

## Using CI/CD with Google Kubernetes Engine

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

With Google Cloud, using continuous integration and continuous deployment pipelines, commonly referred to as simply CI/CD, helps create efficient workflows for maintaining and updating applications.

# Using CI/CD with Google Kubernetes Engine

- 01 Continuous integration and continuous deployment (CI/CD)
- 02 Constructing a CI/CD pipeline
- 03 CI/CD products
- 04 Best practices for using CI/CD on Google Cloud



Google Cloud

In this final section of this course titled, “Using CI/CD with Google Kubernetes Engine,” you’ll:

- Explore CI/CD.
- Examine how to construct a CI/CD pipeline.
- Identify CI/CD products offered by Google Cloud and third parties.
- Learn about best practices for using CI/CD on Google Cloud.
- Create a continuous delivery pipeline by using Cloud Deploy and Skaffold.

# Using CI/CD with Google Kubernetes Engine

- 01 Continuous integration and continuous deployment (CI/CD)
- 02 Constructing a CI/CD pipeline
- 03 CI/CD products
- 04 Best practices for using CI/CD on Google Cloud

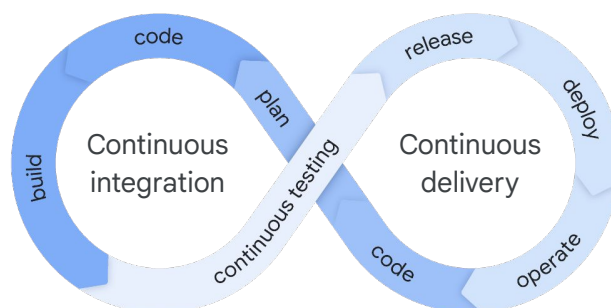


So, what is CI/CD, and why is it a useful software development technique?

# CI/CD:

## Continuous integration and continuous delivery

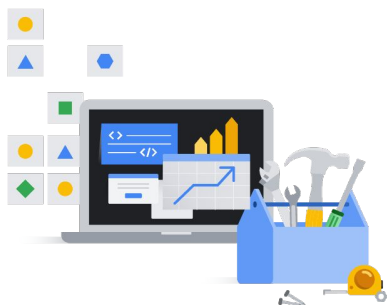
**Software development practice** that automates the process of building, testing, and deploying code changes.



Continuous integration and continuous delivery, which is what CI/CD stands for, is a software development practice that involves automating the process of building, testing, and deploying code changes. This allows developers to deliver updates to users more frequently and reliably.

Let's define each term.

# Continuous integration



The practice of frequently merging code changes into a central repository.

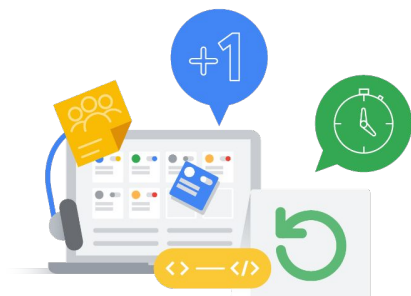
Repository ensures code remains functional through automated:

✓ Builds

✓ Tests

**Continuous integration** refers to the practice of frequently merging code changes into a central repository. This repository then undergoes automated *builds and tests* to ensure the code remains functional after each integration.

# Continuous delivery



Producing software in short cycles and using processes to ensure a reliable release to production.



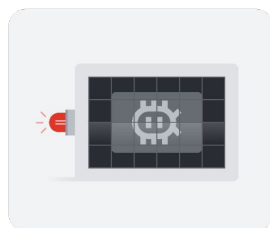
Helps minimize the risk of loss of service quality.



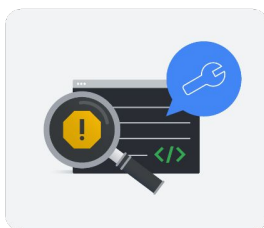
Might mean pushing releases to production several times a day.

And then there is **continuous delivery**, which is a method where teams produce software in short cycles and use processes that ensure the reliable release of software to production at any time. This helps minimize the risk of loss of service quality. For some companies, this means pushing releases to production several times a day.

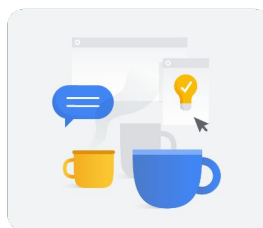
## The benefits of CI/CD



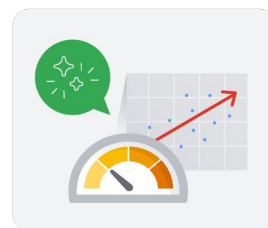
Helps detect bugs sooner.



Helps improve code quality.



Helps enhance collaboration.



Helps reduce the risk of regressions.

Now that you know what CI/CD stands for, let's explore some of the benefits.

For starters, CI/CD can **help detect bugs sooner**, which can prevent them from turning into larger problems later in the development cycle. For example, if all of your users have the same version of the application, developers can test updates on a percentage as small as 1%. Feedback can be used 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 users, very few bugs will be visible to them.

CI/CD can also **help improve code quality**, because frequent integration and testing can encourage developers to write cleaner and more robust code.

It can **help enhance collaboration** since teams work on a shared codebase.

And finally, CI/CD can also **help reduce the risk of regressions**, because merging code frequently minimizes the chances of breaking existing functionalities.

