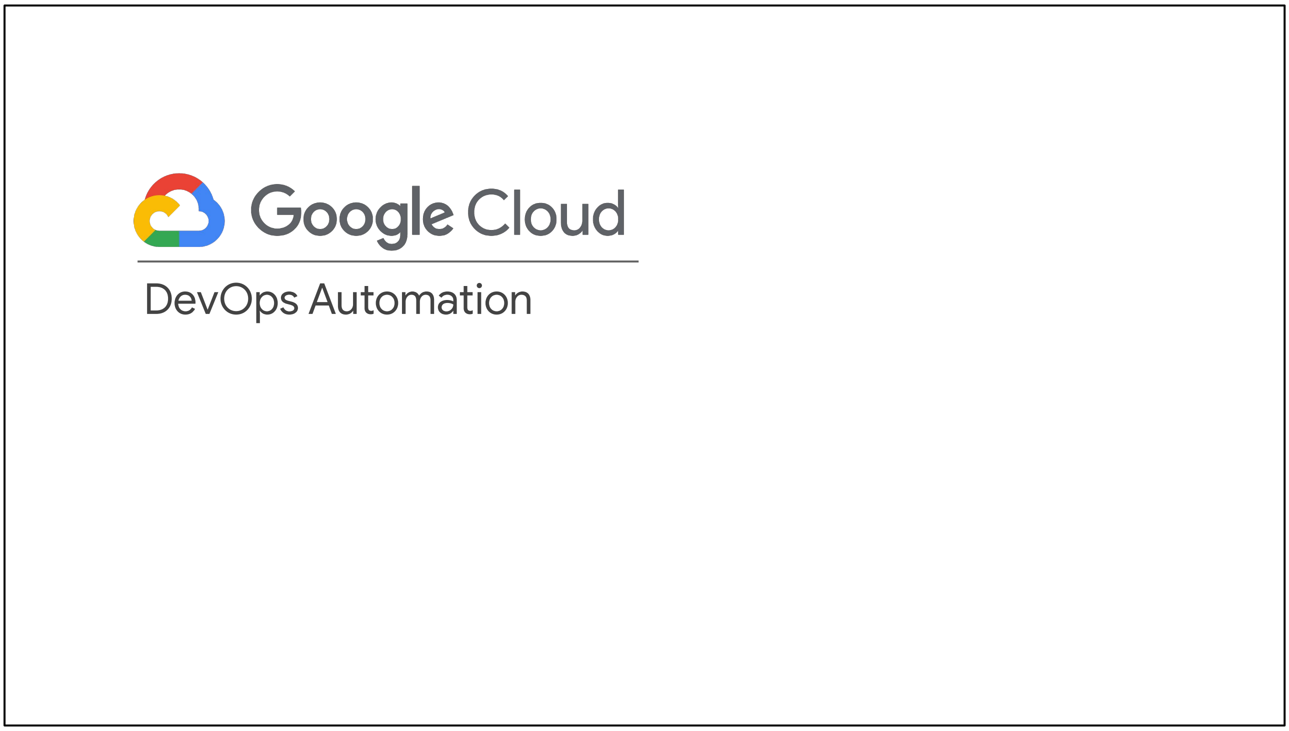
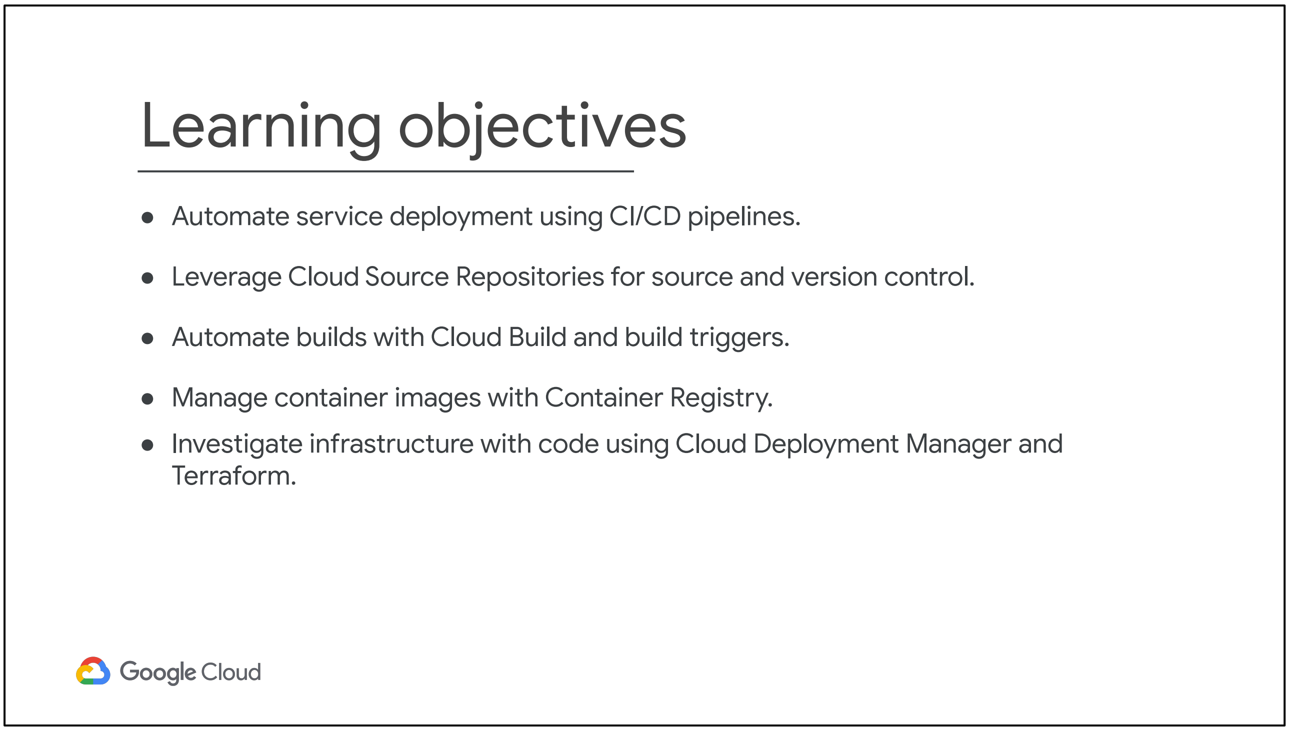


