Welcome to Using CI/CD to Deploy Kubernetes Workloads.

The concept of a development pipeline for testing and deploying new iterations of software is not a new one.

As you begin to move your applications to Google Cloud, it is important to keep the same efficient workflow, but take advantage of the superpower-like capabilities of cloud computing. That's the theme of this module.

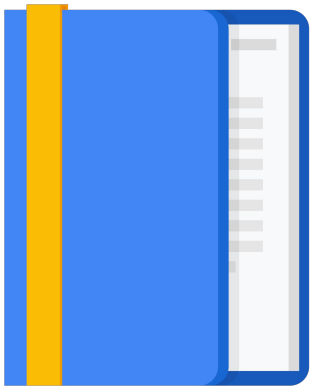
We are going to introduce the concept of CI/CD, highlight some of the tools that are available to you in Google Cloud, and look at several use cases to find the one that fits your organization best.

Learn how to ...

Manage application code in a source repository that can trigger code changes to a continuous delivery pipeline.

Create a continuous delivery pipeline using Cloud Build, and start it manually or automatically with a code change.

Implement a canary deployment that hosts two versions of your application in production for release testing.



Google Cloud

In this module, you'll learn how to:

- Manage application code in a source repository that can trigger code changes to a continuous delivery pipeline.
- Create a continuous delivery pipeline using Cloud Build and start it manually or automatically with a code change.
- Implement a canary deployment that hosts two versions of your application in production for release testing.

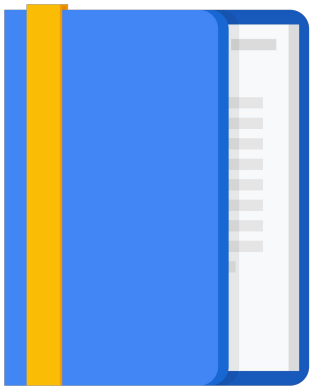
Agenda

CI/CD background

CI/CD in Google Cloud

Example CI/CD architectures

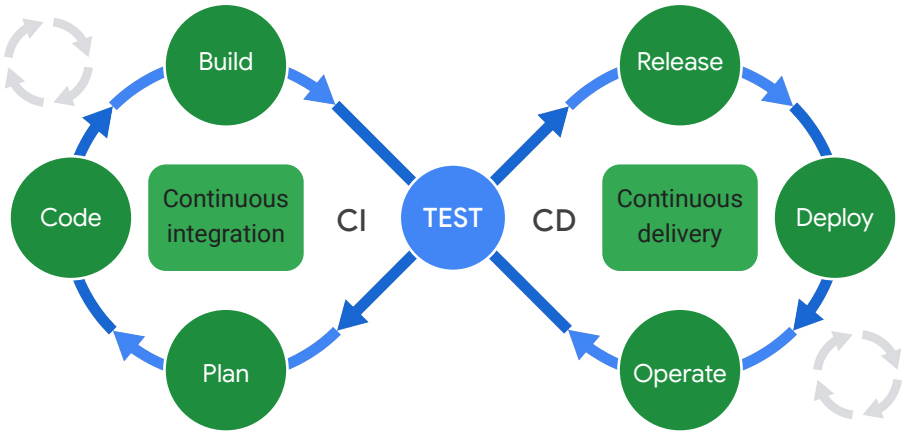
Summary



Google Cloud

Let’s begin by looking at what CI/CD is, and why it is a useful software development technique.

Introduction to CI/CD (1/3)

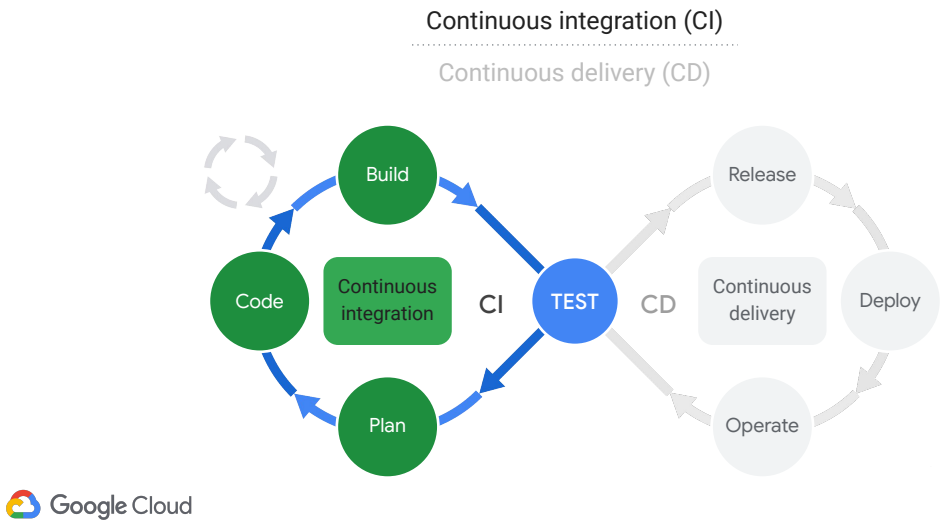


Google Cloud

Continuous integration/Continuous delivery, or CI/CD, gives your software development process the agility to meet your customers' needs quickly.

CI/CD also helps reduce the risk of delivering changes by allowing for more frequent and incremental updates to applications in production.

