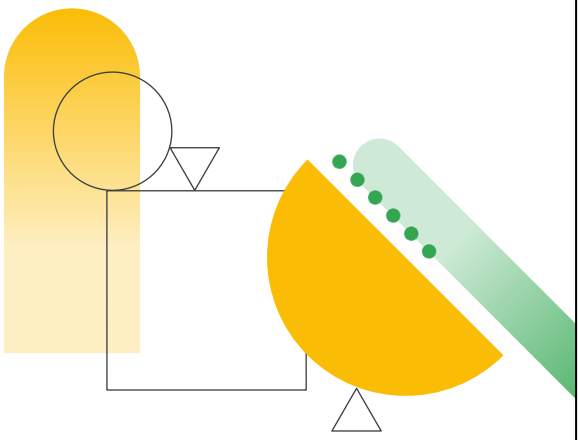
Google Cloud

DevOps Automation



This module introduces DevOps automation, a key factor in achieving consistency, reliability, and speed of deployment.

Learning objectives

01 Automate service deployment using CI/CD pipelines.

02 Leverage Cloud Source Repositories for source and version control.

03 Automate builds with Cloud Build and build triggers.

04 Manage container images with Artifact Registry.

05 Investigate infrastructure with code using Terraform.

This section defines the Google Cloud services that support continuous integration and continuous delivery practices, part of a DevOps way of working.

With DevOps and microservices, automated pipelines for integrating, delivering, and potentially deploying code are required. These pipelines ideally run on on-demand provisioned resources. This section introduces the Google tools for developing code and creating automated delivery pipelines that are provisioned on demand.

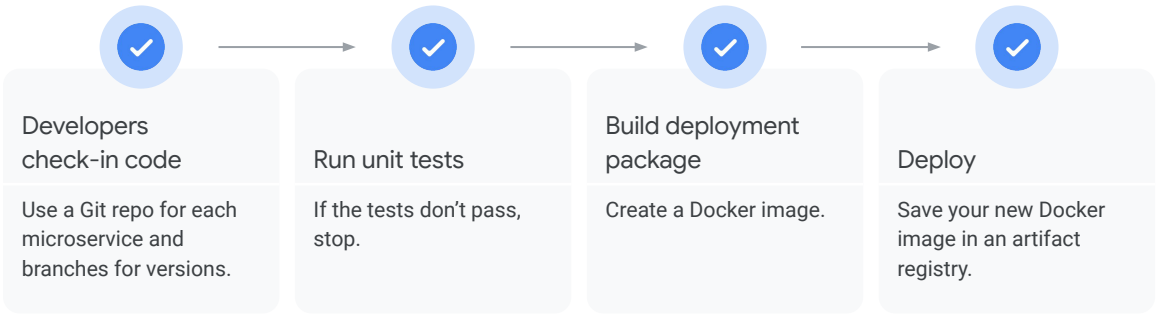
Agenda

01	Continuous Integration Pipelines
02	Infrastructure as Code
03	Lab
04	Quiz
05	Review

Continuous Integration Pipelines

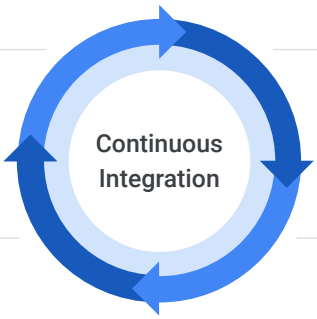
Let’s begin by talking about continuous integration pipelines.

Continuous integration pipelines automate building applications



This is a very simplistic view of a pipeline, which would be customized to meet your requirements. Typical extra steps include linting of code/quality analysis by tools such as SonarQube, integration tests, generating test reports, and image scanning.

Google provides the components required for a continuous integration pipeline



The **Cloud Source Repositories** service provides private Git repositories hosted on Google Cloud. These repositories let you develop and deploy an app or service in a space that provides collaboration and version control for your code. Cloud Source Repositories is integrated with Google Cloud, so it provides a seamless developer experience.