# Using CI/CD with Google Kubernetes Engine

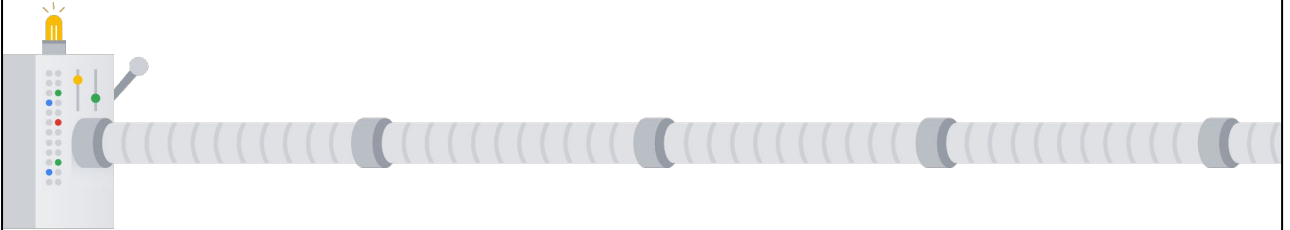
- 01 Continuous integration and continuous deployment (CI/CD)
- 02 Constructing a CI/CD pipeline**
- 03 CI/CD products
- 04 Best practices for using CI/CD on Google Cloud



Now that you've been introduced to what CI/CD is, let's explore how to construct and implement a CI/CD pipeline.



# CI/CD pipelines



Pipelines can be triggered manually or automatically.

A pipeline represents the stages involved in getting an application's codebase ready to be released to production. Pipelines can be triggered manually or automatically.

Let's explore each stage of the pipeline.

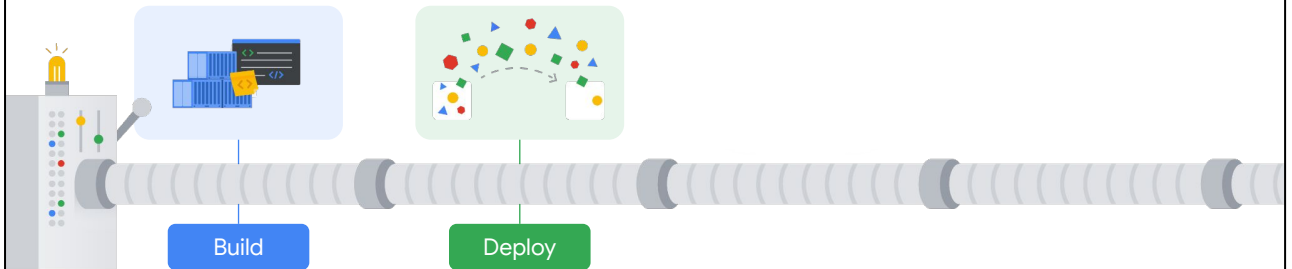
## CI/CD pipeline stages



Codebase is checked out to a specific version and artifacts are built.

The first stage is **build**. During the build stage, the codebase is checked out to a specific version, and artifacts, for example Docker container images, are built.

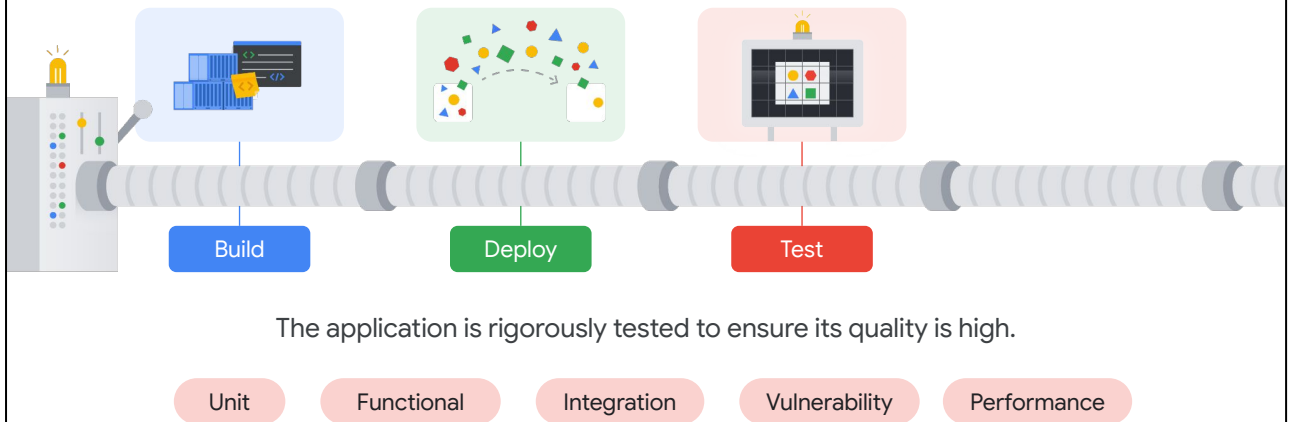
## CI/CD pipeline stages



Artifacts are deployed into a test environment that replicates the production environment.

