Broadcom CI/CD

Official CI Documentation

Official CD Documentation

URLs necessary for GTO Projects

SEO Github

GTO Github

SEO CI Jenkins

GTO CI Jenkins

SEO CD Dev

SEO CD Github Dev

SEO CD Prod

SEO CD Github Prod

GTO CD

GTO CD Github

Jfrog Artifactory SEO

Jfrog Artifactory GTO

Sonarqube SEO

Sonarqube GTO

Blackduck

CI/CD Pipeline

The CI/CD pipeline's main goal is to make it easy to send our application, which is packaged as a container, from our storage system (Broadcom Artifactory) to a specific kind of environment (Kubernetes on Google Cloud or Anthos) without manual work.

The process involves:

CI (triggered by push event/manually to App/UI Github):

1. **Create the Application**: We make our application code.
2. **Check If It Works**: We can test it (but it's optional).
3. **Prepare for Shipping**: We package it using something called HELM Charts.
4. **Containerize It**: We turn it into a Docker container.
5. **Keep It Secure**: We make sure it's safe by checking for security issues (using Sonarqube and Blackduck).
6. **Store It Safely**: We save the HELM Charts and Docker container in our Jfrog Repository.
7. **Let the CD Know**: We tell the CD part of our system that it's ready to go by sending deployment information through Github (using a push event).

CD (triggered by push event/manually to CD Github):

8. **Get Ready**: The CD part gets ready by setting up the environment (namespace).
9. **Unlock Files (if needed)**: It decrypts any files that are protected.
10. **Bring In the Container**: It copies the Docker container from Jfrog to the Google Container Registry (GCR), especially for Google Kubernetes Engine (GKE) deployments.
11. **Apply the Plan**: It uses the HELM Chart to set up everything in the designated environment (namespace).

Pipeline Environments

The pipeline has four environments, each tied to a specific branch or naming convention. These environments are named snapshot, integration, verify, and prod. The branch or name information is specified in the Jenkinsfile. Snapshot and integration are typically considered lower-level environments, while prod and verify are higher-level environments. One key thing to remember is that lower environments generate artifacts, while higher environments promote them.

Lower Environment:

- **Snapshot (branch: develop)** - This environment is responsible for creating artifacts.
- **Integration (branch: qat)** - Here, we create release candidates.

Higher Environment:

- **Verify (branch: verify)** - This environment promotes artifacts from Integration (qat).
- **Prod (branch: master/main)** - In this environment, artifacts from Verify (verify) are promoted.

So, the qat branch is where artifacts are initially created. Afterward, the tagged artifacts move through the pipeline, being promoted from qat to verify, and finally to master/main (in this specific order).

Important Files and Directories for CI/CD

All the files related to the CI/CD processes lie in the build/anthos directory on the root of the repository. The structure of the build folder looks somewhat like:

- Build/Anthos
    - Deploy
        - Helm (files related to HELM)
        - Docker (Dockerfiles for build. Note, this can exist in module folder)
    - Deployment (HELM configurations or environment segregated configs)
        - Snapshot (configurations for develop branch)
            - Cluster name (name for dev cluster)
                - Values.yaml (config for the cluster)
        - Integration (configuration for qat branch)
            - Cluster name (name for qat cluster)
                - Values.yaml (config for the cluster)
        - Verify (configuration for verify branch)
            - Cluster name (name for verify cluster)
                - Values.yaml (config for the cluster)
            - DR Cluster name
                - Values.yaml
        - Prod (configuration for prod branch)
            - Cluster name (name for prod cluster)
                - Values.yaml (config for the cluster)
            - DR Cluster name
                - Values.yaml
        - Resources
            - Build-info.yaml (configurations related to CI process)
    - Jenkinsfile (configuration for branches including paths of other files and worker node specifications)

Jenkinsfile

This file contains important configuration settings for our Continuous Integration (CI) process. It includes information about different branches that correspond to various environments like snapshots, integration, verification, and production. You'll also find details about where build information is stored and where deployment folders are located in the 'deployment_configs' section. Additionally, you can specify the machine to run the build process using the 'jenkins_node_label' field.