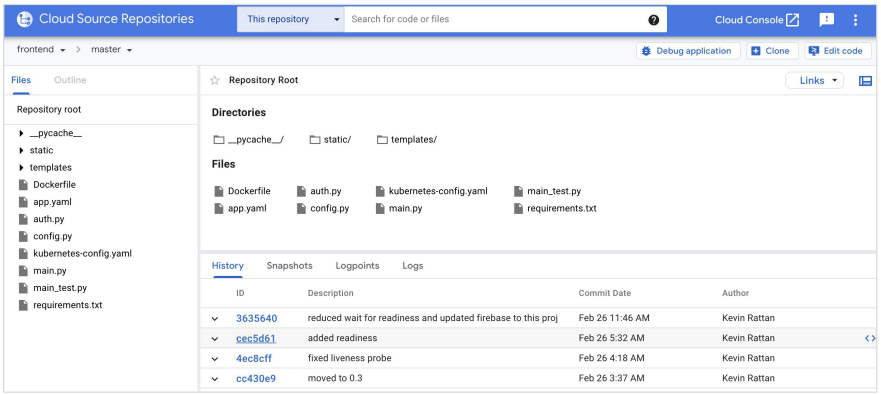
Cloud Build executes your builds on Google Cloud infrastructure. It can import source code from Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives. Cloud Build executes your build as a series of build steps, where each build step is run in a Docker container. A build step can do anything that can be done from a container, irrespective of the environment. There are standard steps, or you can define your own steps.

Build triggers A Cloud Build trigger automatically starts a build whenever you make any changes to your source code. You can configure the trigger to build your code on any changes to the source repository or only changes that match certain criteria.

Artifact Registry is a single place for your team to manage Docker images, perform vulnerability analysis, and decide who can access what with fine-grained access control.

Cloud Source Repositories provides managed Git repositories

Control access to your repos using IAM within your Google Cloud projects.



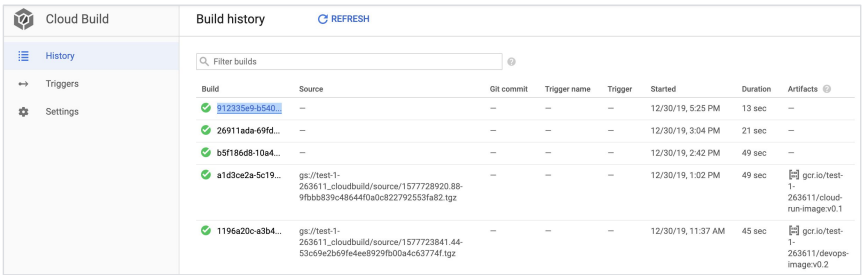
You can use IAM to add team members to your project and to grant them permissions to create, view, and update repositories.

Repositories can be configured to publish messages to a specified Pub/Sub topic. Messages can be published when a user creates or deletes a repository or pushes a commit.

Some other features of Cloud Source Repositories include the ability to use audit logging to provide insights into what actions were performed where and when, and direct deployment to App Engine. It is also possible to connect an existing GitHub or Bitbucket repository to Cloud Source Repositories. Connected repositories are synchronized with Cloud Source Repositories automatically.

Cloud Build lets you build software quickly across all languages

- Google-hosted Docker build service
  - Alternative to using Docker build command
- Use the CLI to submit a build
  `gcloud builds submit --tag gcr.io/your-project-id/image-name`



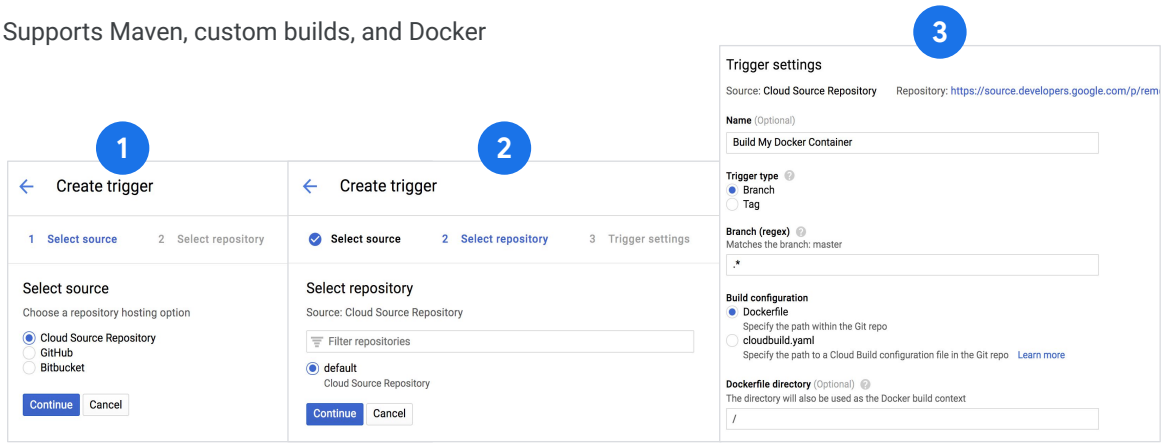
Developers gain complete control over defining workflows for building, testing, and deploying across multiple environments including VMs, serverless, and Kubernetes. There is no more need to provision or maintain build environments: all is handled by Cloud Build. You write a build config to provide instructions to Cloud Build on what tasks to perform. These are defined as as series of steps. Each step is executed by a Cloud builder. Cloud builders are containers with common languages and tools installed in them.
Builders can be configured to fetch dependencies, run unit tests, static analyses, and integration tests, and create artifacts with build tools such as docker, gradle, maven, bazel, and gulp. Cloud Build executes the build steps you define. Executing build steps is similar to executing commands in a script.