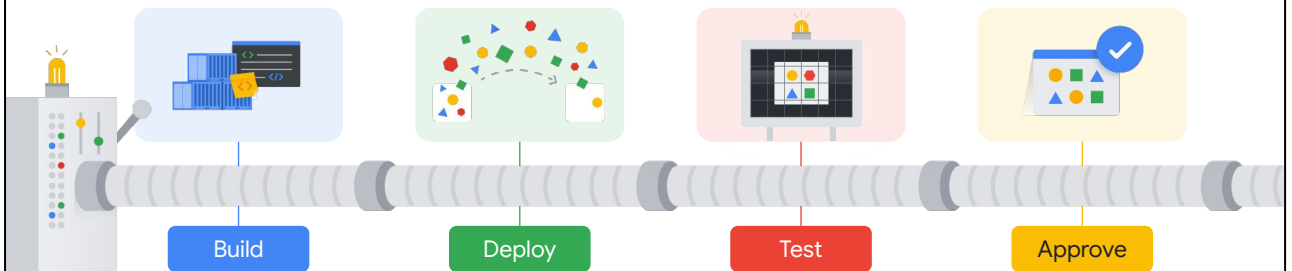
The **deploy** stage is next. This is when artifacts are deployed into a test environment that replicates the production environment.

## CI/CD pipeline stages



After the application has been deployed, it's moved to the **test** stage. During this stage the application is rigorously tested to ensure its quality is high. The most common types of tests are unit, functional, integration, vulnerability, and performance.

## CI/CD pipeline stages



Developers will decide if a pipeline should proceed to the production environment.

And finally, if the application passes every test, it moves on to the **approve** stage. During this stage, developers will decide if a pipeline should proceed to the production environment.

## There is no single version of a CI/CD pipeline



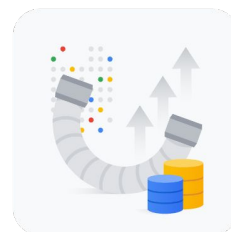
Manual pipelines  
(no CI/CD)



Packaged tools  
with a fixed set of  
built-in automation



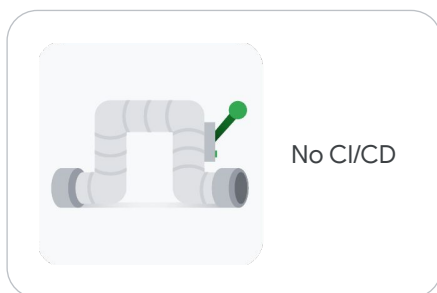
Dev-centric  
CI/CD



Ops-centric  
CI/CD

Now it's important to understand that there is no single version of a CI/CD pipeline, yet rather a variety that represents different types of implementations, which range from manual to almost completely automated.

## Manual pipelines



Each developer is responsible for:

- Integrating code they write.
- Deploying code they write.
- Ensuring new code doesn't conflict with other code.

With **manual pipelines**—in other words, one that has no CI/CD—each developer is responsible for integrating and deploying all of the code that they write. They must also ensure that the new code does not conflict with any other developer's code.

## Packaged tools with a fixed set of built-in automation



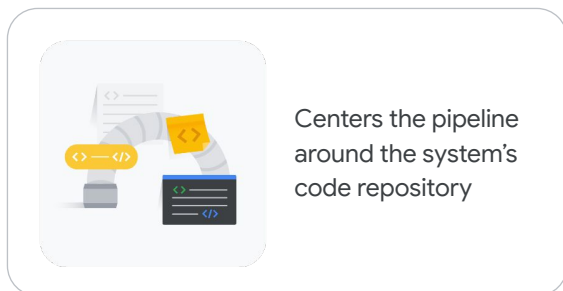
A pipeline reliant on a CI/CD tool

- Allows for very specific customization.
- Can make development difficult to maintain and review.
- Not suitable for complex systems.

Another less manual version of a pipeline is a **packaged tool with a fixed set of built-in automation**. This type of pipeline is completely reliant on a CI/CD tool. While these tools generally allow for very specific customization, it can make development difficult to frequently maintain and review. It's not suitable for complex systems.



## Dev-centric CI/CD



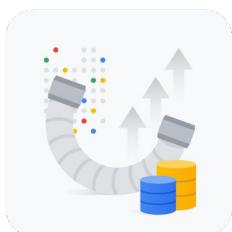
- GitOps → Stores both the code and configuration files in a source repository, which is used as the authority for an application.
- Use Cloud Build to create dev-centric pipelines.

Then there is **dev-centric**, or developer-centric, **CI/CD**, which centers the pipeline around the system's code repository. An example of this would involve using GitOps, which is a methodology that stores both the code and configuration files in a source repository and then uses that source repository as the authority for an application.

Within a Kubernetes environment, manifest and YAML files are stored and used to define the deployments for an application.

To create these types of pipelines in Google Cloud, you can use Cloud Build.

## Ops-centric CI/CD



Prioritizes automating operational tasks to ensure smooth and reliable software delivery.

Common tool: Spinnaker

- Designed to handle very large deployments (thousands).
- Cost can be justified by the scale of the deployment.

The final pipeline example is **ops-centric**, or operations-centric, **CI/CD**, which prioritizes automating operational tasks like infrastructure management, monitoring, and incident response to ensure smooth and reliable software delivery. By streamlining operations, developers can free up time to focus on code.

Spinnaker is a common tool to implement ops-centric CI/CD, which is designed to handle very large deployments, with possibly thousands of instances. The cost of managing the Spinnaker can be justified based on the scale of the deployment.