The first line of the Jenkinsfile specifies which shared library to use and which branch of the shared library is needed for our process.

```
@Library('GTSO-CI-Jenkins-Library@master') _
def config = [
    build_config: "build/resources/build-info.yaml",
    snapshot_branch: "develop*",
    integration_branch: "qat*" ,
    verify_branch: "verify*",
    prod_branch: "master*",
    run_inside_node: false,
    deployment_configs: "build/deployment",
    jenkins_node_label: "ecxbuilder",      Avirup Roy Chowdhury, 8
```

You can also control which modules should be built and deployed in large application repositories by simply turning on or off switches in this file. This can help reduce the time it takes to build if you don't need to build everything. However, it's crucial to remember that you should only promote modules set as 'true' in the Quality Assurance Testing (QAT) environment to higher environments. Failing to do this may lead to errors when trying to use an image that was never actually built, causing 'ImagePullBack' errors on the cluster.

Build-info

This file holds all the information related to the building process in our Continuous Integration (CI) pipeline. It covers configurations and the steps necessary for creating 'build-info.' While most stage names and patterns are self-explanatory, let's highlight a few key points:

- The 'input' parameter in the stage description essentially determines which stage must run before yours starts.

- The 'enabled' flag can be used to turn a stage on or off during the build process.
- The 'version_existing_action' section, which is set globally, acts as a switch. It decides whether an error should cause the pipeline to fail or proceed silently. For example, if we need to promote an artifact to the 'verify' stage, and it wasn't built or couldn't be overwritten, this toggle determines whether the pipeline fails due to the error or continues without making much noise.
- In the 'environment section,' you'll find the 'tag' for the artifact being built. Typically, we use the Jenkins build number for 'develop' and 'qat,' but for 'verify' and 'prod,' we manually edit the tag in 'build-info' before merging to 'verify' with the tag we want to promote.

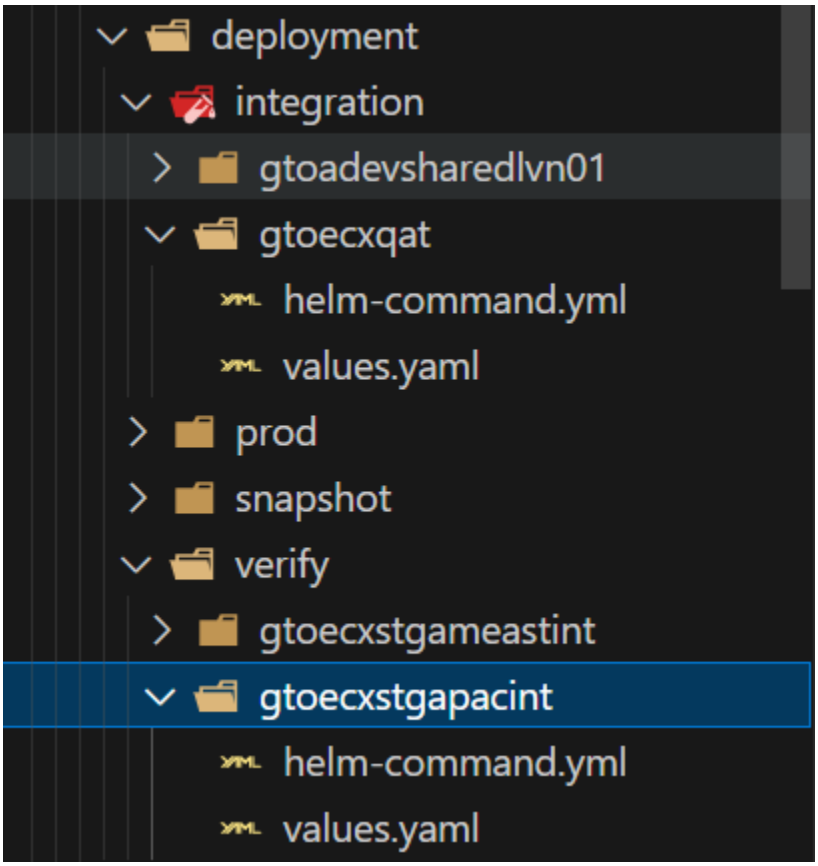
```
environmentVars:
  snapshot:
    tag: $BUILD_NUMBER$PREFIX
  integration:
    tag: $BUILD_NUMBER$PREFIX
  verify:
    tag: 7$PREFIX
  prod:
    tag: 2$PREFIX
```