You can either use the build steps provided by Cloud Build and the Cloud Build community or write your own custom build steps:

Available builders can be found here:
https://github.com/GoogleCloudPlatform/cloud-builders

Build triggers watch a repository and build a container whenever code is pushed

Supports Maven, custom builds, and Docker



A Cloud Build trigger automatically starts a build whenever a change is made to source code. It can be set to start a build on commits to a particular branch or on commits that contain a particular tag. You can specify a regular expression to match the branch or tag value. The syntax for the regular expression is at https://github.com/google/re2/wiki/Syntax.

The build configuration can be specified either in a Dockerfile or a Cloud Build file. The configuration required is shown in the slide.

Artifact Registry is a universal package manager for build artifacts and dependencies

- Artifact Registry can store Docker and OCI container images in a Docker repository.
- Images built using Cloud Build are automatically saved in Artifact Registry.
  - Tag images with the prefix **`docker tag SOURCE-IMAGE LOCATION-docker.pkg.dev/PROJECT-ID/REPOSITORY/IMAGE:TAG`**
- Can use Docker push and pull commands with Artifact Registry.
  - `docker push LOCATION-docker.pkg.dev/PROJECT-ID/REPOSITORY/IMAGE:TAG`
  - `docker pull LOCATION-docker.pkg.dev/PROJECT-ID/REPOSITORY/IMAGE:TAG`

Artifact Registry is a universal package manager for build artifacts and dependencies. Artifact Registry can store Docker and OCI container images in a Docker repository.

Artifact Registry integrates with Google Cloud CI/CD services or your existing CI/CD tools. You can store artifacts from Cloud Build and deploy artifacts to Google Cloud runtimes, including Google Kubernetes Engine (GKE), Cloud Run, Compute Engine, and App Engine flexible environment. Identity and Access Management provides consistent credentials and access control.

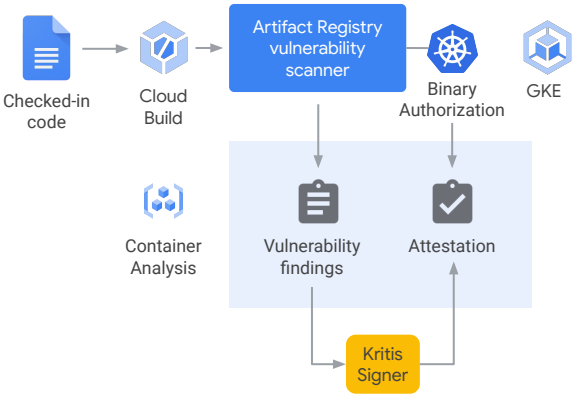
Artifact Registry implements a Docker protocol so that you can push and pull images directly with Docker clients, including the Docker command-line tool.

Google Cloud services that typically integrate with Artifact Registry, such as Cloud Build and Google Kubernetes Engine, are configured by with default permissions to access repositories in the same project and do not require a separate client.

Artifact Analysis is a family of services that provide software composition analysis, metadata storage and retrieval. Its detection points are built into a number of Google Cloud products such as Artifact Registry and Google Kubernetes Engine for easy and quick enablement. The service works with both Google Cloud's first-party products and also allows you to store information from third-party sources. The scanning services leverage a common vulnerability store for matching files against known vulnerabilities.

