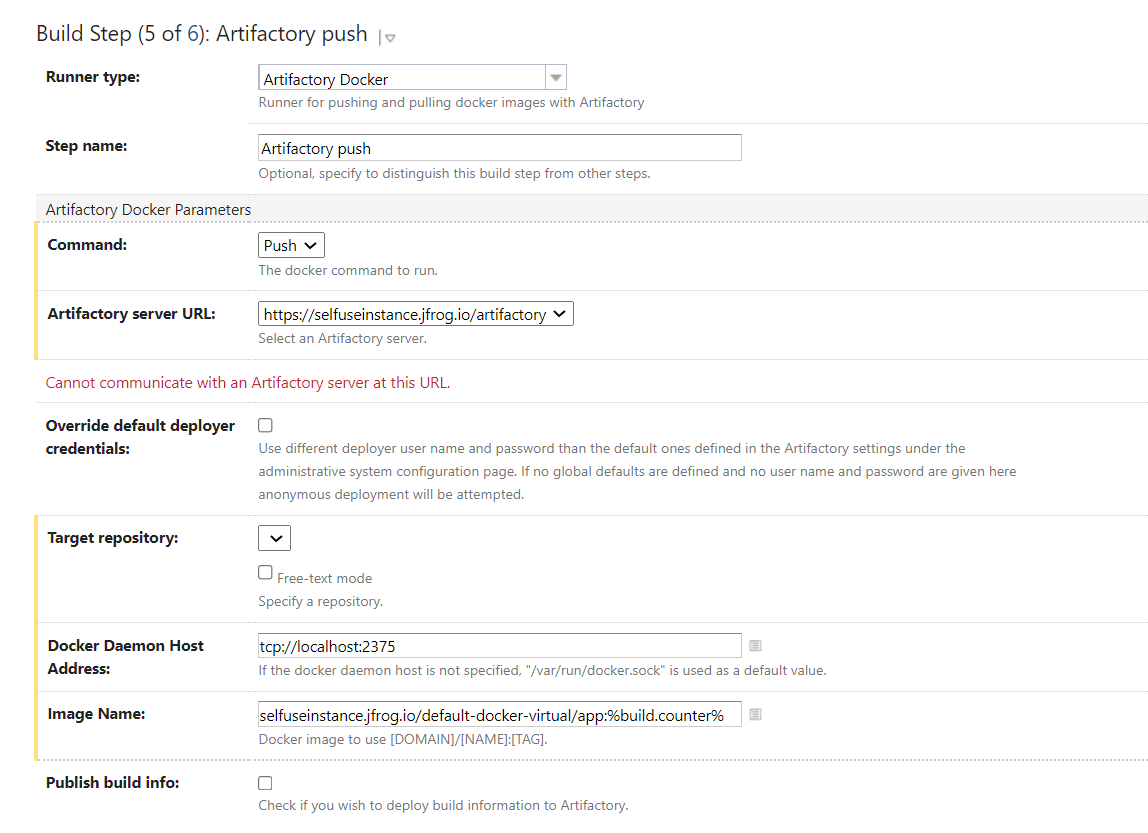


Fig9: Pushing image to JFrog

Create Release:

In later section you will find Octopus URL and API key to access your Octopus Instance. You must create a release to deploy new version of war to your target environment through Octopus.

Graphical user interface, text, application, email

Description automatically generated

Fig10: Create release to Octopus

Create Deployment:

Once you created the release, you can start the deployment to Octopus via below step.

Text

Description automatically generated

Fig11: Trigger deployment to Octopus

Make sure you check this box below, then it will wait for the deployment to be completed to go to next steps.

Text, email

Description automatically generated

Fig12: Wait for deployment to complete

Configure Assertible

Assertible Tool has been used to perform post-deployment test of the services. This is one of the best tools in the market which comes free for personal usage plan and it can test website, APIs and HTTP service easily.

*Note: The service that we will be deploying to Kubernetes as part of the build-deploy pipeline will have a dynamic-IP address after each deployment. To overcome this limitation, we have configured Assertible to test against a service with a dummy static endpoint (IP). This should serve the purpose of demonstrating the required setup, and the capability of the tool. In a real use case, a similar implementation can be used, by providing the correct target URL (could be a cluster URL, that will be static endpoint)*

Please follow below steps to configure your assertible dashboard:

1. Go to <https://assertible.com/login> and sign up with GitHub.

2. Go to services and create new service.

Graphical user interface, text, application

Description automatically generated

Fig13: Assertible configuration

3. put your service URL (e.g., http://<public Ip>/sample-endpoint) in import URL section and create the service. Please note you have to access your application through public IP of your GKE cluster where your app/service has been deployed.