Deployment folders

These folders contain configuration settings for Kubernetes workloads at the cluster level. Imagine a scenario where you want to deploy the same application in both the development and production environments. However, the development environment may require fewer resources and different configurations, like horizontal pod autoscaling or log collection, compared to the production environment. Even though it's the same application, these fine-tuning and cluster-specific settings are managed using the 'values.yaml' file found in these folders.

These folders are organized by environment and, in cases of multi-region or disaster recovery (DR) deployments, by cluster names to provide precise control over configurations.

It's important to note that all files in these corresponding deployment/cluster folders are copied as-is to the Continuous Deployment (CD) repository. This behavior allows us to add extra Jenkinsfiles or encrypted files to the folder, which are then copied to the CD repository during the build process and can be used as part of the CD process."



Deploy folders

This folder contains all the Dockerfiles and HELM charts for the applications. The HELM charts are written in an umbrella chart format, more details here.

Pipeline Enhancements

The following enhancements have been extended to the library for convenience of GTO pipeline build process:

1. Dynamic Tagging

Every artifact (like Docker or Helm) needs a unique tag to identify it, usually based on the Jenkins job's build number. For example, if your Jenkins job's build number is 27, the artifact's tag will be something like "27" or "27-proj" (with an optional suffix). In the Verify and Prod sections of the pipeline, you need to make sure to update the artifact's tag with the same build number when promoting it. This helps keep track of which version of the artifact you're working with.

2. Selective Build

In some cases, repositories for large applications might be too big to build all at once because it would take a long time. So, you can choose to build specific modules of the application by updating a flag next to the module's name in the Jenkinsfile. This change will also automatically update the values.yaml file. But remember, when promoting artifacts from qat to verify, only promote the modules that were actually built in the qat environment. If you promote modules that were not built in qat, it could lead to issues with updating tags for the wrong modules.

How to add environment wise application configs

If your application needs a specific file to be mounted into a Kubernetes pod, you can define and organize these files based on both the environment and cluster requirements. You can place these files inside the following directory structure:

<module-name>\build\deploy\helm\subcharts\be\<module-name>\env-files\<environment>\<region>\g
(\prop optional)

The <region> flag typically represents 'us-west' for the primary region deployment and 'apac' for disaster recovery (DR) region deployments. You can also include an optional 'prop' directory if needed."

How to promote to Higher Environments

To promote a file from QAT to Verify:

a. Check the last tag used in the QAT Jenkins job for your application. Make sure that the Jenkins pipeline completed successfully and ran through all the necessary stages.
b. If the names of the modules are explicitly mentioned in the Jenkinsfile, ensure that you only enable the modules that were built in the QAT environment. This helps avoid unnecessary promotions.
c. If module names aren't specified in the Jenkinsfile, navigate to the specific 'values.yaml' file located at <module-name>\build\deployment\verify<environment>\gtoecxstgameastint<cluster>\values.yaml.
   Here, you can toggle the modules in the 'tags' section as needed.
d. Update this tag value in the 'verify' section of the environment variables within the 'build-info' file. This tag signifies the version that you intend to promote to the Verify environment and subsequently to the Production environment.
e. Create a Pull Request (PR) from QAT to Verify in your version control system and trigger the Verify job to run. This ensures that the promotion process proceeds smoothly.