Introduction to CI/CD (2/3)

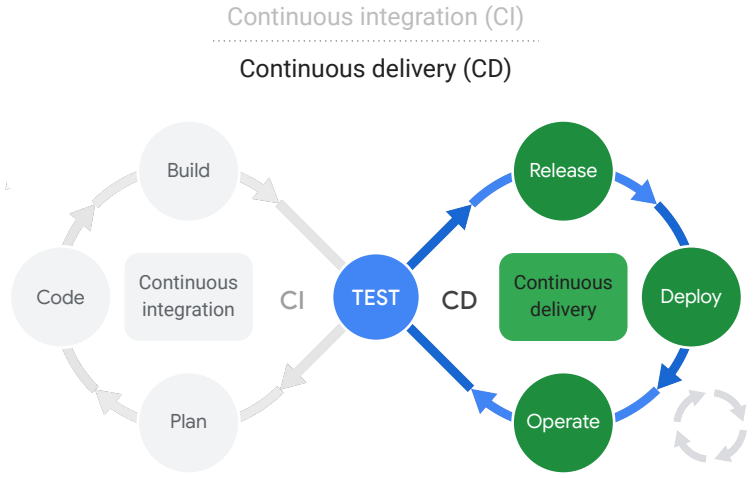


Google Cloud

Continuous integration is the practice of regularly merging the work products of individual developers together into a repository.

The primary purpose is to enable early detection of integration bugs, which results in tighter cohesion and more development collaboration.

Introduction to CI/CD (3/3)

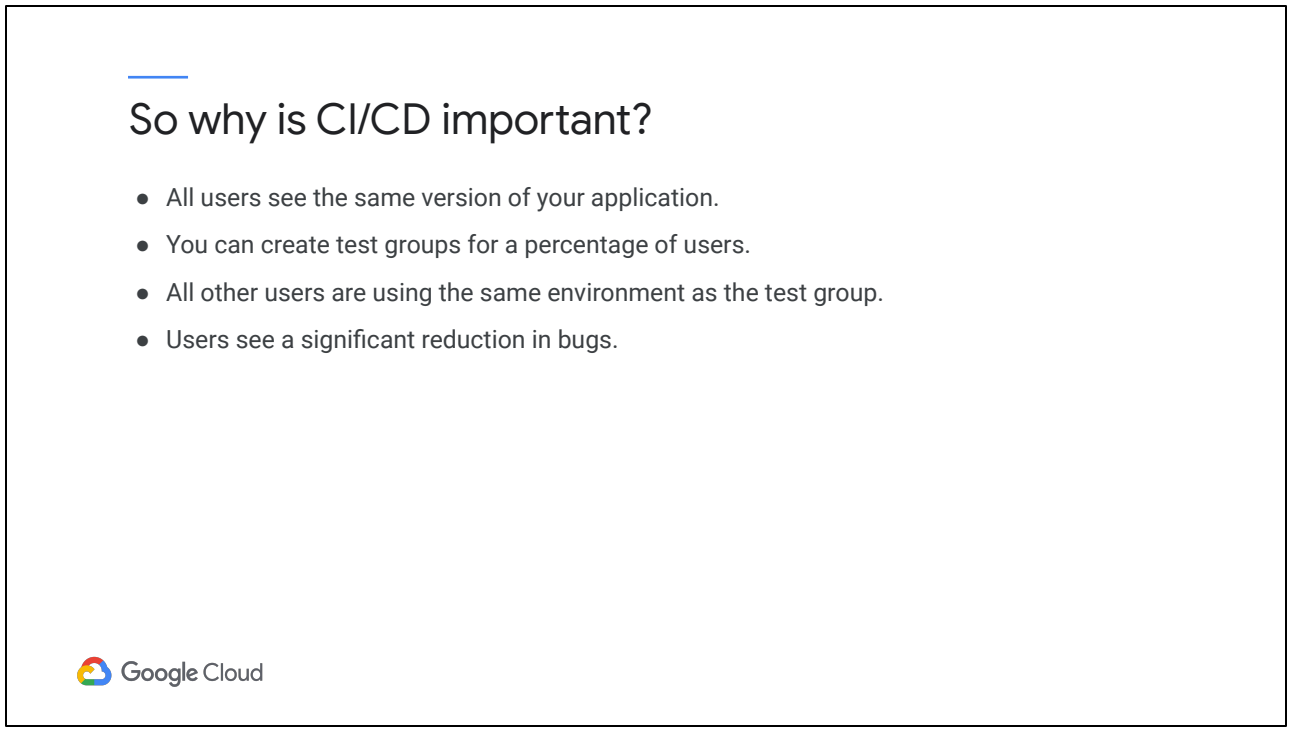


Google Cloud

With continuous delivery, teams produce software in short cycles and use processes that ensure the reliable release of software to production at any time.

The overall goal is to be able to release any change to production at any time, while minimizing the risk of loss of service quality.

Many companies employ a CI/CD process that enables them to reliably release to production several times a day.



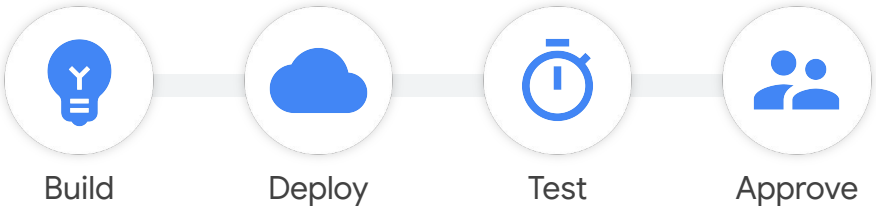
If you ask a developer why CI/CD is important, they will tell you that CI/CD automates the rollout of updates and that it can ensure that your users all see the same version of your app. This in turn can make resolving bugs and issues easier, because it reduces the amount of variance that can exist in the users' environments and makes better use of the developers' time and effort.

