Deploying Applications

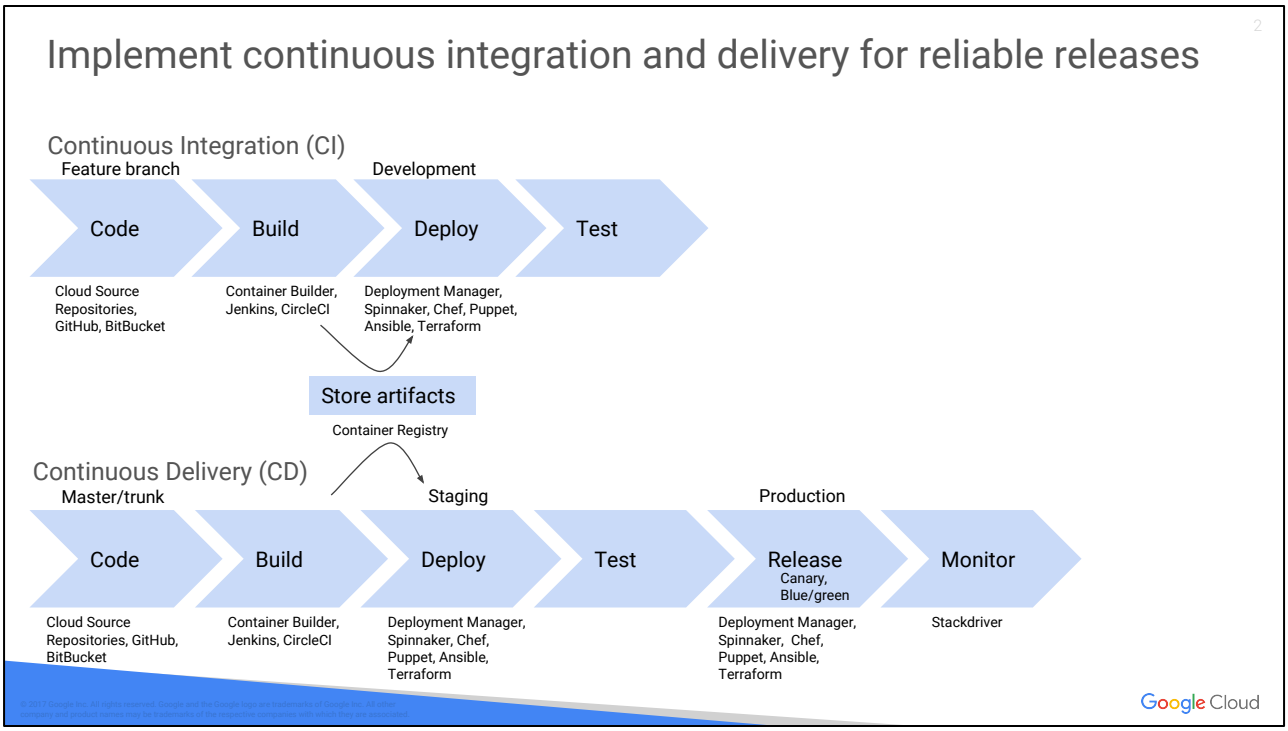
Developing Applications with Google Cloud Platform

CONTAINER BUILDER, CONTAINER REGISTRY, DEPLOYMENT MANAGER

DEPLOYING THE APPLICATION INTO CONTAINER ENGINE

Google Cloud

Version 2.0
Last modified: 2017-09-25



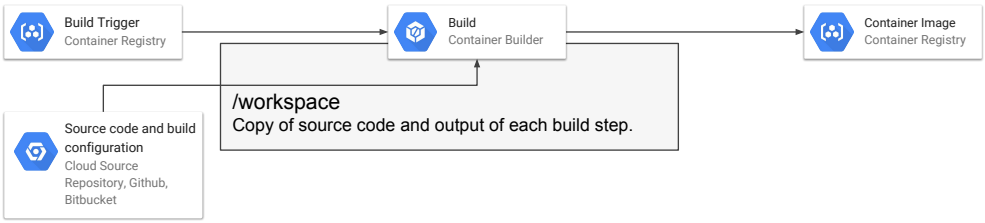
Continuous integration is a developer workflow in which developers frequently pull from the master (or trunk) and commit their changes into a feature branch in a source code repository such as Cloud Source Repositories or GitHub. This commit triggers a build in a build system such as Jenkins or CircleCI. The build process builds a new application image by using Container Builder and stores it in an artifact repository such as Container Registry. A deployment system such as Spinnaker deploys the artifacts into your cloud environment. You can use Deployment Manager to stand up the resources for the managed services that your application needs. For example, you can use Deployment Manager to create Cloud Storage buckets and Cloud Pub/Sub topics. After your application is deployed in your development environment, you can automatically run tests to verify your code. If all tests pass, you can merge your changes from the feature branch to the master.