Google Cloud

DevOps Automation

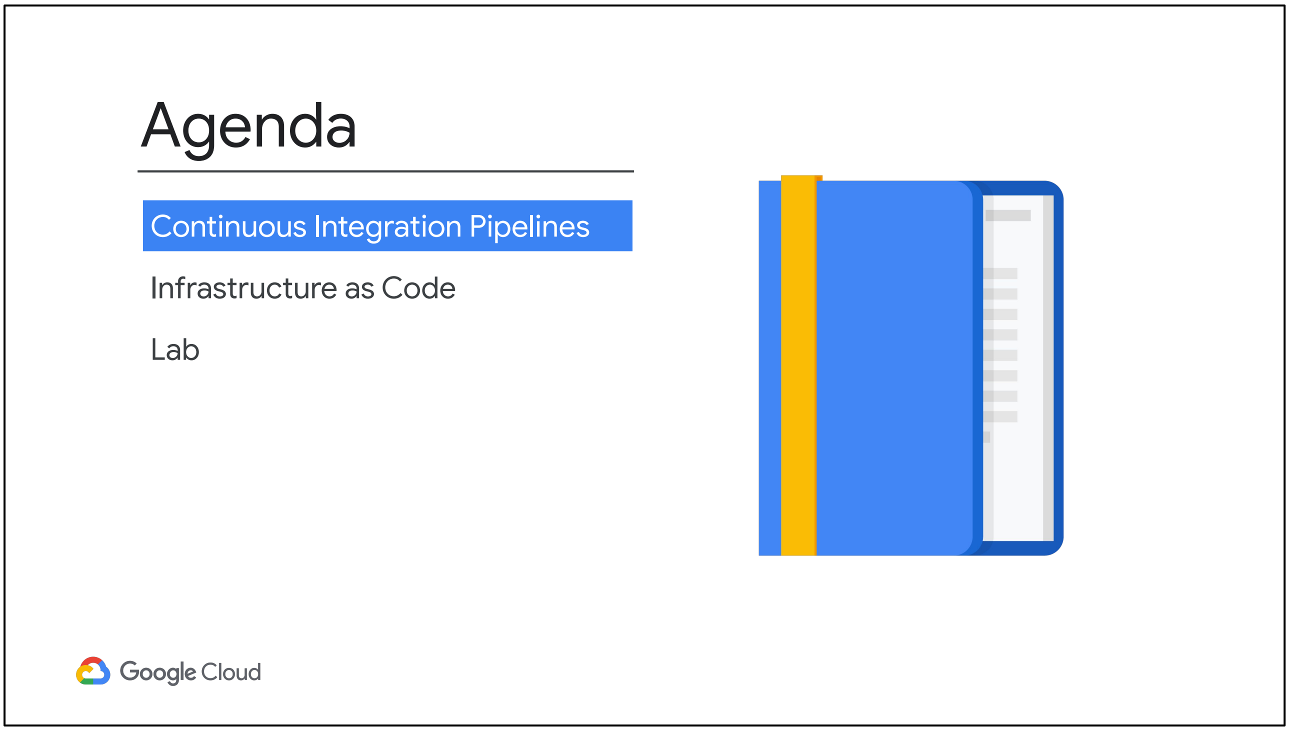
This module introduces DevOps automation, a key factor in achieving consistency, reliability, and speed of deployment.



Learning objectives

- Automate service deployment using CI/CD pipelines.
- Leverage Cloud Source Repositories for source and version control.
- Automate builds with Cloud Build and build triggers.
- Manage container images with Container Registry.
- Investigate infrastructure with code using Cloud Deployment Manager and 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