But as a business choice, CI/CD can be far more effective than just having your developers work efficiently. If all your users have the same version of the application, the developers can test updates on a small percentage of the user base, for example, 1%.

You can use the feedback from that 1% to identify and fix any bugs or anomalies in the new update, so that when you roll the update out to the other 99% of your users, you can be confident that very few bugs will be visible to the users. This is because all of your users are using the exact same version of the application as the test group.

This can mean that your users see a significant reduction in bugs in the application updates that are rolled out. Your app still has the same number of bugs, but from the user's perspective, your software has very few, if any bugs.

CI/CD pipelines



Google Cloud

To implement CI/CD correctly, you need to construct a pipeline. A **pipeline** is a process that takes a version of an application's codebase and performs all steps necessary to release it to production. Pipelines can be triggered manually, or they can be triggered automatically when changes are pushed to the codebase. They are generally composed of separate stages, most commonly these four.

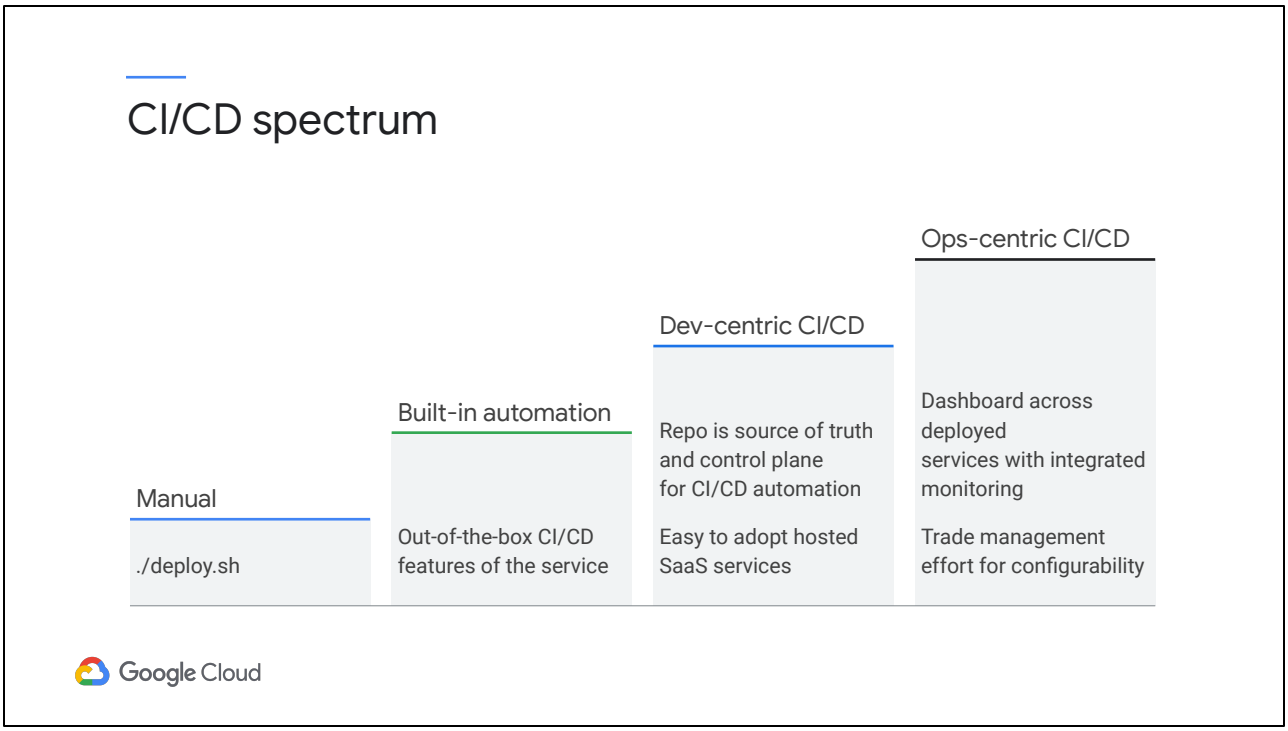
- The first stage is **Build.** In this stage the codebase is checked out to a specific version, and artifacts, for example Docker container images, are built.
- After the build stage, the next stage is **Deploy.** In this stage, the artifacts are deployed into a test environment that replicates the production environment.
- After the application has been deployed, it is moved to the **Test** stage. Here the application is tested in multiple ways to ensure application quality. The most common type of tests are unit, functional, integration, vulnerability, and performance.
- Finally, if the application passed all of the tests, it can move to the **Approve** stage. In this stage, developers can manually determine whether a pipeline should proceed to the production environment.

CI/CD spectrum

Google Cloud

When we talk about CI/CD, there is not just one “version.” There is a spectrum of implementations, ranging from manual to almost completely automated.

Each point on the range has benefits and disadvantages. Let's look at some of the options.



At the lowest of levels is complete manual, in other words, no CI/CD! In this situation, each developer is responsible for integration and deployment of all code that they write, while ensuring that it does not conflict with any other developer's code. As you can imagine, this is not the end of the spectrum that we are aiming for!

The next level up is a packaged tool with a fixed set of Built-in automation. This is completely reliant on the CI/CD tool, and these tools generally allow very specific customization that is frequently difficult to maintain or review. It is not suitable for any reasonably complex system.