Continuous delivery is a workflow that is triggered when changes are pushed to the master repository. The build system builds the code and creates application images. The deployment system deploys the application images to the staging environment and runs integration tests, performance tests, and more. If all tests pass, the build is tagged as a release candidate. You can manually approve a release candidate build. This approval can trigger deployment to production environments as a canary or blue/green release. You can monitor the performance of your application in the production environment by using monitoring services such as Google Stackdriver. If the new deployment functions optimally, you can switch over your entire traffic to this new release. If you discover problems, you can roll back to the last stable release.

The continuous deployment workflow varies slightly in that there is no manual approval process. The deployment system automatically deploys release candidates to the production environment.

In the rest of this presentation, you will consider two keys aspects of your cloud-native application: the container image for your application and the Google Cloud Platform resources required by your application.

Use Container Builder and Container Registry to create application images




```
cloudbuild.yaml
steps:
- name: 'gcr.io/cloud-builders/docker'
  args: [ 'build', '-t', 'gcr.io/$PROJECT_ID/cb-demo-img', '.' ]
images:
- 'gcr.io/$PROJECT_ID/cb-demo-img'
tags:
- "test"
- "2.0b35"
```


Google Cloud

The container image for your application is a complete package that contains the application binary and all the software that is required for the application to run. When you deploy the same container image on your development, test, and production environments, you can be assured that your application will perform exactly the same way in each of these environments. Google Cloud Container Builder is a fully managed service that enables you to set up build pipelines to create a Docker container image for your application and deploy the image to Google Cloud Container Registry. You do not need to download all build tools and container images to a build machine or manage your build infrastructure.

By using Container Registry and Container Builder, you can create build pipelines that are automatically triggered when you commit code to a repository. In Container Registry, you can create a build trigger that is executed based on a trigger type. A trigger type specifies whether builds should be triggered based on commits to a particular branch in a repository or commits that contain a particular tag.

You must also create a build configuration file that specifies the steps in the build pipeline. Steps are analogous to commands or scripts that you execute to build your application. Each build step is a Docker container that is invoked by Container Builder when the build is executed. The step *name* identifies the container to invoke for that build step. The *images* attribute contains the name of the container image to be created by this build configuration. Container Builder enables you to specify different types of source repositories, tag container images to enable searches, and create build steps that perform operations such as downloading and processing data without

creating a container image. The build configuration can be specified in YAML or JSON format.

Container Builder mounts your source code into the /workspace directory of the Docker container associated with a build step. The artifacts produced by each build step are persisted in the /workspace folder and can be used by the following build step. Container Builder automatically pushes the built container image to Container Registry. In Container Registry, you can view the status and history of builds. Container Builder also publishes build status notifications to Google Cloud Pub/Sub. You can subscribe to these notifications to take action based on build status or other attributes.

For more information, see:

- Container Builder Documentation: https://cloud.google.com/container-builder/docs/
- Container Builder API documentation: https://cloud.google.com/container-builder/docs/api/reference/rest/
- Container Registry: https://cloud.google.com/container-registry/docs/

Use Deployment Manager to launch your Google Cloud Platform resources

```
Deployment Configuration (YAML file)
imports:
- path: path/to/template.jinja
  name: my-template
- path: path/to/another/template.py

resources:
  - name: resource-a
    type: type-of-resource
    properties:
      property-a: value
      property-b: value
      ...
```

- ✓ Define template properties and environment variables.
- ✓ Define conditions to create different deployments based on a single configuration.
- ✓ Specify dependencies between resources.
- ✓ Specify startup scripts to run when VM launches.
- ✓ Define outputs.

Google Cloud

Deployment Manager enables you to launch VM instances based on container images that are created by your build pipeline. You can also use Deployment Manager to launch other Google Cloud Platform resources that are required for your application. The Deployment Manager API is integrated into Google Cloud Platform.

A deployment configuration defines the cloud resources to provision. The configuration includes the type and properties of the resources that are part of the deployment.
A deployment configuration consists of a top-level configuration file in YAML syntax, templates, and additional files. You can declare all your resources in the top-level configuration file for simple deployments. For more complex configurations or to create reusable chunks of configuration, you can create templates that create subsets of the configuration. You can then import one or more templates to construct your complete configuration. Deployment Manager will expand these templates to create your final configuration.

These reusable templates can be developed using Jinja or Python syntax. Jinja templates and Python code are used to generate the YAML configuration.

Deployment Manager enables you to do the following:

- You can reuse templates and configure resources differently for different environments. For example, you can create test and production environments by using the same deployment configuration with different template properties,

- environment variables, and conditions.
- For complex deployments, you can specify dependencies between resources. For example, the creation of a VM might depend on the creation of persistent disks.
- You can specify startup scripts that are run when the VM launches. These startup scripts might download software or run other commands to configure the VM.
- You can declare outputs for the template. For example, you can expose the IP address of a database created in a template so that other scripts can connect to the database.

For more information about Deployment Manager, see:

- Deployment Manager documentation: https://cloud.google.com/deployment-manager/docs/
- Deployment Manager examples on GitHub: https://github.com/GoogleCloudPlatform/deploymentmanager-samples