Graphical user interface, text, application, email

Description automatically generated

Fig14: Adding service to assertible

4. Now your service is configured, more complex tests can be configured if required. Here we just want to ensure the URL is working, post deployment. So, we can just trigger the assertible test from TeamCity with no specific method to test.

5. Now go to settings of the service that you created and capture the trigger url with curl command.

Graphical user interface, text, application, email

Description automatically generated

Fig15: trigger url configuration of service

6. Put the below command in TeamCity build pipeline, choose build step as runner type=command line and command is as copied from assertible.

Text

Description automatically generated with medium confidence

Fig16: Unit test configuration in TC

Once our build pipeline is run, we can check the API status from assertible dashboard as following. If it returns 200, our deployment is successful, and site is up.

Graphical user interface, text, application, email

Description automatically generated

Fig17: Assertible Dashboard

## Octopus Deploy

Octopus Deploy has two parts.

1. Octopus Server where the application or service (that needs to be deployed) is added and configured.
2. Octopus Tentacles, which are installed on the set of machines where the applications configured on Octopus Server need to be deployed.

Both the Octopus server and tentacles are installed as a service.

Setup:

We have used Octopus cloud instance (<https://octopus.com/docs>) has been used which is provided by Octopus hosted in cloud as free trial.

The configuration of different environments namely Production and then adding machines or cloud locations within those respective environments is completed. Octopus server is then connected to tentacles on these machines via Thumbprints which make the connection secure and break-free.

For more details on setting up GKE as deployment target on Octopus Deploy refer to reference section.

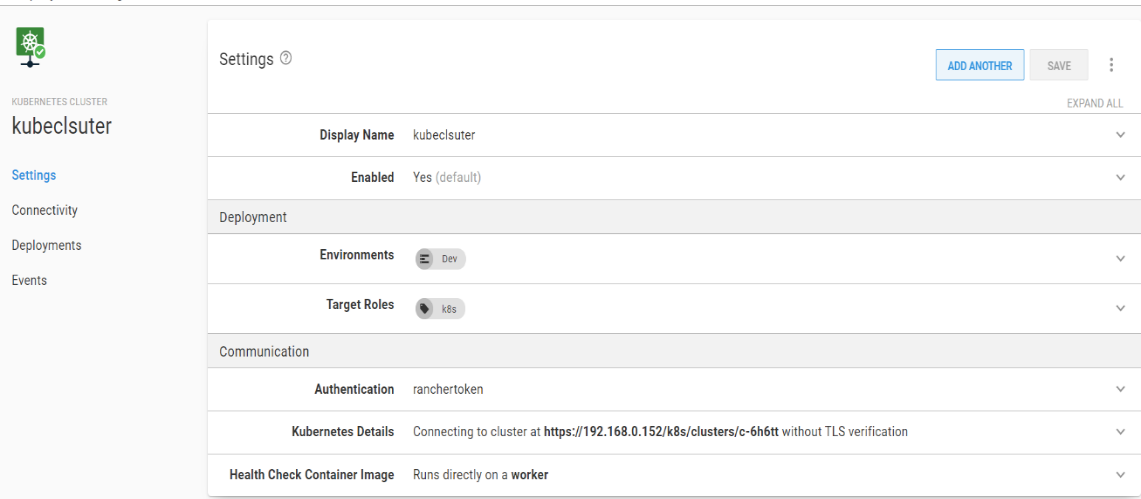


Fig18: Kubernetes cluster configuration

Linking JFrog Artifactory with Octopus Deploy:

* Configure JFrog Artifactory as External Feeds in Octopus library as shown below.