# Using CI/CD with Google Kubernetes Engine

- 01 Continuous integration and continuous deployment (CI/CD)
- 02 Constructing a CI/CD pipeline
- 03 CI/CD products**
- 04 Best practices for using CI/CD on Google Cloud



## CI/CD tools in the Google Cloud Marketplace

Jenkins	One of the oldest open-source continuous integration servers.
Spinnaker	Open-source, multi-cloud, continuous delivery platform for software releases.
CircleCI	A continuous integration and delivery platform for teams of all sizes.
GitLab CI	A continuous integration tool built into GitLab.
Drone	A CI/CD platform built with a container-first architecture. Runs builds with Docker.

The Google Cloud Marketplace has a wide range of tools to help users design and create complex infrastructures, applications, and CI/CD pipelines.

Let's explore some of them.

- **Jenkins** is one of the oldest open-source continuous integration servers, and remains the most popular option in use today.
- **Spinnaker** is also an open-source tool. It's a multi-cloud, continuous delivery platform that helps users release software changes with high velocity and confidence.
- **CircleCI** is a continuous integration and delivery platform designed to make it easy for teams of all sizes to rapidly build and release quality software at scale.
- **GitLab CI** is also a continuous integration tool. It's built into GitLab, a platform for Git repository hosting and development tools.
- Then there is **Drone**, which is a modern CI/CD platform built with a container-first architecture. Running builds with Docker is at the core of Drone's design.

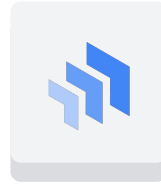
## Google Cloud's CI/CD tools



Artifact Registry



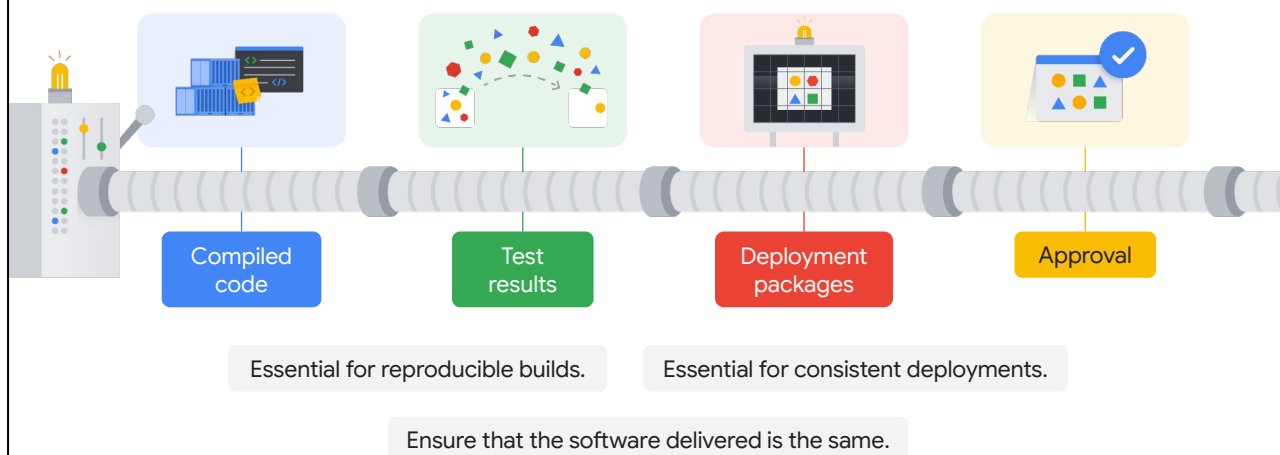
Cloud Build



Cloud Deploy

And then there are specific tools owned and operated by Google, including Artifact Registry, Cloud Build, and Cloud Deploy.

## Artifacts are the outputs produced at each stage



In CI/CD pipelines, artifacts are the outputs produced at each stage. They're typically compiled code, test results, and deployment packages.

Artifacts are essential for reproducible builds and consistent deployments, because they ensure that the software delivered is the same, regardless of the environment or time.

# Artifact Registry



Artifact Registry



A fully-managed, secure, and scalable artifact repository.



Stores, manages, and distributes build artifacts.



Can store artifacts of any type, including:



Source code



Binaries

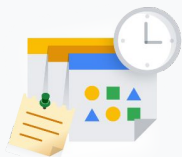


Docker images

Google's **Artifact Registry** is a fully-managed, secure, and scalable artifact repository for storing, managing, and distributing build artifacts.

Artifact Registry can be used to store artifacts of any type, including source code, binaries, and Docker images.

## Artifact Registry features

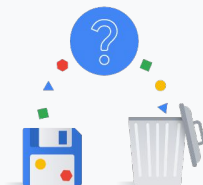


### Version control

Used to track changes over time.

Helpful for:

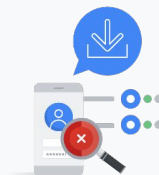
- Debugging
- Rollbacks



### Retention policies

A way to retain or delete articles for a period of time.

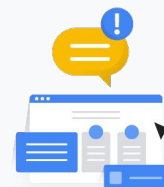
- Reduces the risk of data loss.
- Frees up storage.



### Access controls

Define who can view or download artifacts.