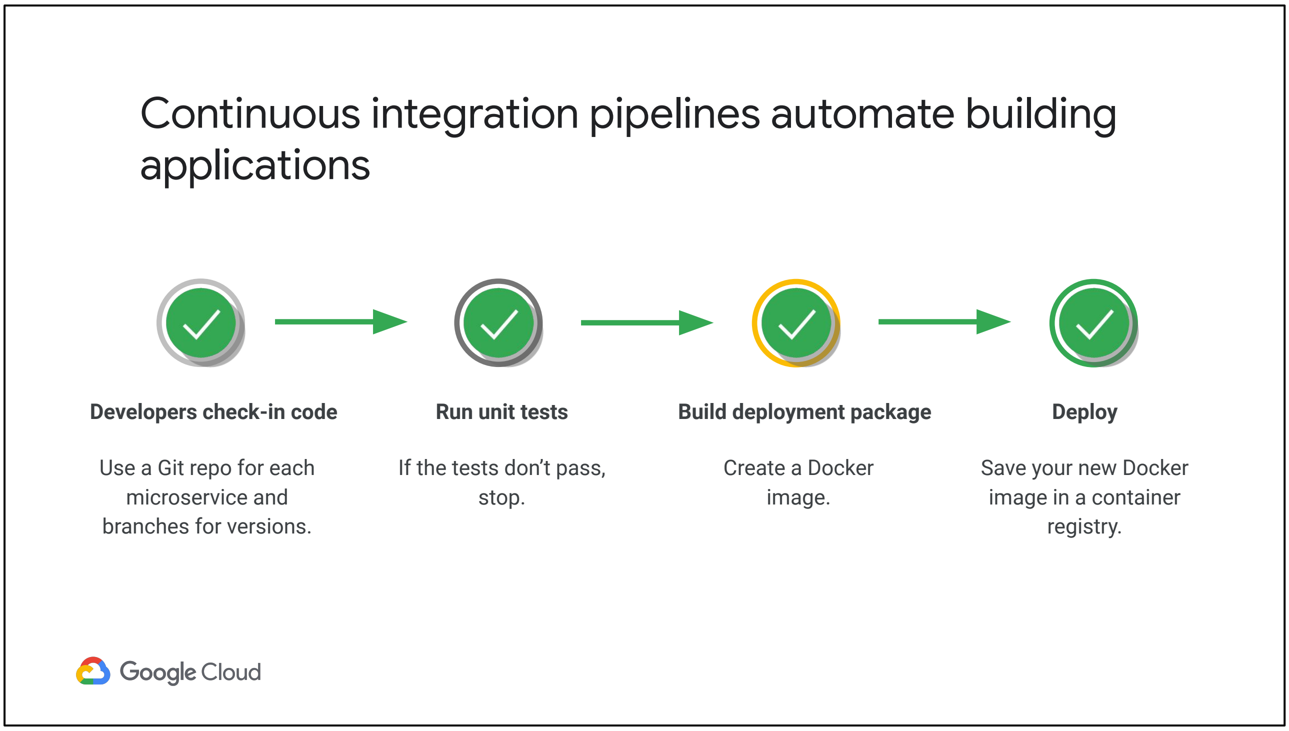
Continuous Integration Pipelines

Infrastructure as Code

Lab

Google Cloud

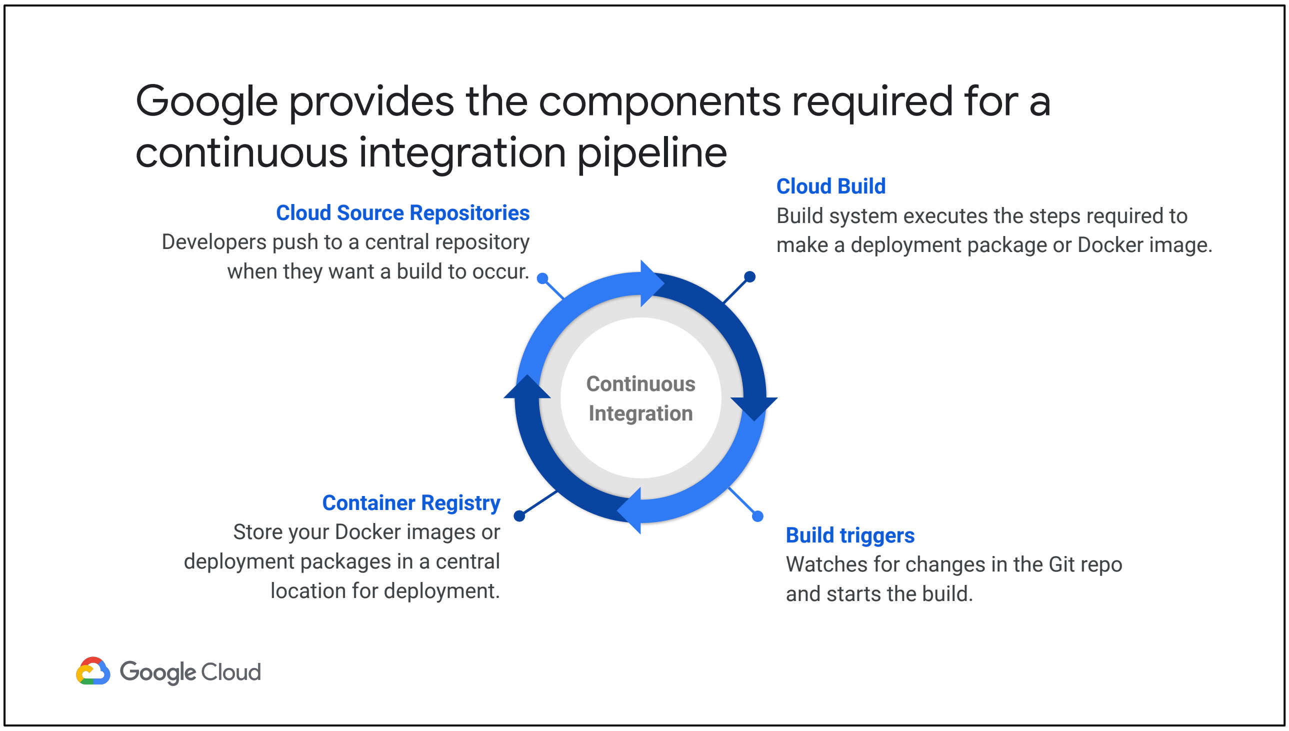
Let’s begin by talking about continuous integration pipelines.



Continuous integration pipelines automate building applications

Developers check-in code
Use a Git repo for each microservice and branches for versions.
Run unit tests
If the tests don't pass, stop.
Build deployment package
Create a Docker image.
Deploy
Save your new Docker image in a container registry.
Google Cloud

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

Cloud Source Repositories
Developers push to a central repository when they want a build to occur.

Cloud Build
Build system executes the steps required to make a deployment package or Docker image.

Container Registry
Store your Docker images or deployment packages in a central location for deployment.

Build triggers
Watches for changes in the Git repo and starts the build.

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.

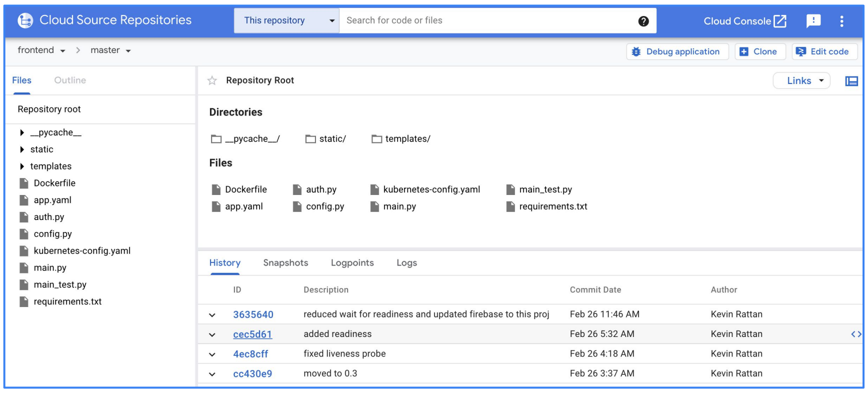
Cloud Build executes your builds on Google Cloud infrastructure. It can import source code from Cloud Storage, Cloud Source Repositories, GitHub, or Bitbucket, execute a build to your specifications, and produce artifacts such as Docker containers or Java archives. Cloud Build executes your build as 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.

Container 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.



Directories

__pycache__/ static/ templates/

Files

Dockerfile auth.py kubernetes-config.yaml main_test.py
app.yaml config.py main.py requirements.txt

ID	Description	Commit Date	Author
3635640	reduced wait for readiness and updated firebase to this proj	Feb 26 11:46 AM	Kevin Rattan
cec5d61	added readiness	Feb 26 5:32 AM	Kevin Rattan
4ec8cff	fixed liveness probe	Feb 26 4:18 AM	Kevin Rattan
cc430e9	moved to 0.3	Feb 26 3:37 AM	Kevin Rattan

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

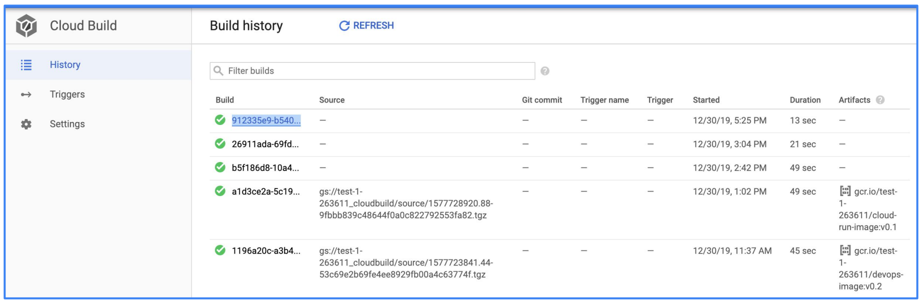
Repositories can be configured to publish messages to a specified Pub/Sub topic. Messages can be published when a user creates or deletes a repository or pushes a commit.