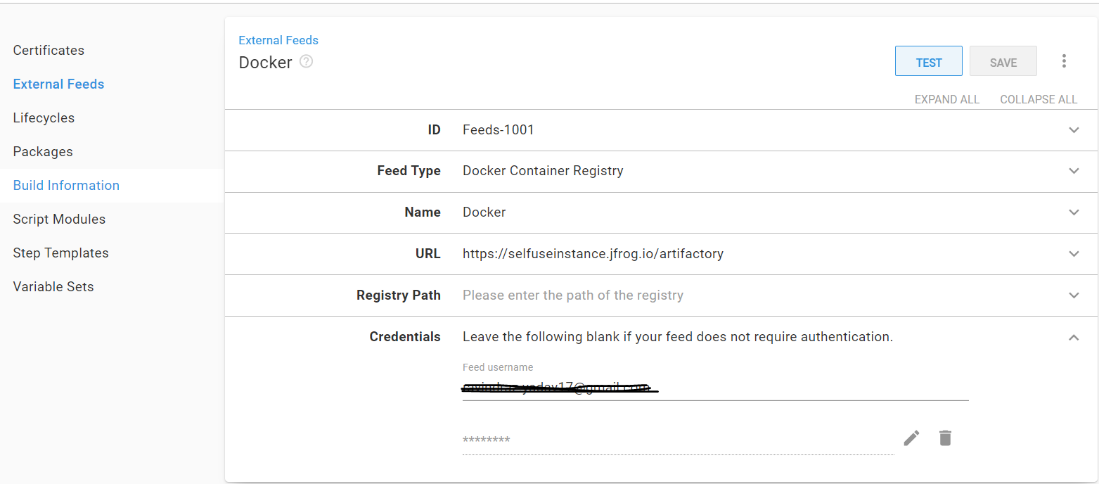


Fig19: JFrog configuration in octopus

Deployment Pipeline in Octopus Deploy:

Following are the steps, which will be performed by the deployment pipeline: -

1. Check whether the microservice is getting deployed for first time. If yes, deploy it in Production environment or else skip to step 2.
2. If microservice is already deployed, then deploy the new release of the microservice using Canary deployment methodology.
3. Initially split the traffic between new and old pod in 25% and 75% ratio respectively. If all good, proceed to step 4 or else rollback the change.
4. Split the traffic between new and old pod in 50% and 50% ratio respectively. If all good, proceed to step 5 or else rollback the change.
5. Split the traffic between new and old pod in 75% and 25% ratio respectively. If all good, proceed to step 6 or else rollback the change.
6. Transfer the 100% traffic to new pod. If all good, delete the old pod or else rollback the change.

Please note that the step 3 to step 6 (i.e diverting traffic to new Pod in increments) are automated using Flagger tool. For more details on Flagger and its implementation, see the section 5.6 and 5.7.

Scripts:

|  |  |
| --- | --- |
| **Step** | **Script Location in Git** |
| 1. **Check if package is getting deployed for the first time or not** | [check](https://github.com/DeepakkumarGanesan/devops_reference/blob/main/deployment_scripts/check_first.ps1)\_[first](https://github.com/DeepakkumarGanesan/devops_reference/blob/main/deployment_scripts/check_first.ps1).ps1 |
| 1. **Deploy the first release in Production**   This script will run only if the microservice/application is getting deployed first time in production | [deploy\_stable.ps1](https://github.com/DeepakkumarGanesan/devops_reference/blob/main/deployment_scripts/deploy_stable.ps1) |
| 1. **Deploy the new release in Production**   This script will be executed only when the package getting deployed is a new version of the already deployed package on the Google Kubernetes Engine. | [deploy\_new.ps1](https://github.com/DeepakkumarGanesan/devops_reference/blob/main/deployment_scripts/deploy_new.ps1) |

*Table4: Script path*

## Kubernetes

Here we have used [Google Kubernetes Engine](https://cloud.google.com/kubernetes-engine) service provided by Google. For details on how to setup GKE, see the installation part in the section 5.7.

## Istio

The Istio mesh helps in implementing the canary deployment by allowing fine-grained traffic control that decouples traffic distribution and management from replica scaling. Instead of manually controlling replica ratios, you can define traffic percentages and targets, and Istio will manage the rest.