- Protects sensitive artifacts.



### Webhooks

Automated notifications sent to a specific URL.

- Keeps track of changes.

Artifact Register features help make it easy for users to manage their artifacts while getting the benefits of Google Cloud's infrastructure. Features include:

**Version control**, which can be used to track changes over time, which is helpful when debugging or if you need to roll back to a previous version of an artifact.

**Retention policies**, which provide a way to automatically retain artifacts for a designated period of time, or alternatively, automatically delete artifacts after a certain period of time. This can help reduce the risk of data loss, or free up storage space.

**Access controls**, which can be used to define who can view or download artifacts, which helps protect sensitive artifacts from unauthorized users.

And **webhooks**, which are automated notifications sent to a specific URL, let users know when an artifact is created, updated, or deleted. This is useful for keeping track of artifacts changes.



## Other important information about Artifact Registry



Artifact Registry



Built on top of Google Cloud Storage.



Supports a variety of artifact formats, including AAR, APK, JAR, POM, and ZIP.



Can be integrated with a variety of build tools, including Gradle, Maven, and Bazel.

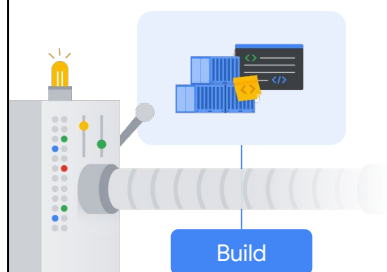


Available in all Google Cloud regions.

In addition, some important, general information to know about Artifact Registry is that:

- It's built on top of Google Cloud Storage, which is a highly scalable and reliable storage platform.
- It supports a variety of artifact formats, including AAR, APK, JAR, POM, and ZIP.
- It can be integrated with a variety of build tools, including Gradle, Maven, and Bazel.
- And it's available in all Google Cloud regions.

# Cloud Build



When raw source code is transformed into a deployable package.



Cloud Build

Can execute builds on:

- Google Cloud infrastructure.
- An on-premises environment.
- A combination of the two.

Next up is **Cloud Build**. In a CI/CD pipeline, the *build* stage refers to when raw source code is transformed into a deployable package. The process typically involves fetching dependencies, compiling the code, and packaging the application.

To help with this process, Google offers Cloud Build, which is a service that can execute builds on Google Cloud infrastructure, an on-premises environment, or a combination of the two.

# Cloud Build



Cloud Build



Can import source code from a variety of repositories or cloud storage space.



Can execute a build to defined specifications.



Can produce artifacts, such as container images.

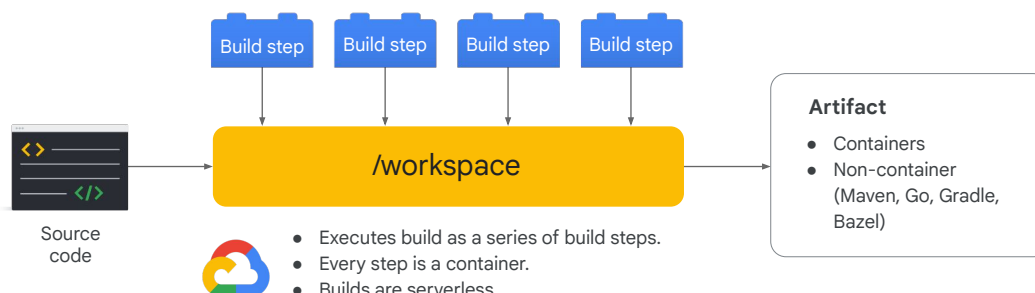


Can automatically start the build process in response to specific changes in your code.

Cloud Build can import source code from a variety of repositories or cloud storage spaces, execute a build to defined specifications, and produce artifacts, such as container images.

Cloud Build can be customized to automatically start the build process in response to specific changes in your code. This automation is achieved through pre-defined triggers, which define the specific events that will initiate a build.

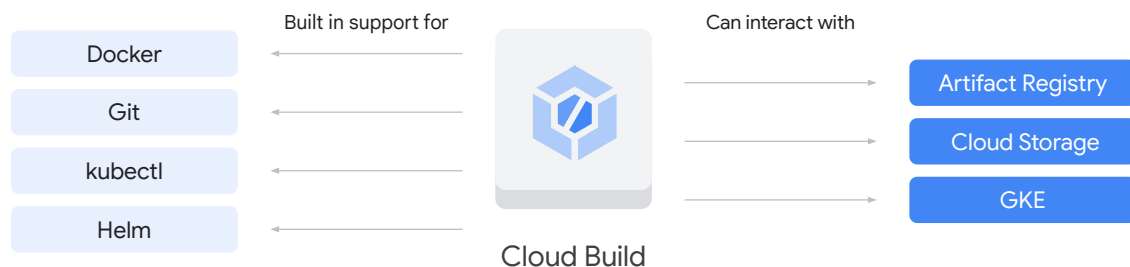
# The build process



The build process starts by mounting a common volume named `/workspace`. Then, the source code of an application is added to this volume.

Cloud Build creates each of the build steps as independent containers. This allows each content to operate on the common workspace. This environment reduces the need for redundant data stores.

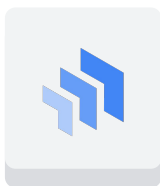
# Cloud Build features



To help deploy and manage complex Kubernetes applications, Cloud Build has built in support for tools like Docker, Git, kubectl, and Helm.