Some other features of Cloud Source Repositories include the ability to debug in production using Cloud Debugger, 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 .
```



Build	Source	Git commit	Trigger name	Trigger	Started	Duration	Artifacts
✓ 912335e9-b540...	–	–	–	–	12/30/19, 5:25 PM	13 sec	–
✓ 26911ada-69fd...	–	–	–	–	12/30/19, 3:04 PM	21 sec	–
✓ b5f186d8-10a4...	–	–	–	–	12/30/19, 2:42 PM	49 sec	–
✓ a1d3ce2a-5c19...	gs://test-1-263611_cloudbuild/source/1577728920.88-9fbbb839c48644f0a0c822792553fa82.tgz	–	–	–	12/30/19, 1:02 PM	49 sec	gcr.io/test-1-263611/cloud-run-image:v0.1
✓ 1196a20c-a3b4...	gs://test-1-263611_cloudbuild/source/1577723841.44-53c69e2b69fe4ee8929fb00a4c63774f.tgz	–	–	–	12/30/19, 11:37 AM	45 sec	gcr.io/test-1-263611/devops-image:v0.2

Google Cloud

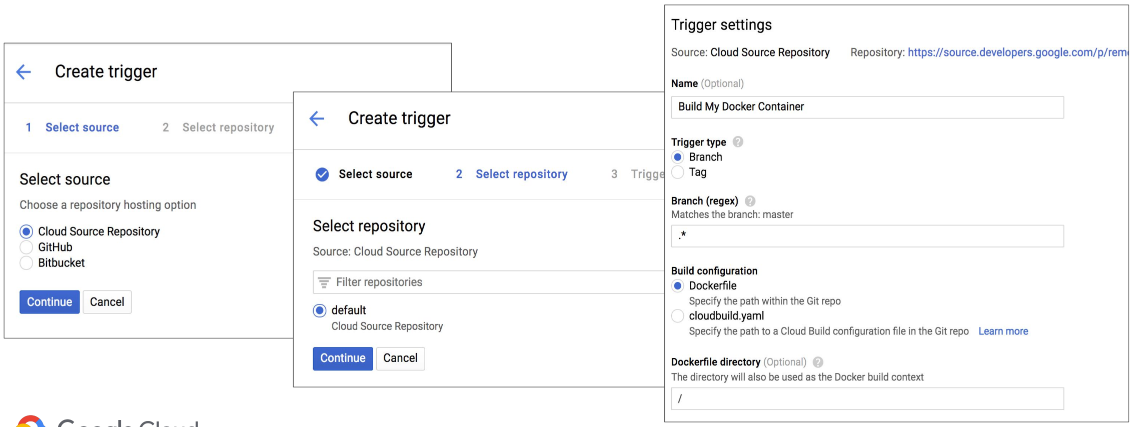
Developers gain complete control over defining workflows for building, testing, and deploying across multiple environments including VMs, serverless, and Kubernetes. There is no more need to provision or maintain build environments: all is handled by Cloud Build. You write a build config to provide instructions 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



Create trigger

1 Select source　2 Select repository

Select source

Choose a repository hosting option

- [✓ Cloud Source Repository]
- GitHub
- Bitbucket

Continue　Cancel

Create trigger

✓ Select source　2 Select repository　3 Trigge

Select repository

Source: Cloud Source Repository

Filter repositories

- [✓ default]
  Cloud Source Repository

Continue　Cancel

Trigger settings

Source: Cloud Source Repository　Repository: https://source.developers.google.com/p/rem

Name (Optional)

Build My Docker Container

Trigger type

- [✓ Branch]
- Tag

Branch (regex)

Matches the branch: master

.*

Build configuration

- [✓ Dockerfile]
  Specify the path within the Git repo
- cloudbuild.yaml
  Specify the path to a Cloud Build configuration file in the Git repo　Learn more