Binary authorization allows you to enforce deploying only trusted containers into GKE

- Enable binary authorization on GKE cluster.
- Add a policy that requires signed images.
- When an image is built by Cloud Build an “attestor” verifies that it was from a trusted repository (Source Repositories, for example).
- Artifact Registry includes a vulnerability scanner that scans containers.



Binary authorization ensures that internal processes that safeguard the quality and integrity of your software have been successfully completed before an application is deployed to your production environment. Binary authorization is a Google Cloud service and is based on the Kritis specification: https://github.com/grafeas/kritis/blob/master/docs/binary-authorization.md

The Kritis signer listens to Pub/Sub notifications from an artifact registry vulnerability scanner when new image versions are uploaded and makes an attestation if the image passed the vulnerability scan. Google Cloud binary authorization service then enforces the policy requiring attestations by the Kritis signer before a container image can be deployed.

This flow prevents deployment of images with vulnerabilities below a certain threshold.

More details can be found at:

https://cloud.google.com/binary-authorization/docs/cloud-build

https://cloud.google.com/binary-authorization/docs/vulnerability-scanning

Infrastructure as Code

Moving to the cloud requires a mindset change

On-Premises

- Buy machines.
- Keep machines running for years.
- Prefer fewer big machines.
- Machines are capital expenditures.

Cloud

- Rent machines.
- Turn machines off as soon as possible.
- Prefer lots of small machines.
- Machines are monthly expenses.

The on demand, pay-per-use model of cloud computing is a different model to traditional on-premises infrastructure provisioning. Resources can be allocated to best meet demand in a timely manner, and the cloud supports experimentation and innovation by providing immediate access to an ever-increasing range of services.

In the cloud, all infrastructure needs to be disposable

- Don't fix broken machines.
- Don't install patches.
- Don't upgrade machines.
- If you need to fix a machine, delete it and re-create a new one.

- To make infrastructure disposable, automate everything with code:
  - Can automate using scripts.
  - Can use declarative tools to define infrastructure.

The key term is infrastructure as code (IaC). The provisioning, configuration, and deployment activities should all be automated.

Having the process automated minimizes risks, eliminates manual mistakes, and supports repeatable deployments and scale and speed. Deploying one or one hundred machines is the same effort.

Costs can be reduced by provisioning ephemeral environments, such as test environments that replicate the production environment.

Infrastructure as code (IaC) allows for the quick provisioning and removing of infrastructures

- Build an infrastructure when needed.
- Destroy the infrastructure when not in use.
- Create identical infrastructures for dev, test, and prod.
- Can be part of a CI/CD pipeline.
- Templates are the building blocks for disaster recovery procedures.
- Manage resource dependencies and complexity.
- Google Cloud supports many IaC tools.

Terraform is one of the tools used for Infrastructure as Code or IaC. Before we dive into understanding Terraform, let's look at what infrastructure as code does. In essence, infrastructure as code allows for the quick provisioning and removing of infrastructures.

The on-demand provisioning of a deployment is extremely powerful. This can be integrated into a continuous integration pipeline that smoothes the path to continuous deployment.

Automated infrastructure provisioning means that the infrastructure can be provisioned on demand, and the deployment complexity is managed in code. This provides the flexibility to change infrastructure as requirements change. And all the changes are in one place. Infrastructure for environments such as development and test can now easily replicate production and can be deleted immediately when not in use. All because of infrastructure as code.

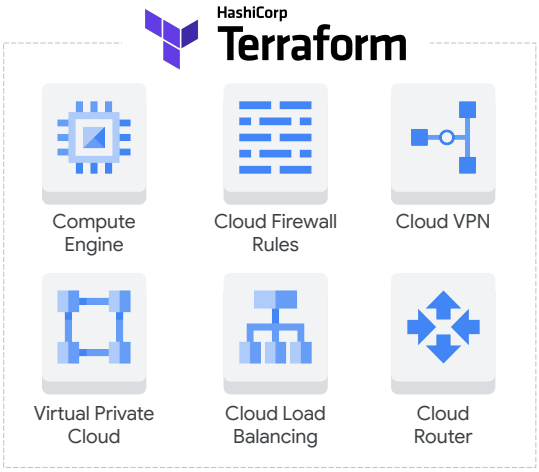
Several tools can be used for IaC. Google Cloud supports Terraform, where deployments are described in a file known as a configuration. This details all the resources that should be provisioned. Configurations can be modularized using templates, which allows the abstraction of resources into reusable components across deployments.