Dev-centric CI/CD is centered around the systems code repository. A great example of this style of CI/CD is using GitOps. GitOps is a methodology that uses your source repository as the pillar of truth for your application. You store both your code and your configuration files in the repository. With a Kubernetes environment, you will store manifest and yaml files to define the deployments for your app. Cloud Build is a Google tool that allows you to create such a pipeline in Google Cloud. But we will return to this in a little while.

Finally there is Ops-centric CI/CD, which is focused on ensuring high levels of management and monitoring. This control comes at a cost, and is less friendly for developers, but offers very high levels of configurability. A common tool to implement OPs-centric CI/CD is Spinnaker. Spinnaker is designed to handle very large deployments with possibly thousands of instances. In this scenario, the cost of managing the tool is justified based on the scale of the deployment.

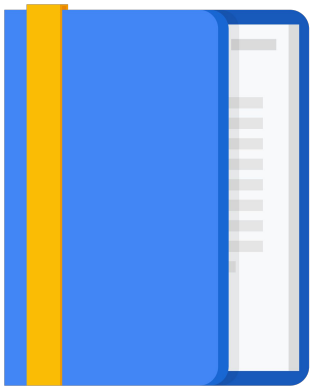
Agenda

CI/CD background

CI/CD in Google Cloud

Example CI/CD architectures

Summary

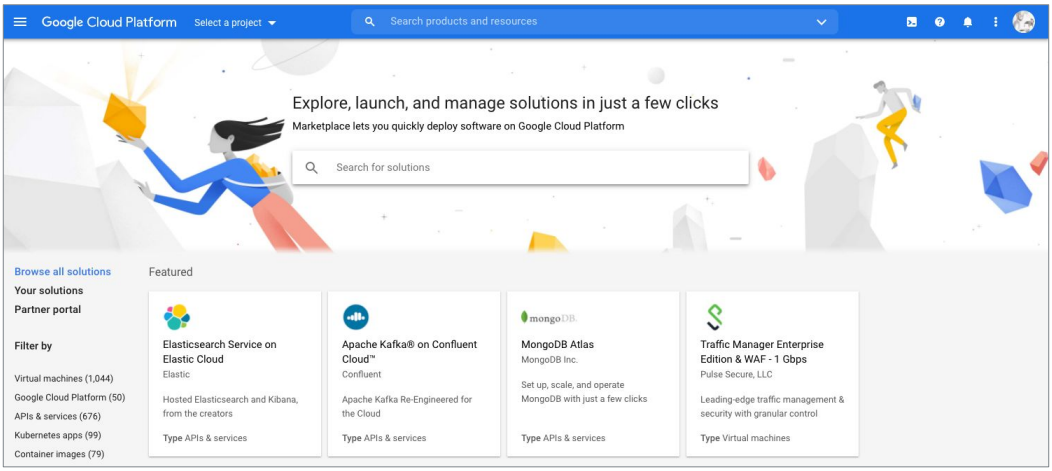


Google Cloud

As you have seen already, creating a CI/CD pipeline is a decision with many options, some overlapping, and can become more complicated depending on pipeline customization, number of environments, and scale of deployments.

Thankfully, Google Cloud has a selection of tools to help with this. Let's look at some of them now.

Example marketplace CI/CD tools (1/2)



Google Cloud

Google Cloud Marketplace has a wide range of tools to assist you in the creation of your infrastructures, applications, and, in this case, your CI/CD pipelines.

Let's look at some of the most commonly used CI/CD tools available in the marketplace. If you already use CI/CD with your applications, these may be familiar to you.

Example marketplace CI/CD tools (2/2)

Jenkins

Spinnaker

CircleCI

GitLab CI

Drone

Cloud Build

Google Cloud

Jenkins is one of the earliest open-source continuous integration servers and remains the most common option in use today.

Spinnaker is an open-source, multi-cloud, continuous delivery platform that helps you release software changes with high velocity and confidence.

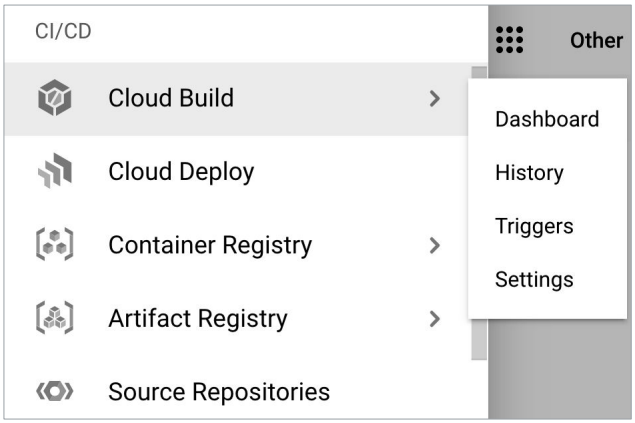
The CircleCI continuous integration and delivery platform makes it easy for teams of all sizes to rapidly build and release quality software at scale.

GitLab CI is a continuous integration tool built into GitLab, a platform for Git repository hosting and development tools.

Drone is a modern CI/CD platform built with a container-first architecture. Running builds with Docker is at the core of Drone's design.

You may be familiar with the tools that we have mentioned so far. Cloud Build is Google's own Continuous Integration tool that allows you to create a build and test and deploy workflows within your Google Cloud environment. Cloud Build also works with local builds and in on-premises environments. We briefly looked at Cloud Build earlier in this specialization. Let's take a moment to look at Cloud Build in a bit more detail.