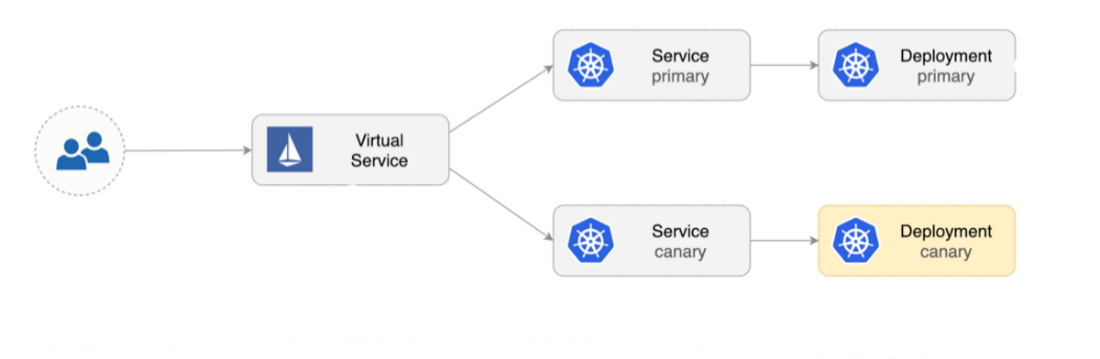


Fig20: Traffic Distribution

**For more details on how to setup Istio on GKE, see the installation part in the Appendix 7.3**

## Flagger.

[Flagger](https://github.com/fluxcd/flagger) is a progressive delivery tool that automates the release process for applications running on Kubernetes. It reduces the risk of introducing a new software version in production by gradually shifting traffic to the new version while measuring metrics and running conformance tests.

**To set up flagger on GKE, please continue to follow Appendix 7.3.**

Flagger takes a Kubernetes deployment and optionally a Horizontal Pod Autoscaler (HPA), then creates a series of objects (Kubernetes deployments, Cluster-IP services, Istio destination rules and virtual services). These objects expose the application inside the mesh and drive the canary analysis and promotion.

Diagram

Description automatically generated

Fig21: Deployment roll back or promotion architecture and route distribution

When the canary analysis starts, Flagger will call the pre-rollout webhooks before routing traffic to the canary. The canary analysis will run for set time while validating the HTTP metrics and rollout hooks every minute.

## Splunk

[Splunk](https://www.splunk.com/en_us/resources.html)is a software platform widely used for monitoring, searching, analysing and visualizing the machine-generated data in real time.

**Please refer in Appendix 7.4 for Splunk Cloud setup.**

Splunk Connect:

We used Splunk Connect for Kubernetes to stream the logs to Splunk from Kubernetes. It is a collection of Helm charts that deploy a Splunk-supported deployment of Fluentd to your Kubernetes cluster. It includes a Splunk-built Fluentd HEC plugin to ship logs and metadata into Splunk

Steps to Integrate Splunk and Kubernetes Cluster:

1. Create Indexes:

Go to Settings -> Indexes -> New Index. Give Index name and other mandatory fields and click on save button. Index will be created.

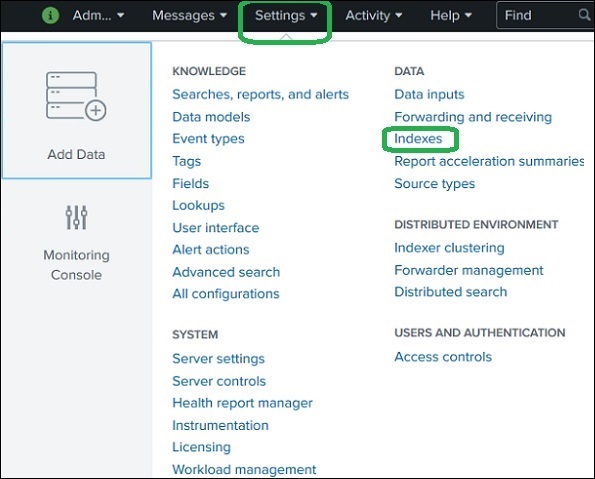


Fig22: Creating Splunk indexes

For our Use-Case we need 3 indexes such as Events, Metrics and Objects.

1. Configure HTTP Event Collector (HEC):

Go to Setting -> Data Inputs -> HTTP Event Collector. Create a new HEC and add the indexes in the sources and in the last page copy the HEC token.

Graphical user interface, application, email

Description automatically generated

Fig23: Creating Splunk token

**Graphical user interface, text, application, Teams

Description automatically generated**Fig24: Creating HEC configuration

In HEC Page we see need to configure the Global Settings as well, where all Tokens to be Enabled and we need to enable SSL and Save.