Dockerfile directory (Optional)

The directory will also be used as the Docker build context

/

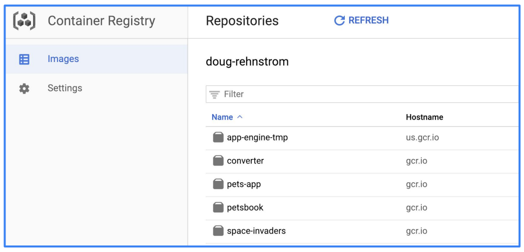
Google Cloud

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 with the branch or tag value to match. 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.

Container Registry is a Google Cloud–hosted Docker repository

- Images built using Cloud Build are automatically save in Container Registry.
  - Tag images with the prefix **`gcr.io/your-project-id`**`/image-name`
- Can use Docker push and pull commands with Container Registry.
  - `docker push gcr.io/your-project-id/image-name`
  - `docker pull gcr.io/your-project-id/image-name`



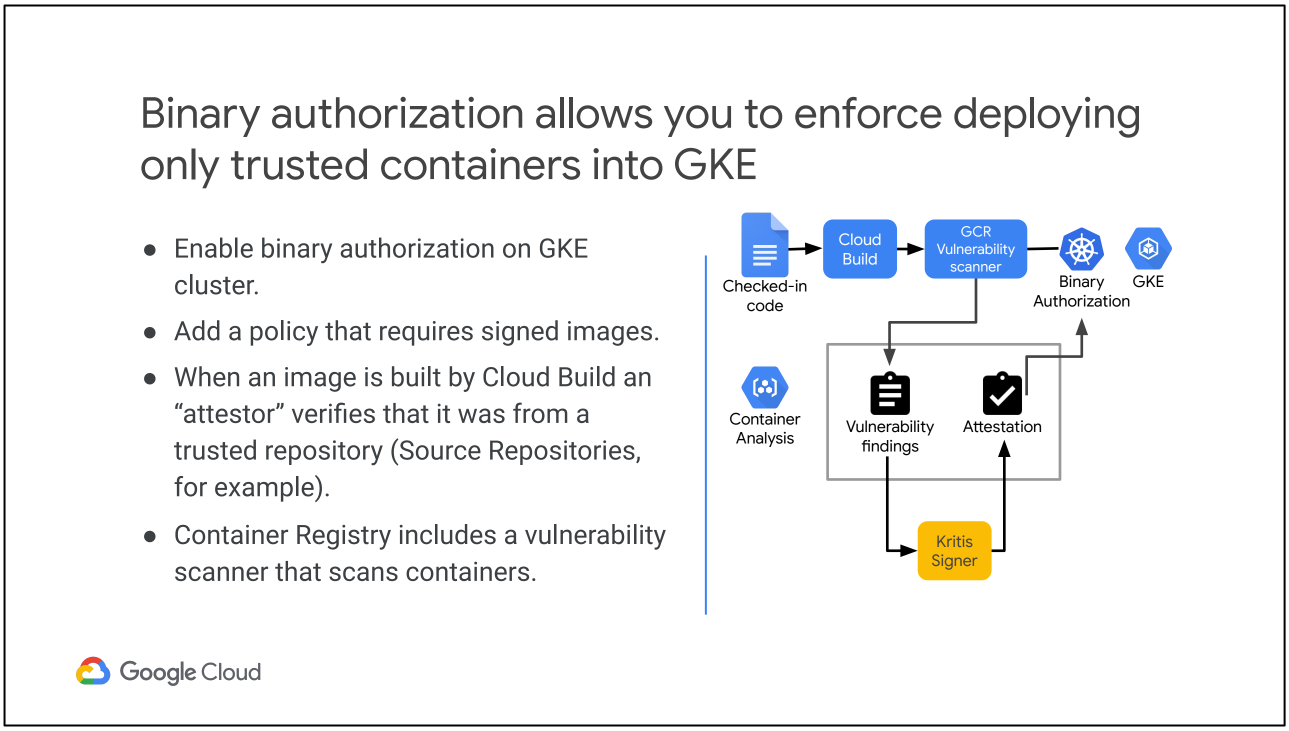
doug-rehnstrom

Name	Hostname
app-engine-tmp	us.gcr.io
converter	gcr.io
pets-app	gcr.io
petsbook	gcr.io
space-invaders	gcr.io

Google Cloud

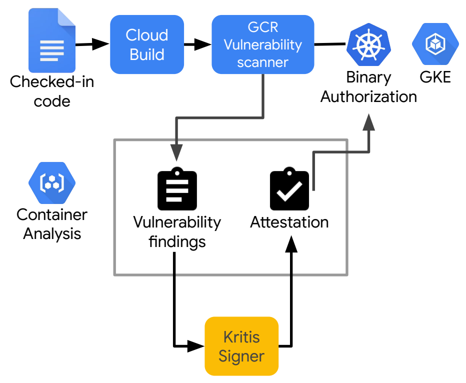
Container Registry provides a secure, private Docker image repository on Google Cloud. Image layer files and tags are stored in Cloud Storage. Cloud Storage IAM allows configuration of who can access, view, or download images. Container Registry provides native Docker support. This means you can push and pull Docker images to your private registry using the standard Docker command line interface. Multiple registries can be defined.

Container Analysis is a service that can also perform vulnerability scanning of images against a constantly updated database of known vulnerabilities. Container Analysis can be integrated with binary authorization to prevent the deployment of images with known security issues. Container Analysis must be explicitly enabled for a project.



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).
- Container Registry includes a vulnerability scanner that scans containers.



Google Cloud

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 a container 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