Cloud Build

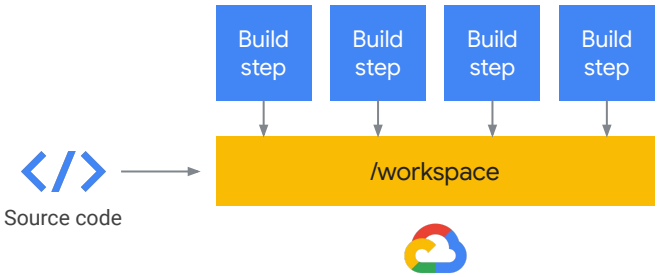


Google Cloud

Within the Google Cloud Console, we have Cloud Build. Cloud Build is a service that can execute your builds on Google Cloud's infrastructure, an on-premises environment, or a mixture of the two.

Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to your specifications, and produce artifacts such as container images or Java archives. You can configure Cloud Build to react to changes in either source code or release tags, based on pre-defined triggers.

Cloud Build workflow

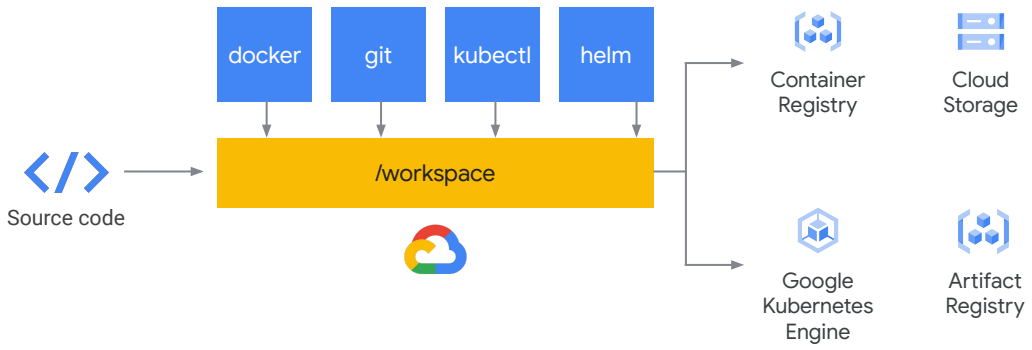


Google Cloud

Within Cloud Build, a common volume “/workspace” is mounted. Then the application’s source code is added to this volume.

Each of the build steps is independently created as a container. Because the workspace is common, each container can operate on it. This environment has reduced the need for redundant data stores.

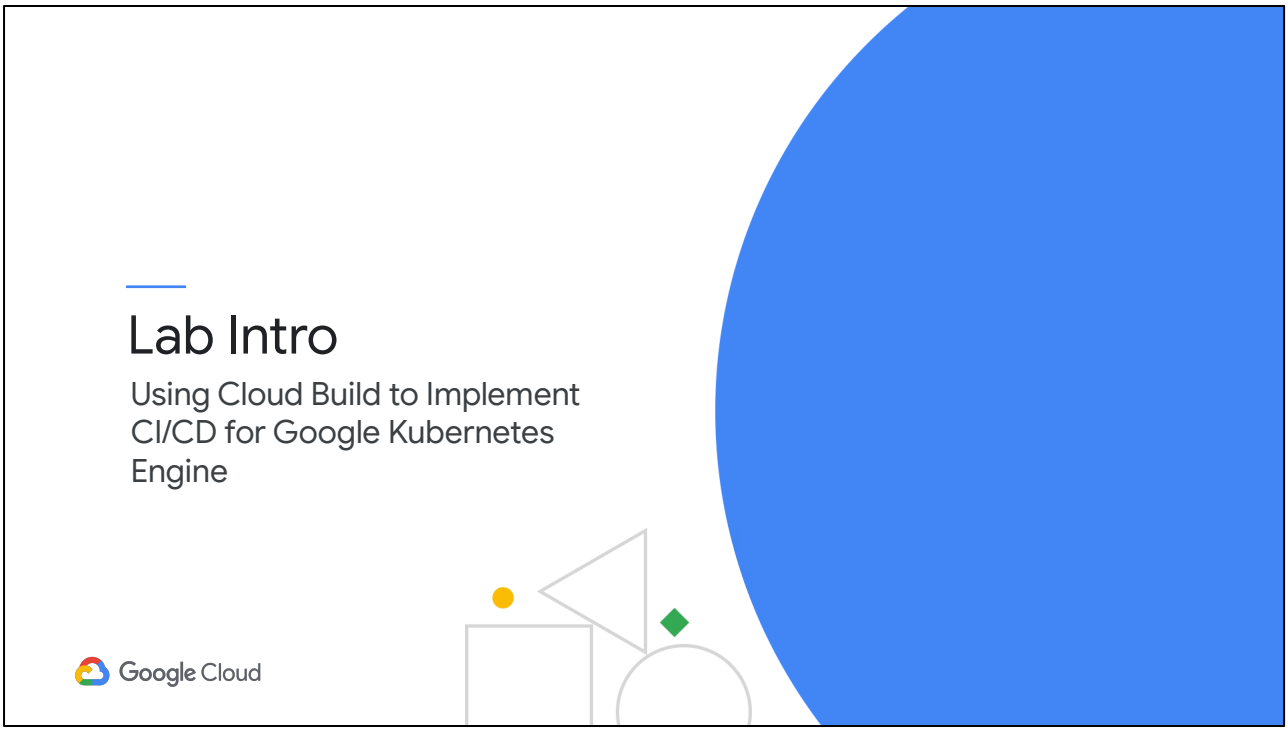
Cloud Build workflow: Example



Google Cloud

In Cloud Build, tools such as docker, git, kubectl, and helm can operate as containers in a step on the workspace volume.