1. Steps to be done in Kubernetes:

* Curl command to connect Http Event Collector

curl -k https://prd-p-nc3f0.Splunkcloud.com/services/collector/event -H "Autorization: Splunk <HEC\_Token>" -d '{"event": "hello Splunk"}'

* Create a Splunk-values.yaml file with the indexes details

Sample Script:

<https://github.com/DeepakkumarGanesan/devops_reference/blob/main/splunk-values.yaml>

* Helm command to stream the logs to Splunk indexes and pass Splunk-values.yaml as input to the command.

helm install kube-Splunk –namespace Splunk –f Splunk-values.yaml Splunk/Splunk-connect-for-kubernetes

This will start the log stream to our Splunk.

1. Splunk Visualization:

With the events in the indexes, we can create reports, tables, charts and more.

Below is our Dashboard created with the log events.

Graphical user interface, application

Description automatically generatedFig25: Splunk Dashboard

Also, Splunk provides out of the box Application called “[Splunk App for Infrastructure](https://www.splunk.com/en_us/blog/it/splunk-app-for-infrastructure-1-4-all-about-the-containers.html)” for Kubernetes cluster, which provides visibility into the performance of your entire containerized and microservice environment, out-of-the-box dashboards for easy troubleshooting, and faster investigation time to resolve issues.

Please note, since *Splunk App for Infrastructure* cannot be installed on Splunk Cloud Trial Version, so, we couldn't showcase its feature as part of this implementation.

# Conclusion:

This reference implementation is an end-to-end automated pipeline with all required tools integrated to release a web application - from code upload at code repository all the way to deployment to a target environment and monitoring of the environment. The tools that have been used are mostly opensource and are one of the few options available for each capability. The pipeline can be built using other tools as well and the design of the pipeline may also wary, based on the project requirements.

# References:

TeamCity: [https://www.jetbrains.com/help/TeamCity/TeamCity-documentation.html](https://www.jetbrains.com/help/teamcity/teamcity-documentation.html)

Octopus: <https://octopus.com/docs/guides>

JFrog: <https://jfrog.com/artifactory/>

GKE configuration: <https://octopus.com/blog/deploy-to-rancher-with-octopus>

Flagger & Istio: <https://docs.flagger.app/tutorials/istio-progressive-delivery>

Splunk Connect: <https://faun.pub/logging-in-kubernetes-using-splunk-c2785948fdc0>

# Appendix:

## 7.1 Team City Installation and initial configuration:

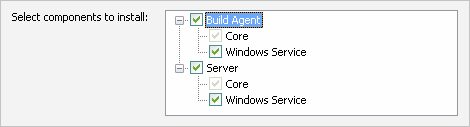
The following steps are followed from the TeamCity documentation

**Installation**

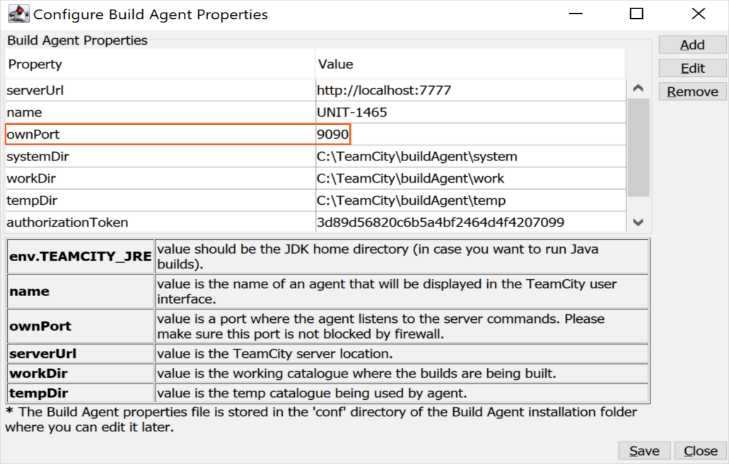
* Go to <https://www.jetbrains.com/teamcity/download/#section=on-premises> ,the full-featured TeamCity bundled with 3 build agents with a limit of 100 build configurations.
* Run the downloaded .exe file and follow the instructions of the TeamCity Setup wizard. The TeamCity web server and one build agent will be installed on the same machine.
* During installation, you can configure:

[TeamCity Home Directory](https://www.jetbrains.com/help/teamcity/teamcity-home-directory.html) where TeamCity will be installed.

whether the TeamCity server and agent will run as Windows services



* ports:
  + Server port: 8111 is the default port, which can be already used by other applications (for example, Skype). Change the server port if it is already in use. In the example below, we've set the port to 7777.
  + Agent port: 9090 is the default for incoming connections from the server. If this port is already in use, you'll be asked to change it by setting the ownPort property to a different value.