Cloud Build can also interact with other Google Cloud products, such as Artifact Registry, Cloud Storage, and GKE.

# Cloud Deploy



Cloud Deploy



Automates the deployment of applications to GKE.



Handles the continuous delivery (CD) stage.



Connects the build artifacts to the deployment environment.



Provides a single place to view and manage all of your deployments.



Can be integrated with other Google Cloud services (Cloud Monitoring, Cloud Logging).

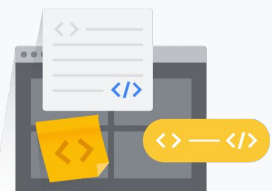
And finally, there is **Cloud Deploy**, which is a Google Cloud service used to automate the deployment of applications to Google Kubernetes Engine.

Cloud Deploy plays a crucial role in the CI/CD pipeline. While CI/CD focuses on automating the entire software delivery process, Cloud Deploy specifically handles the continuous delivery (CD) stage, seamlessly connecting the build artifacts to the deployment environment.

It provides a single place to view and manage all of your deployments, and can be integrated with other Google Cloud services, such as Cloud Monitoring and Cloud Logging.

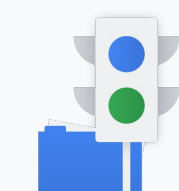
--

## Cloud Deploy features



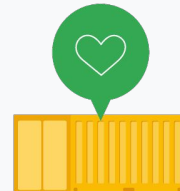
Can deploy apps from many sources:

- Source code repositories (GitHub and Bitbucket)
- Container images (Docker and Google Artifact Registry)
- Helm charts



Supports deployment methods:

- Blue/green deployments
- Rolling updates
- Canary deployments



Can help improve Kubernetes deployments in terms of:

- Reliability
- Scalability
- Security

Cloud Deploy can deploy applications from many sources, including source code repositories, such as GitHub and Bitbucket; container images, such as Docker and Google Artifact Registry, and Helm charts.

And Cloud Deploy also supports many deployment methods. The list includes blue/green deployments, rolling updates, and Canary deployments.

Cloud Deploy is a powerful and easy-to-use tool that can help you improve the reliability, scalability, and security of Kubernetes deployments.

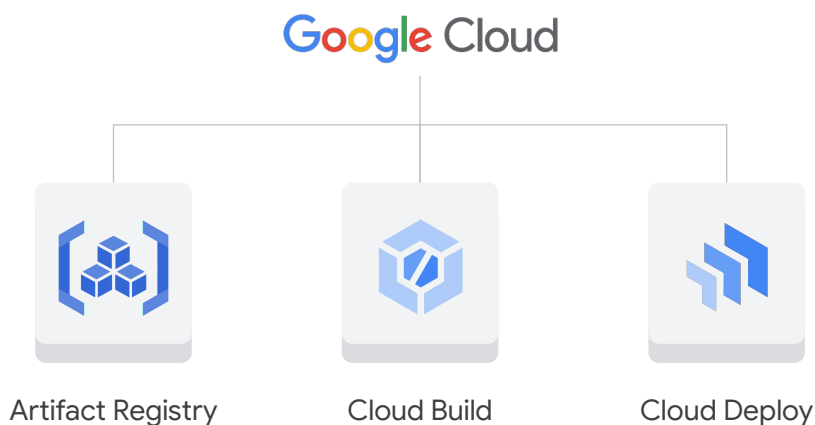
# Using CI/CD with Google Kubernetes Engine

- 01 Continuous integration and continuous deployment (CI/CD)
- 02 Constructing a CI/CD pipeline
- 03 CI/CD products
- 04** Best practices for using CI/CD on Google Cloud





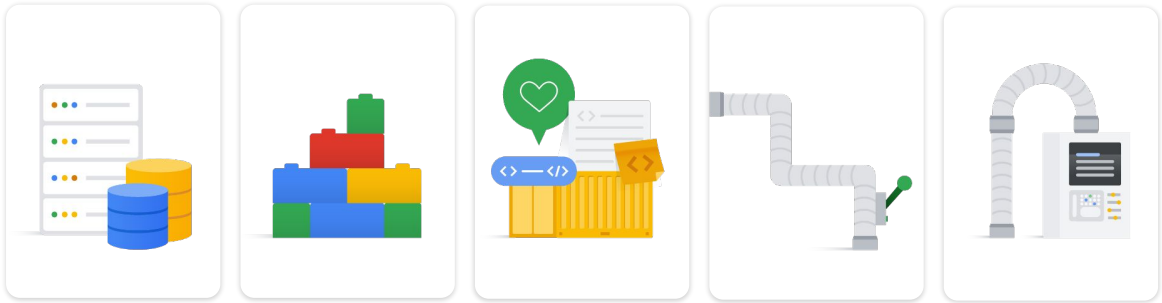
## CI/CD enables faster, more reliable software releases



The continuous integration and continuous delivery development technique, or CI/CD, helps software development teams ensure that code is of high quality and can be deployed to production reliably and quickly.

As we've just explored, Google offers a variety of tools and services to help teams implement CI/CD pipelines, including Artifact Registry, Cloud Build, and Cloud Deploy.