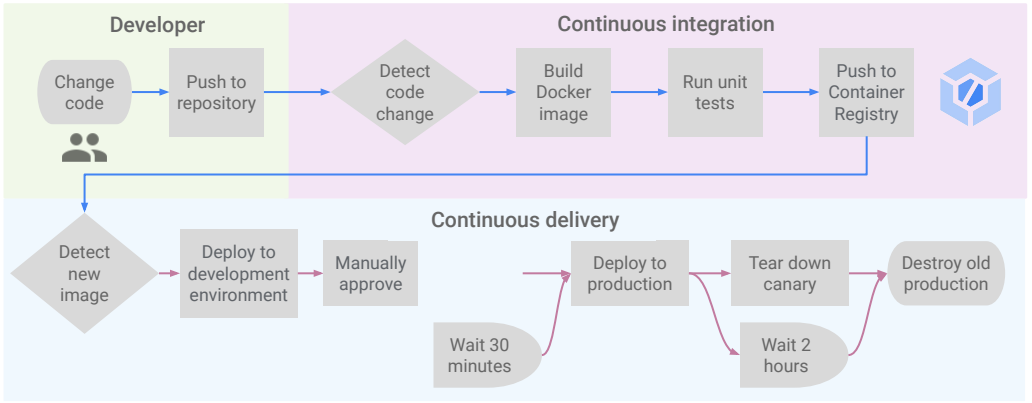
Cloud Build also allows for interaction with other Google Cloud products, such as Container Registry, Artifact Registry, Cloud Storage, and GKE.



Now that you have seen how CI/CD can be implemented in Google Cloud, let's complete a lab to see this process working. In this lab, you'll create a continuous integration and delivery pipeline on Google Cloud. You will use the GitOps methodology.

https://cloud.google.com/kubernetes-engine/docs/tutorials/gitops-cloud-build

Pipeline process flow: Remove old environment

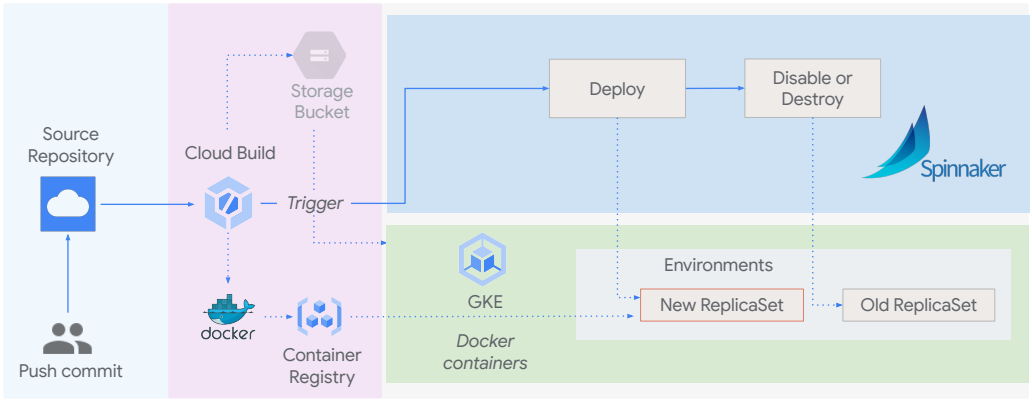


Google Cloud

Now that you have seen how Cloud Build can be used to establish a Continuous integration workflow with your application, let's look at a more complicated example. First, let's look at the pipeline, and see what is happening at each step.

- The first point of the pipeline is that the developer alters the source code of the application and pushes the update to the repository.
- This will trigger the CI tool, which detects a change in the code.
- The trigger builds a new docker container image based on the updated code.
- After the container image is built, the CI tool performs its Unit Tests; if these pass successfully the docker image is pushed to the Container Registry.
- At this stage, the Continuous Delivery tool detects the new image and pushes it to the development environment.
- After deployment, it awaits a manual check, before it is moved to the new production environment, and traffic is slowly moved to this environment.
- After a time limit is reached, the old production environment is destroyed.

Continuous deployment to containers



Google Cloud

We can extend the last example so that after the application is deployed to production, it is deployed to a Kubernetes cluster.

In this example we are using Spinnaker to deploy new containers using the new container image built by Cloud Build into GKE. After the new image is deployed, Spinnaker will destroy the old one.

The key point here is that the underlying infrastructure remains.

After a developer has pushed a commit, Cloud Build will trigger a Spinnaker pipeline that will create and deploy a new ReplicaSet and destroy the existing one.

In this architecture the Kubernetes cluster nodes remain, and only pods are created and destroyed.

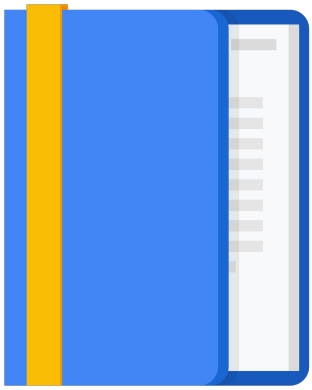
Agenda

CI/CD background

Examples of CI/CD tools

Example CI/CD architectures

Summary



Google Cloud

Now that you understand what CI/CD is and what common tools are used in CI/CD pipelines, let's look at some examples of CI/CD in different environments.