If the TeamCity server is installed as a Windows service, follow the [usual procedure](https://bit.ly/2yJF87R) of starting and stopping services. Otherwise, to start/stop the TeamCity server and one default agent at the same time, use the runAll script, provided in the <TeamCity Home>\bin directory:

* to **start** the server and the default agent, use

.\runAll.bat start

* to **stop** the server and the default agent, use

.\runAll.bat stop

If you did not change the default port (8111) during the installation, the TeamCity web UI can be accessed at [http://localhost](http://localhost/) in a web browser running on the same machine where the TeamCity server is installed. Otherwise, use http://localhost:<port> (<http://localhost:8100>  in our case).

**Initial Configuration:**

For configuration of TeamCity; A project first needs to be created and then we can modify our configurations as per our use case.

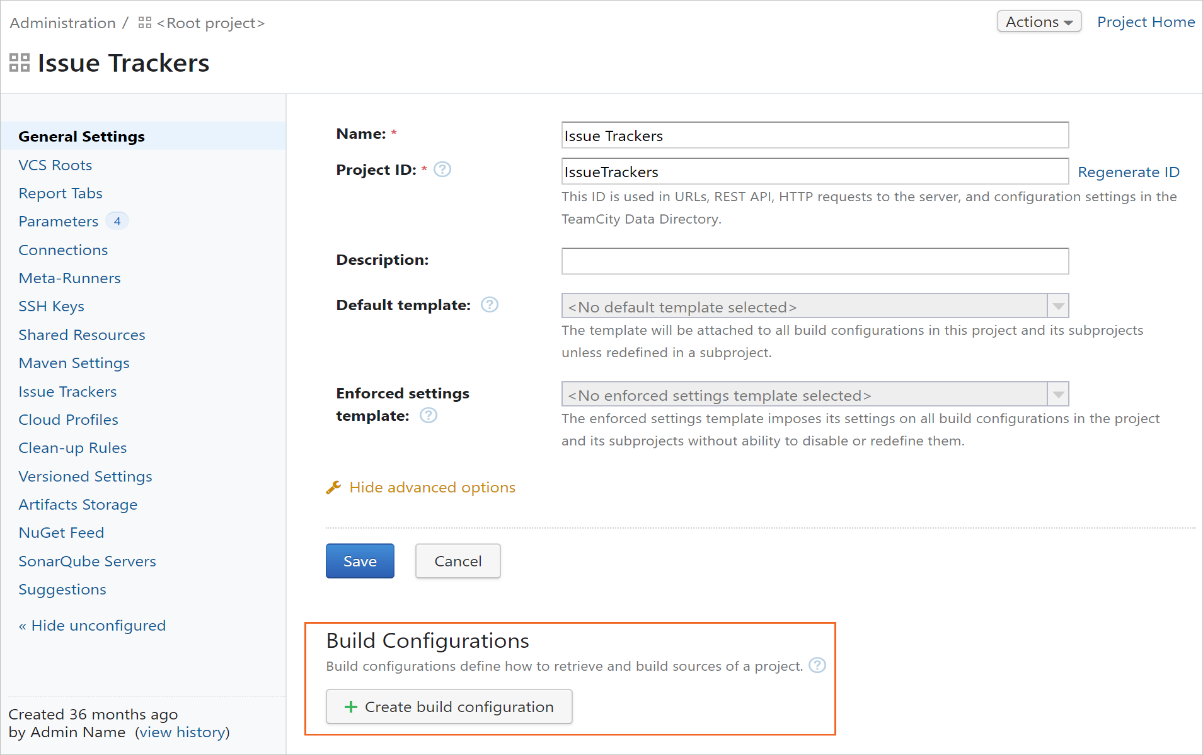


Fig: TeamCity Overview

Create project and initial build configuration:

1. On the Create project page, select to create a project from GitHub.com.
2. Select a repository. TeamCity will verify the repository connection. If the connection is verified, the new page opens.
3. TeamCity will display the project and build configuration name. If required, modify the names and click Proceed. TeamCity will autodetect the default Git branch, but you have an option to change it and to add other branches to monitor by entering their [specification](https://www.jetbrains.com/help/teamcity/working-with-feature-branches.html#Configuring+branches).
4. TeamCity will add a VCS build trigger and attempt to autodetect build steps: Ant, NAnt, Gradle, Maven, MSBuild, Visual Studio solution files, PowerShell, Xcode project files, Rake, and IntelliJ IDEA projects.  
   On the Auto-detected Build Steps page, select the detected step(s) to use in your build configuration. Click Use selected.  
   If no steps found, you will have to [configure build steps manually](https://www.jetbrains.com/help/teamcity/configuring-build-steps.html).
5. Your project and a build configuration are configured. Click Run to start the build. Depending on the build configuration settings, TeamCity can suggest some additional configuration options. Review *Suggestions* at the end of the settings list and configure required ones.