In addition to Terraform, Google Cloud also provides support for other IaC tools, including:

- Chef
- Puppet
- Ansible
- Packer

Terraform is an infrastructure automation tool

- Repeatable deployment process
- Declarative language
- Focus on the application
- Parallel deployment
- Template-driven



Terraform lets you provision Google Cloud resources—such as virtual machines, containers, storage, and networking—with declarative configuration files. You just specify all the resources needed for your application in a declarative format and deploy your configuration. HashiCorp Configuration Language (HCL) allows for concise descriptions of resources using blocks, arguments, and expressions.

This deployment can be repeated over and over with consistent results, and you can delete an entire deployment with one command or click. The benefit of a declarative approach is that it allows you to specify what the configuration should be and let the system figure out the steps to take.

Instead of deploying each resource separately, you specify the set of resources that compose the application or service, which allows you to focus on the application. Unlike Cloud Shell, Terraform will deploy resources in parallel.

Terraform uses the underlying APIs of each Google Cloud service to deploy your resources. This enables you to deploy almost everything we have seen so far, from instances, instance templates, and groups, to VPC networks, firewall rules, VPN tunnels, Cloud Routers, and load balancers. For a full list of supported resource types, see the documentation at Using Terraform with Google Cloud.

Terraform language

- Terraform language is the interface to declare resources.
- Resources are infrastructure objects.
- The configuration file guides the management of the resource.

```
resource "google_compute_network" "default" {
  name = "${var.network_name}"
  auto_create_subnetworks = false
}

<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```


The Terraform language is the user interface to declare resources. Resources are infrastructure objects such as Compute Engine virtual machines, storage buckets, containers, or networks. A Terraform configuration is a complete document in the Terraform language that tells Terraform how to manage a given collection of infrastructure. A configuration can consist of multiple files and directories.

The syntax of the Terraform language includes:

- Blocks that represent objects and can have zero or more labels. A block has a body that enables you to declare arguments and nested blocks.
- Arguments are used to assign a value to a name.
- An expression represents a value that can be assigned to an identifier.

Terraform can be used on multiple public and private clouds

- Considered a first-class tool in Google Cloud
- Already installed in Cloud Shell

```
provider “google” {
   region      = “us-central1”
}

resource “google_compute_instance” {
   name        = “instance name”
   machine_type= “n1-standard-1”
   zone        = “us-central1-f”