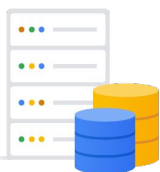
## CI/CD best practices



So, before you set off and start building CI/CD pipelines, you might consider some best practices.

## CI/CD best practices

Use a managed artifact repository.



Artifact  
Registry

A managed, secure, and scalable artifact repository.

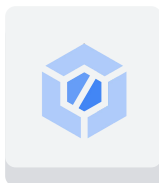
Stores, manages, and distributes build artifacts.

Helps improve the efficiency and reliability of your CI/CD pipeline.

**Use a managed artifact repository.** Google's Artifact Registry is a managed, secure, and scalable artifact repository that can be used to store, manage, and distribute build artifacts. This can help to improve the *efficiency and reliability* of your CI/CD pipeline.

## CI/CD best practices

Use a build service.



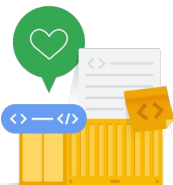
Cloud  
Build

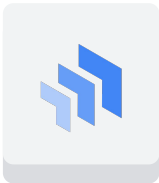
Helps to improve the scalability and flexibility of your CI/CD pipeline.

**Use a build service.** Google's Cloud Build is a service that can execute builds on Google Cloud infrastructure, an on-premises environment, or a combination of the two. This can help to improve the *scalability and flexibility* of your CI/CD pipeline.

## CI/CD best practices

Use a deployment service.





Cloud Deploy

Helps automate and simplify the deployment process.

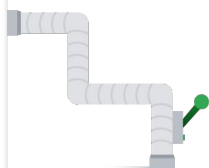
Can help improve the reliability and scalability of your applications.

**Use a deployment service.** Google's Cloud Deploy is a service that helps deploy and manage applications on Google Kubernetes Engine (GKE).

This can help to automate and simplify the deployment process, and can also help to improve the *reliability and scalability* of your applications.

## CI/CD best practices

Automate your pipeline.



Can help reduce the risk of human error.

Can help improve the efficiency of your pipeline.

**Automate your pipeline.** Take every opportunity to automate your CI/CD pipeline. Doing so can help reduce the risk of human error, and can also help to improve the efficiency of your pipeline.

## CI/CD best practices

Monitor your pipeline.



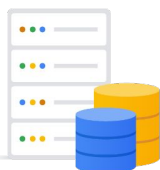
Helps identify and troubleshoot any problems.

Can help improve the performance of your pipeline.

**Monitor your pipeline.** Monitoring your CI/CD pipeline helps ensure that it's running smoothly and meeting your expectations. This can help you identify and troubleshoot any problems, and can also help you improve the performance of your pipeline.

## CI/CD best practices

Use a managed artifact repository.



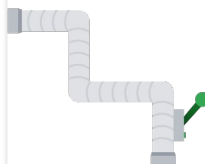
Use a build service.



Use a deployment service.



Automate your pipeline.



Monitor your pipeline.

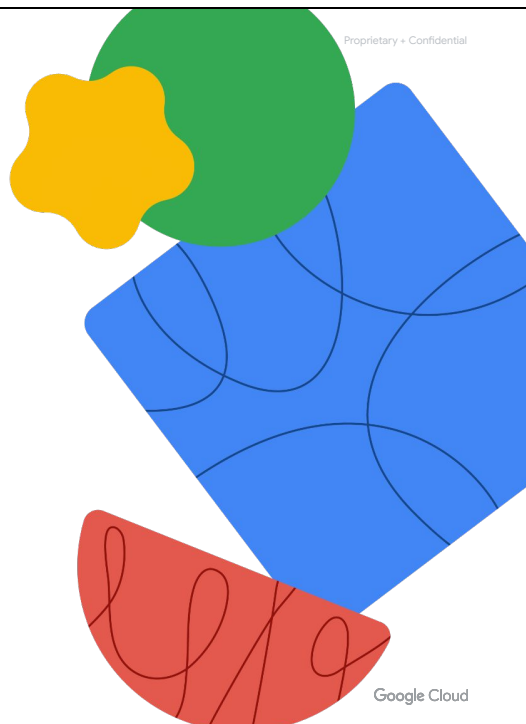


Taking the time to implement these best practices can help ensure the efficiency, reliability, and scalability of your CI/CD pipeline on Google Cloud.



# Quiz questions

Let's pause for a quick check in.



## Quiz | Question 1

### Question

Which of the following is not a benefit of CI/CD?

- A. Increased speed of software delivery.
- B. Improved quality of software.
- C. Reduced risk of breaking existing functionalities.
- D. More efficient use of resources.

## Quiz | Question 1

### Answer

Which of the following is not a benefit of CI/CD?

- A. Increased speed of software delivery.
- B. Improved quality of software.
- C. Reduced risk of breaking existing functionalities.
- D. More efficient use of resources.



Which of the following is not a benefit of CI/CD?

- A. Increased speed of software delivery.: This is the **incorrect answer** because CI/CD can increase the speed of software delivery.
- B. Improved quality of software.: This is the **incorrect answer** because CI/CD can improve the quality of software.
- C. Reduced risk of breaking existing functionalities.: This is the **incorrect answer** because CI/CD can reduced risk of breaking existing codebases.
- D. More efficient use of resources.:** This is the **correct answer** because CI/CD can require significant investment in tools, infrastructure, and training. It may not be the most efficient use of resources for all organizations or projects, especially smaller ones with limited resources.