6. To create a build configuration manually, open **General Settings** of a project and click **Create build configuration** under the **Build Configurations** section.



## 7.2 JFrog Docker Registry setup:

To use JFrog Artifactory as the Docker registry, login into your JFrog instance and click on **Quick Setup** and then on **Docker** icon.

Graphical user interface, application

Description automatically generated

Now, Select **Docker** in the Type dropdown and **default-docker-virtual** in the Repository dropdown and the click on “Set Me Up” button.

Graphical user interface, application

Description automatically generated

Copy all the command mentioned in the Configure, Deploy and resolve section.

Graphical user interface, text, application

Description automatically generated

## 7.3 Flagger and Istio Setup:

To install/enable Istio on GKE, run the below command in the GKE console:

|  |
| --- |
| K8S\_VERSION=$(gcloud container get-server-config --format=json \  | jq -r '.validMasterVersions[0]')  gcloud beta container clusters create istio \  --cluster-version=${K8S\_VERSION} \  --zone=us-central1-a \  --num-nodes=2 \  --machine-type=n1-highcpu-4 \  --preemptible \  --no-enable-cloud-logging \  --no-enable-cloud-monitoring \  --disk-size=30 \  --enable-autorepair \  --addons=HorizontalPodAutoscaling,Istio \  --istio-config=auth=MTLS\_PERMISSIVE |

To install Flagger on GKE, run the below command in the GKE console:

Login into Google Cloud, create a project and enable billing for it. Install the gcloud command line utility and configure your project with gcloud init.

Set the default project, compute region and zone (replace PROJECT\_ID with your own project):

|  |
| --- |
| gcloud config set project PROJECT\_ID gcloud config set compute/region us-central1 gcloud config set compute/zone us-central1-a |

Set up credentials for kubectl:

gcloud container clusters get-credentials istio

Create a cluster admin role binding:

kubectl create clusterrolebinding "cluster-admin-$(whoami)" \  
--clusterrole=cluster-admin \  
--user="$(gcloud config get-value core/account)"

The GKE Istio add-on does not include a Prometheus instance that scrapes the Istio telemetry service. Because Flagger uses the Istio HTTP metrics to run the canary analysis you have to deploy the following Prometheus configuration that’s similar to the one that comes with the official Istio Helm chart.

kubectl -n istio-system apply -f \  
<https://storage.googleapis.com/gke-release/istio/release/1.0.6-gke.3/patches/install-prometheus.yaml>

Add Flagger Helm repository:

helm repo add flagger [https://flagger.app](https://flagger.app/)

Deploy Flagger in the istio-system namespace with Slack notifications enabled:

helm upgrade -i flagger flagger/flagger \  
--namespace=istio-system \  
--set metricsServer=http://prometheus.istio-system:9090 \  
--set slack.url=https://hooks.slack.com/services/YOUR-WEBHOOK-ID \  
--set slack.channel=general \  
--set slack.user=flagger

You can install Flagger in any namespace as long as it can talk to the Istio Prometheus service on port 9090.

Flagger comes with a Grafana dashboard made for canary analysis. Install Grafana in the istio-system namespace:

helm upgrade -i flagger-grafana flagger/grafana \  
--namespace=istio-system \  
--set url=http://prometheus.istio-system:9090 \  
--set user=admin \  
--set password=change-me

## 7.4 Splunk Cloud Setup:

Splunk is available in two different forms such as Splunk Cloud and Splunk Enterprise. For our use case we used Splunk Cloud.

To set up it, please go to following link <https://www.splunk.com/en_us/download/splunk-cloud.html>

Fill all the details in the account creation form and create your free trial account for Splunk Cloud.

Graphical user interface, application, website

Description automatically generated

Fig: Creating Free Trail account for Splunk Cloud

You will receive an email with your Splunk instance URL and the Password for login first time. Please note the instance will be available for 14 days only.