   disk {
      image = “image to build instance”
   }
}

output “instance_ip” {
   value = “${google_compute.ip_address}”
}
```


Terraform can be used on multiple public and private clouds. Terraform is already installed in Cloud Shell.

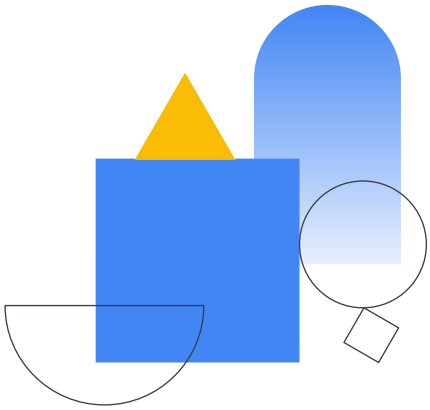
The example Terraform configuration file shown starts with a provider block that indicates that Google Cloud is the provider. The region for the deployment is specified inside the provider block.

The resource block specifies a Google Cloud Compute Engine instance, or virtual machine. The details of the instance to be created are specified inside the resource block.

The output block specifies an output variable for the Terraform module. In this case, a value will be assigned to the output variable "instance_ip."

Lab Intro

Building a DevOps Pipeline



In this lab, you will build a continuous integration pipeline using Cloud Source Repositories, Cloud Build, build triggers, and Artifact Registry.

Lab objectives

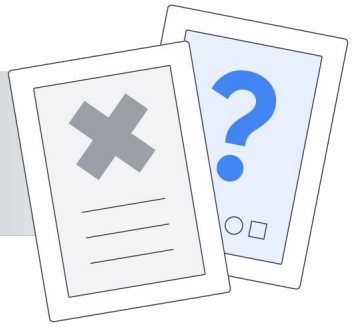
01 Create a Git repository.

02 Create a simple Python application and test your web application in Cloud Shell.

03 Define a Docker build.

04 Manage Docker images with Cloud Build and Artifact Registry.

05 Automate builds with triggers and test your build changes.



Quiz

Question #1

Question

Which Google Cloud tools can be used to build a continuous integration pipeline?

A. Cloud Source Repositories

B. Cloud Build

C. Artifact Registry

D. All of the above

Which Google Cloud tools can be used to build a continuous integration pipeline?
A. Cloud Source Repositories
B. Cloud Build
C. Artifact Registry
D. All of the above

Question #1

Answer

Which Google Cloud tools can be used to build a continuous integration pipeline?

A. Cloud Source Repositories

B. Cloud Build

C. Artifact Registry

D. **All of the above**

All the above answers are correct. Source Repositories provides a private Git repository, Cloud Build builds containers, and Artifact Registry is a Docker images repository that performs vulnerability analysis. All three components are typically used in a continuous integration pipeline where on a commit, code is built and tested and an image is built and published to a registry.

Question #2

Question

List some reasons to automate infrastructure creation using code tools like Terraform.

List some reasons to automate infrastructure creation using code tools like Terraform.

Question #2

Answer

List some reasons to automate infrastructure creation using code tools like Terraform.

- Easier to make dev, test, and prod environments the same
- Easier to change and fix infrastructure over time
- Simplify administration
- Automate provisioning and decommissioning
- Save money

The test and deployment environments should be the same or as close as possible to being the same. Automating the provisioning of these reduces errors that are likely to occur with manual processes, making it easy to change the infrastructure as requirements and technologies change. Automating also means administration and permissions are much easier to manage through IAM. Once automated, the provisioning is simplified and repeatable and the decommissioning is automated too, which means that the costs for the infrastructure are only incurred while it is in use.

Question #3

Question

What Google Cloud feature would be easiest to use to automate a build in response to code being checked into your source code repository?

A. Build triggers

B. Cloud Run functions

C. App Engine

D. Cloud Scheduler

What Google Cloud feature would be easiest to use to automate a build in response to code being checked into your source code repository?

A. Cloud Build triggers
B. Cloud Run functions
C. App Engine
D. Cloud Scheduler

Question #3

Answer

What Google Cloud feature would be easiest to use to automate a build in response to code being checked into your source code repository?

A. Build triggers 

B. Cloud Run functions

C. App Engine

D. Cloud Scheduler

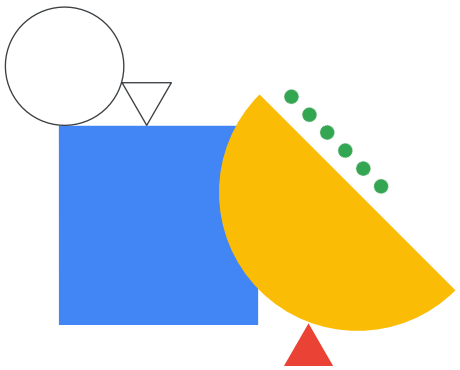
A. This answer is correct. Cloud Build triggers have been designed specifically to trigger a build automatically when changes are made to source code.

B. This answer is not correct. Although these could potentially be used to build a solution, for example, with Webhooks, this is not the use case for Cloud Run functions, which is for serverless backends and has built-in support for scalability.

C. This answer is not correct. App Engine is for deploying applications often deployed as microservices and is a deployment platform.

D. This answer is not correct. Cloud Scheduler is an enterprise-grade scheduler. Automating a build is an event-triggered process, not a scheduled process.

Review: DevOps Automation



In this module, you learned about services you can use to help automate the deployment of your cloud resources and services. You used Cloud Source Repositories, Cloud Build, triggers, and Artifact Registry to create continuous integration pipelines. A CI pipeline automates the creation of deployment packages like Docker images in response to changes in your source code.

You also saw how to automate the creation of infrastructure using infrastructure as code tools like Terraform.

More resources

Google Cloud DevOps solutions

https://cloud.google.com/devops/

Terraform on Google Cloud

https://cloud.google.com/community/tutorials/getting-started-on-gcp-with-terraform

Here is a list of useful resources to find out more details on subjects discussed in this section.


Google Cloud