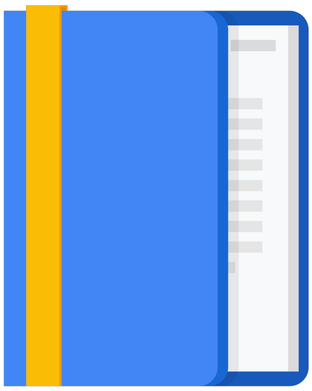
https://cloud.google.com/binary-authorization/docs/vulnerability-scanning

Agenda

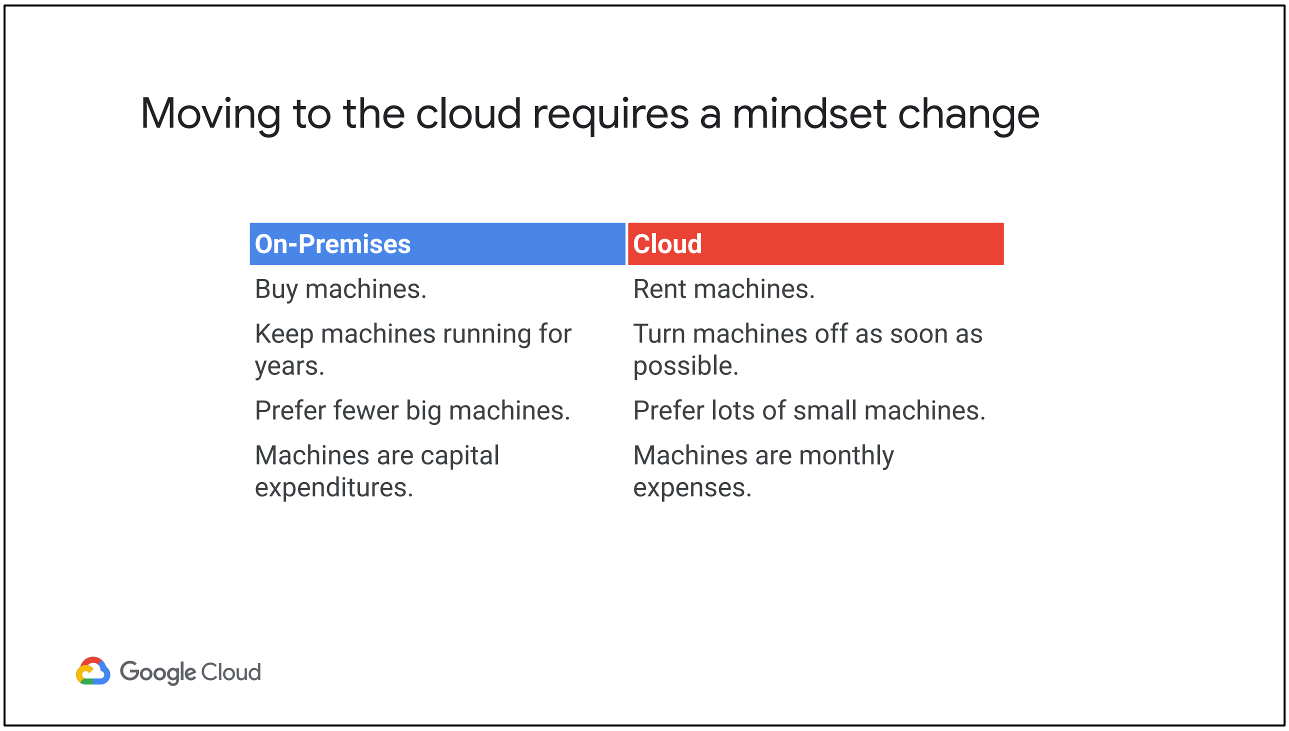
Continuous Integration Pipelines

Infrastructure as Code

Lab



Google Cloud

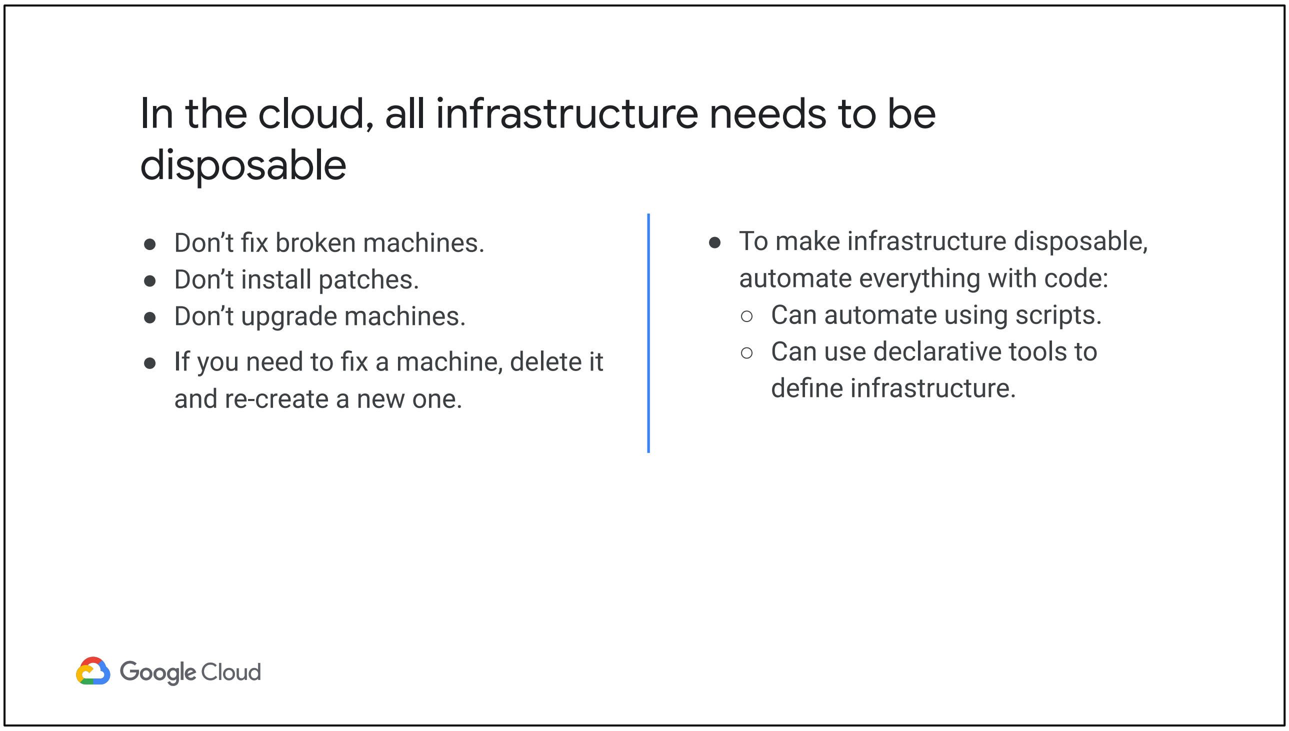


Moving to the cloud requires a mindset change

On-Premises	Cloud
Buy machines.	Rent machines.
Keep machines running for years.	Turn machines off as soon as possible.
Prefer fewer big machines.	Prefer lots of small machines.
Machines are capital expenditures.	Machines are monthly expenses.

Google Cloud

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.

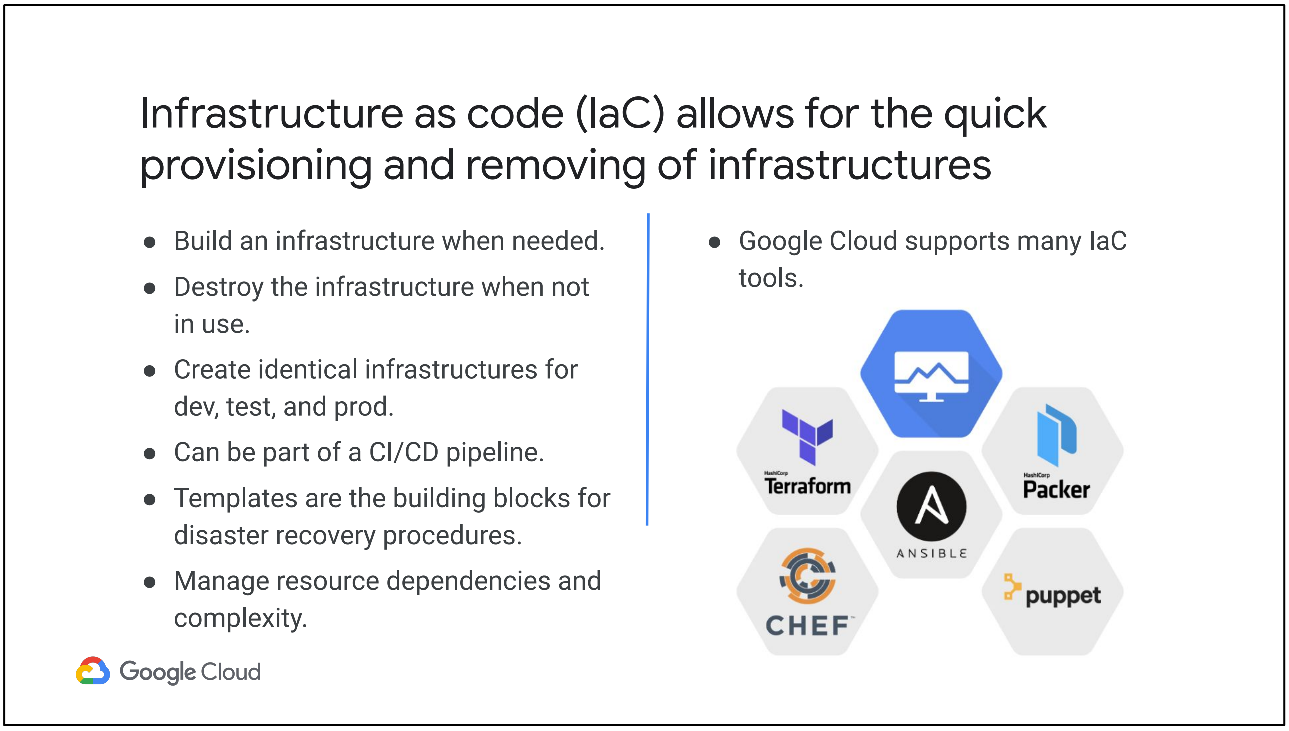
- To make infrastructure disposable, automate everything with code:
  - Can automate using scripts.
  - Can use declarative tools to define infrastructure.

Google Cloud

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.

Google Cloud

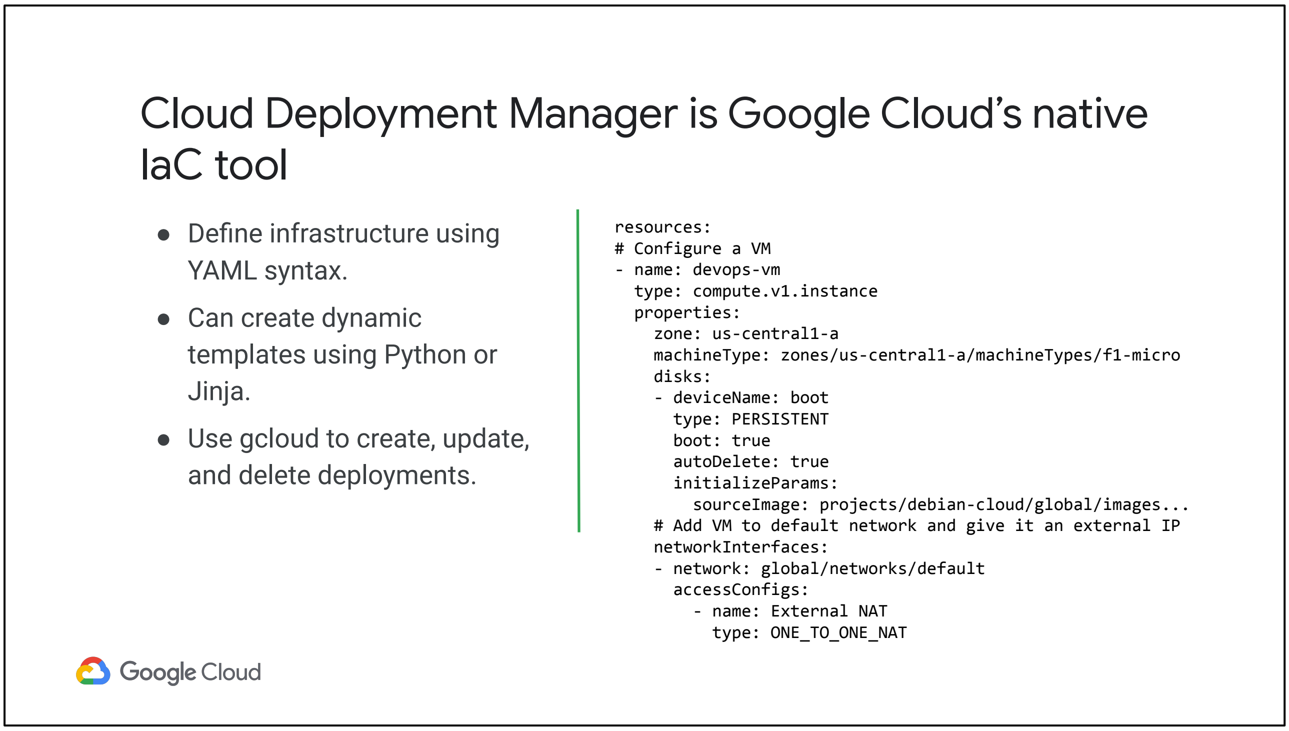
The on-demand provisioning of a deployment is extremely powerful. That this can be integrated into a CI pipeline smooths the path to continuous deployment.

Automated infrastructure provisioning means that the deployment complexity is managed in code and has the flexibility to change as requirements change. And all the changes are in one place.

Several tools can be used for IaC. Google Cloud provides Deployment Manager. Here deployments are described in a YAML file known as a configuration. This details all the resources that should be provisioned. Configurations can be modularized using templates which allow the abstraction of resources into reusable components across deployments.

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

- Terraform
- Chef
- Puppet
- Ansible
- Packer



Cloud Deployment Manager is Google Cloud's native IaC tool

- Define infrastructure using YAML syntax.
- Can create dynamic templates using Python or Jinja.
- Use gcloud to create, update, and delete deployments.