In each of these examples, we will look at a Kubernetes environment using CI/CD.

Example CI/CD architectures

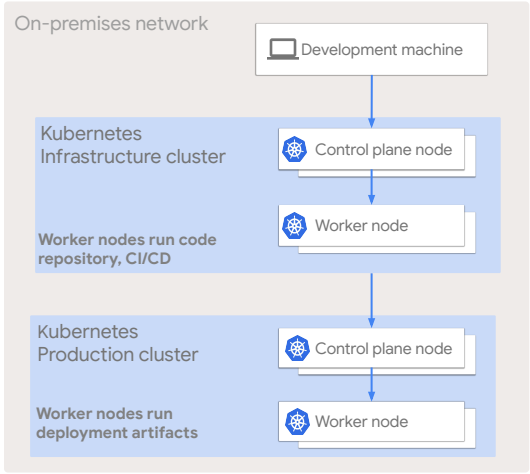
On-premises CI/CD

Cloud CI/CD

Google Cloud

The first example we'll look at is having the entire pipeline on-premises.

On-premises CI/CD with Kubernetes



Advantages

- Automatic scaling
- No single point of failure

Disadvantages

- Additional resources required
- Geographically limited to data center

Google Cloud

The Developer updates the source code and submits it to the Git repository.

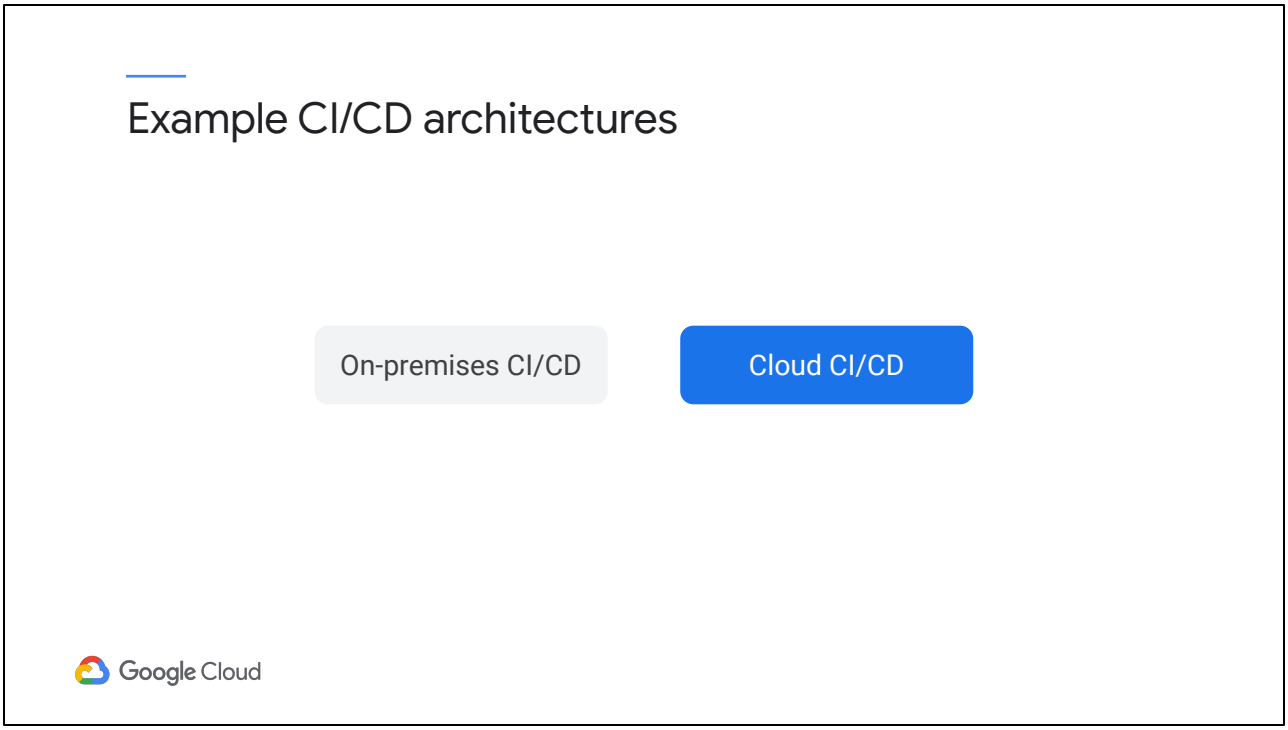
This Git push connects to the control plane node in the infrastructure cluster to delegate work to worker nodes. The worker nodes will build and test the updated source code.

After the tests complete successfully, the worker nodes delegate work to the control plane node in the production cluster.

The production cluster control plane node delegates the task of deploying the updated application to the worker nodes.

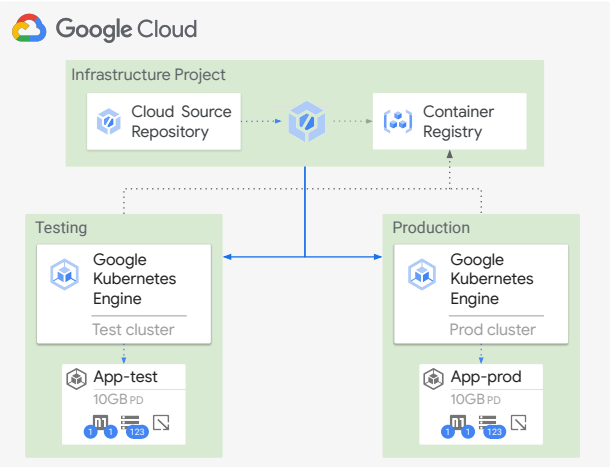
This architecture does have advantages because it is based in Kubernetes. It will scale automatically and it does not have a single point of failure.