## Quiz | Question 2

### Question

CI/CD pipelines exist along a spectrum of implementations. Google Cloud tools such as Cloud Build create pipelines at which level of the spectrum?

- A. Manual
- B. Built-in automation
- C. Dev-centric CI/CD
- D. Ops-centric CI/CD

## Quiz | Question 2

### Answer

CI/CD pipelines exist along a spectrum of implementations. Google Cloud tools such as Cloud Build create pipelines at which level of the spectrum?

- A. Manual
- B. Built-in automation
- C. Dev-centric CI/CD
- D. Ops-centric CI/CD



CI/CD pipelines exist along a spectrum of implementations. Google Cloud tools such as Cloud Build create pipelines at which level of the spectrum?

A. Manual: This is the **incorrect answer** because Cloud Build creates dev-centric CI/CD pipelines, not manual pipelines..

B. Built-in automation: This is the **incorrect answer** because Cloud Build offers more flexibility and customization than packaged tools with a fixed set of built-in automation. .

**C. Dev-centric CI/CD:** This is the **correct answer** because Cloud Build is used to create dev-centric CI/CD pipelines.

D. Ops-centric CI/CD: This is the **incorrect answer** because Cloud Build is designed for dev-centric CI/CD pipelines, which focus on the code repository as the central element. Ops-centric CI/CD prioritizes automating operational tasks and often uses tools like Spinnaker, which are designed for large-scale deployments..

## Quiz | Question 3

### Question

Which of the following is **not** a responsibility of Cloud Deploy?

- A. Configuring CI/CD pipelines.
- B. Ensuring that applications are up-to-date.
- C. Rolling back deployments.
- D. Deploying applications to Kubernetes clusters.

## Quiz | Question 3

### Answer

Which of the following is **not** a responsibility of Cloud Deploy?

- A. Configuring CI/CD pipelines.
- B. Ensuring that applications are up-to-date.
- C. Rolling back deployments.
- D. Deploying applications to Kubernetes clusters.



Which of the following is not a responsibility of Cloud Deploy?

- A. Configuring CI/CD pipelines.:** This is the **correct answer** because Cloud Deploy is focused on the deployment and delivery aspects of the CI/CD process..
- B. Ensuring that applications are up-to-date.:** This is the **incorrect answer** because ensuring applications are up-to-date is a key responsibility of Cloud Deploy, as it handles the continuous delivery stage of the CI/CD pipeline.
- C. Rolling back deployments:** This is the **incorrect answer** because Cloud Deploy supports rolling back deployments to a previous version in case of issues, making it a part of its responsibilities.
- D. Deploying applications to Kubernetes clusters.:** This is the **incorrect answer** because automating the deployment of applications to Google Kubernetes Engine is the primary function of Cloud Deploy.

## Quiz | Question 4

### Question

Which of the following is **not** a recommended practice for a GKE CI/CD pipeline?

- A. Using a version control system like Git to track changes.
- B. Manually testing the application after each build.
- C. Ensuring that the pipeline is automated as much as possible.
- D. Deploying the application to a staging environment before production.



## Quiz | Question 4

### Answer

Which of the following is **not** a recommended practice for a GKE CI/CD pipeline?

- A. Using a version control system like Git to track changes.
- B. Manually testing the application after each build.
- C. Ensuring that the pipeline is automated as much as possible.
- D. Deploying the application to a staging environment before production.



Which of the following is not a recommended practice for a GKE CI/CD pipeline?

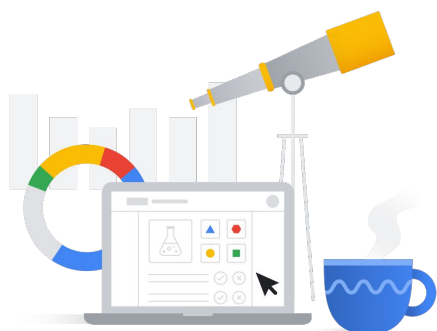
A. Using a version control system like Git to track changes.: This is the **incorrect answer** because using version control is essential for tracking changes and enabling rollbacks.

**B. Manually testing the application after each build.:** This is the **correct answer** because the core principle of CI/CD is automation to increase efficiency and reliability. Manual testing introduces delays and potential human error..

C. Ensuring that the pipeline is automated as much as possible.: This is the **incorrect answer** because automating pipelines as much as possible removes the potential for human error.

D. Deploying the application to a staging environment before production.: This is the **incorrect answer** because Deploying to a staging environment first helps catch issues before affecting production users.

# Lab: Continuous Delivery with Google Cloud Deploy



- 01 Deploy a container image to Google Cloud Artifact Registry using Skaffold
- 02 Create a Google Cloud Deploy delivery pipeline
- 03 Create a release for the delivery pipeline
- 04 Promote the application through the targets in the delivery pipeline.

It's time for some hands-on practice with GKE.

In the lab titled "Continuous Delivery with Google Cloud Deploy," you'll:

- Deploy a container image to Google Cloud Artifact Registry using Skaffold
- Create a Google Cloud Deploy delivery pipeline
- Create a release for the delivery pipeline
- Promote the application through the targets in the delivery pipeline