However, the disadvantages to this setup are that the application is geographically limited to a data center and that it requires additional resources.



Now let's look at an example that is based completely in Google Cloud.

Google Cloud serverless CI/CD



Advantages

- Fast deployments
- No single point of failure
- Serverless (via Cloud Run)

Disadvantages

- Best suited for simpler deployments

Google Cloud

The developer updates the source code and commits it to the repository. This action triggers Cloud Build to create a updated container and pushes it to Container registry.

The updated application is now pulled from the Container registry and deployed to the testing environment. A load balancer connects to the control plane node to the access THE git service running on pod 1.

The Repository notifies the CI/CD pipeline via a trigger and the CI/CD workers pass the updated application to the testing cluster. Tests are now performed on the updated application.

The updated application is deployed to the production cluster via another load balancer set.

This architecture has some strong advantages:

- The deployments are fast.
- The architecture does not have a single point of failure.
- It has a serverless architecture; this means you don't even have to manage the clusters! Cloud Run takes care of this for you.

The key disadvantage here is that while simpler deployments can be enabled by just using Cloud Build triggers, creating a more complex logic or multi-stage pipelines

require customized Cloud Build pipeline code.

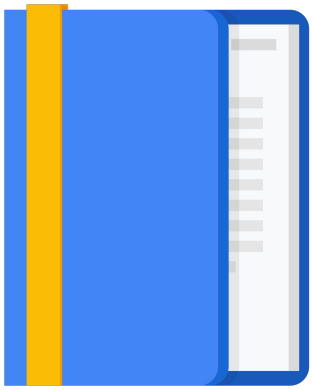
Agenda

CI/CD background

Examples of CI/CD tools

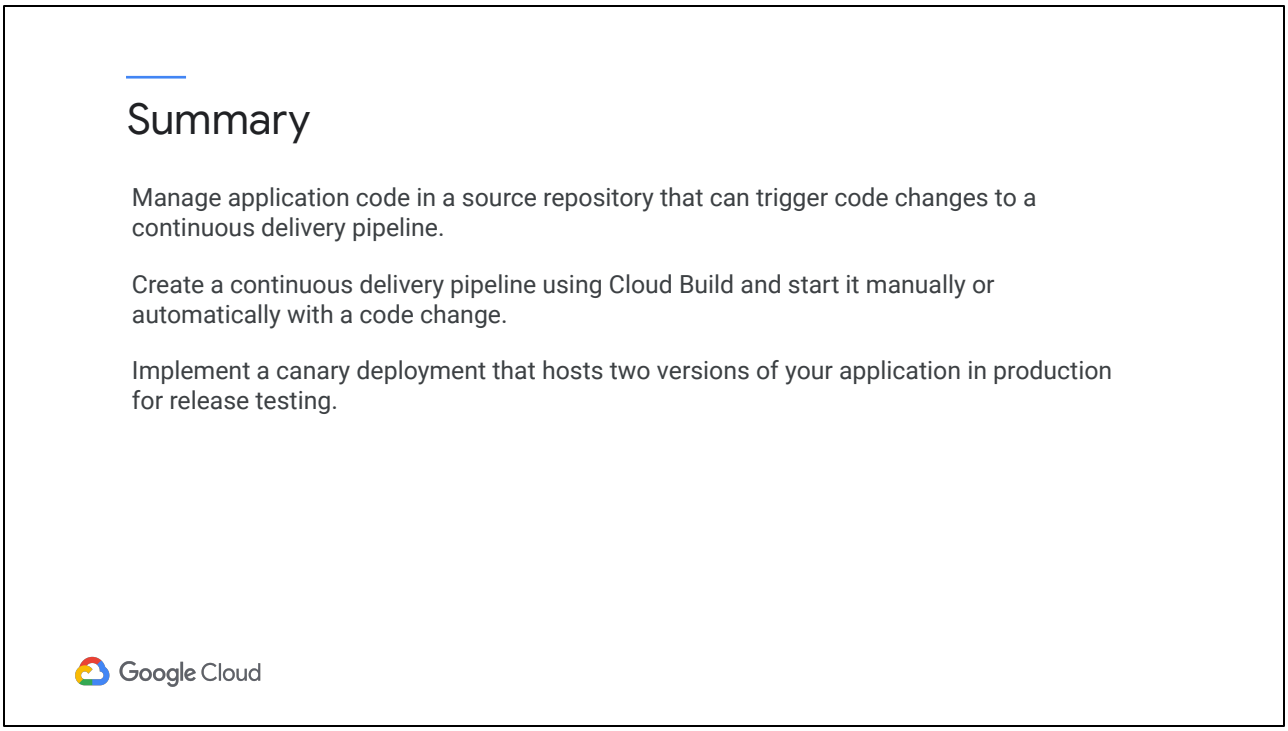
Example CI/CD architectures

Summary



Google Cloud

That concludes the Using CI/CD to Deploy Kubernetes Workloads module.



In this module, you learned how to:

- Manage application code in a source repository that can trigger code changes to a continuous delivery pipeline,
- Create a continuous delivery pipeline using Cloud Build and start it manually or automatically with a code change, and
- Implement a canary deployment that hosts two versions of your application in production for release testing.


Google Cloud