```yaml
resources:
# Configure a VM
- name: devops-vm
  type: compute.v1.instance
  properties:
    zone: us-central1-a
    machineType: zones/us-central1-a/machineTypes/f1-micro
    disks:
    - deviceName: boot
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: projects/debian-cloud/global/images...
    # Add VM to default network and give it an external IP
    networkInterfaces:
    - network: global/networks/default
      accessConfigs:
        - name: External NAT
          type: ONE_TO_ONE_NAT
```

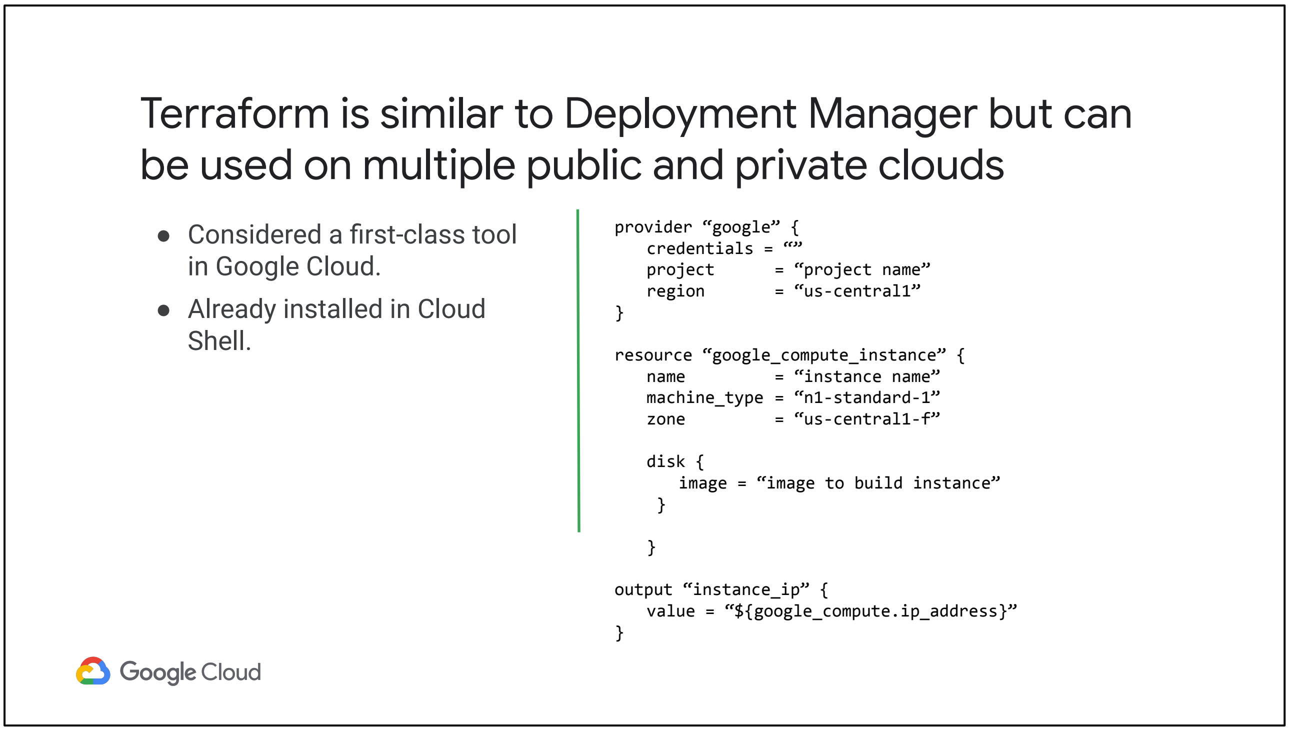
Google Cloud

The YAML snippet above shows a section from a configuration using Cloud Deployment Manager. The configuration must list resources. In the example provided, a virtual machine resource provision is described. You can see the details, including type of VM, zone, disks, and network interface.

Templates allow the separation of configuration into different pieces that can be used and reused across different deployments. These templates can be specific or more generalized. They can make use of template properties, environment variables, and modules to create dynamic configuration and template files. The template files can be written in Python or Jinja, the Python templating language.

Deployments can be created using the Google Cloud Console, APIs, or gcloud. For example, consider a deployment defined in a configuration file named *example.yaml*. To create the deployment, the command would be as follows, where the example deployment is the name of the deployment to be created:

```
gcloud deployment-manager deployments create example-deployment --config example.yaml
```



Terraform is similar to Deployment Manager but can be used on multiple public and private clouds

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

```
provider “google” {
    credentials = “”
    project      = “project name”
    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}”
}
```

Google Cloud

Terraform can be used across multiple cloud environments, both public and private. To seamlessly work with Terraform, the tools are installed in Cloud Shell.

The example configuration in the slide begins by indicating that the provider is Google Cloud. What follows is the configuration of a Compute Engine instance and associate disk.The output is there so that the IP address of the provisioned instance can be obtained.

To provision the deployment, the command *terraform apply* is executed. To display the IP address of the provisioned compute instance, the command *terraform output* can be used. This would display all output variables. For the IP address specifically, you could run *terraform output instance_ip*.

More details are here: https://www.terraform.io/docs/index.html

60 minutes

Lab

Building a DevOps Pipeline

Cloud Source Repositories

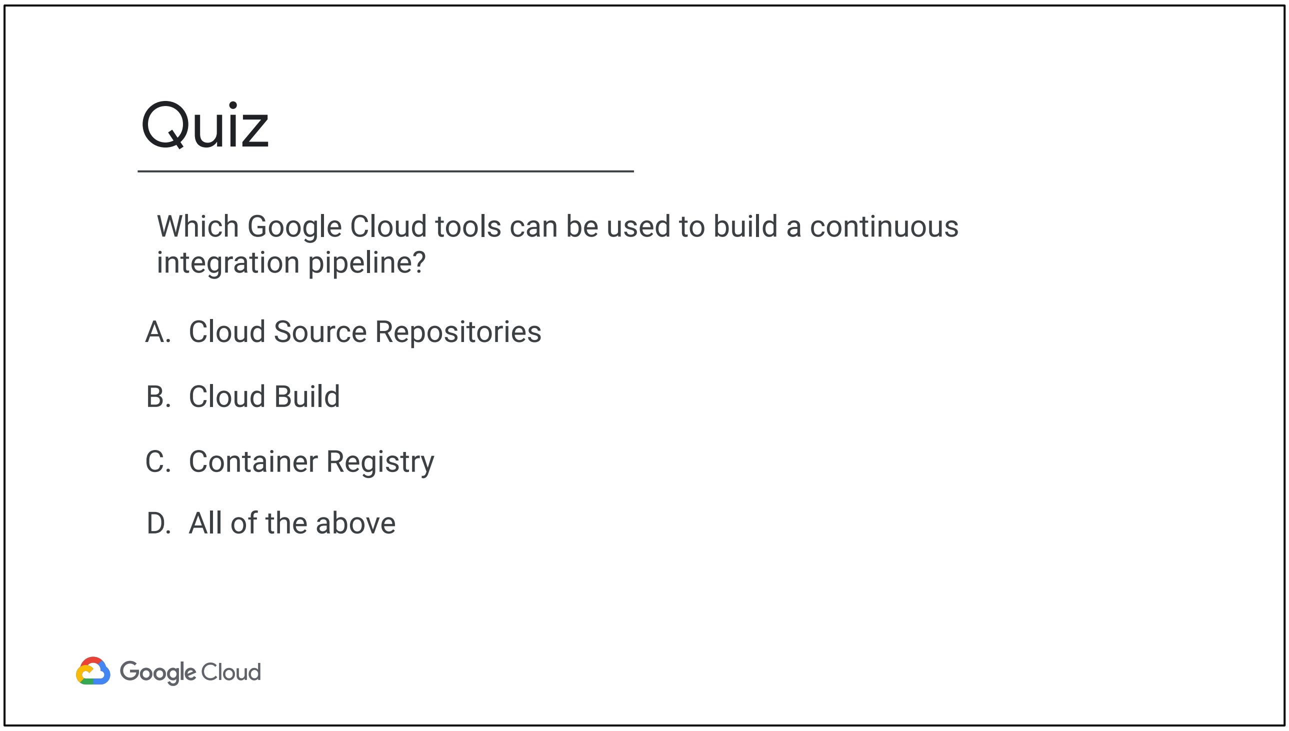
Cloud Build

Container Registry

Objectives

- Create a Git Repository
- Create a Simple Python Application
- Test Your Web Application in Cloud Shell
- Define a Docker Build
- Manage Docker Images with Cloud Build and Container Registry
- Automate Builds with Triggers
- Test Your Build Changes

Google Cloud



Quiz

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

A. Cloud Source Repositories

B. Cloud Build

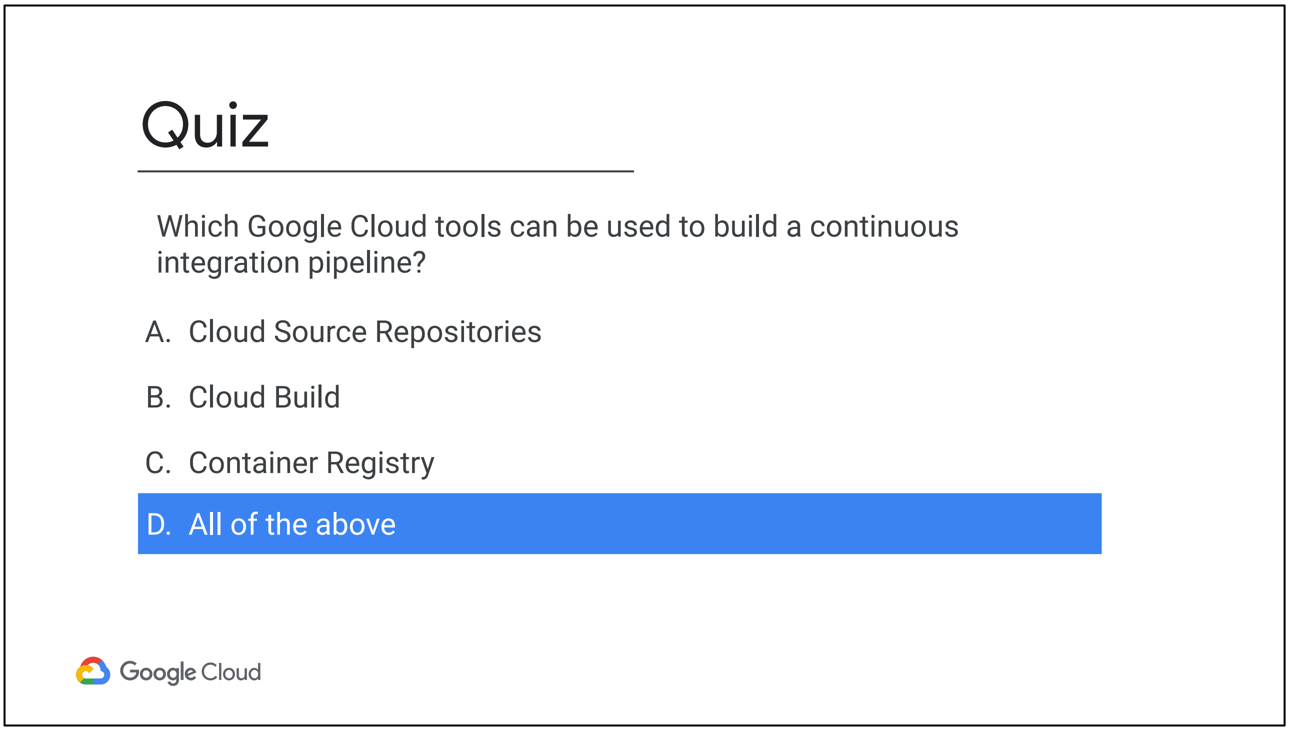
C. Container Registry

D. All of the above

Google Cloud

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

A. Cloud Source Repositories
B. Cloud Build
C. Container Registry
D. All of the above



Quiz

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

A. Cloud Source Repositories

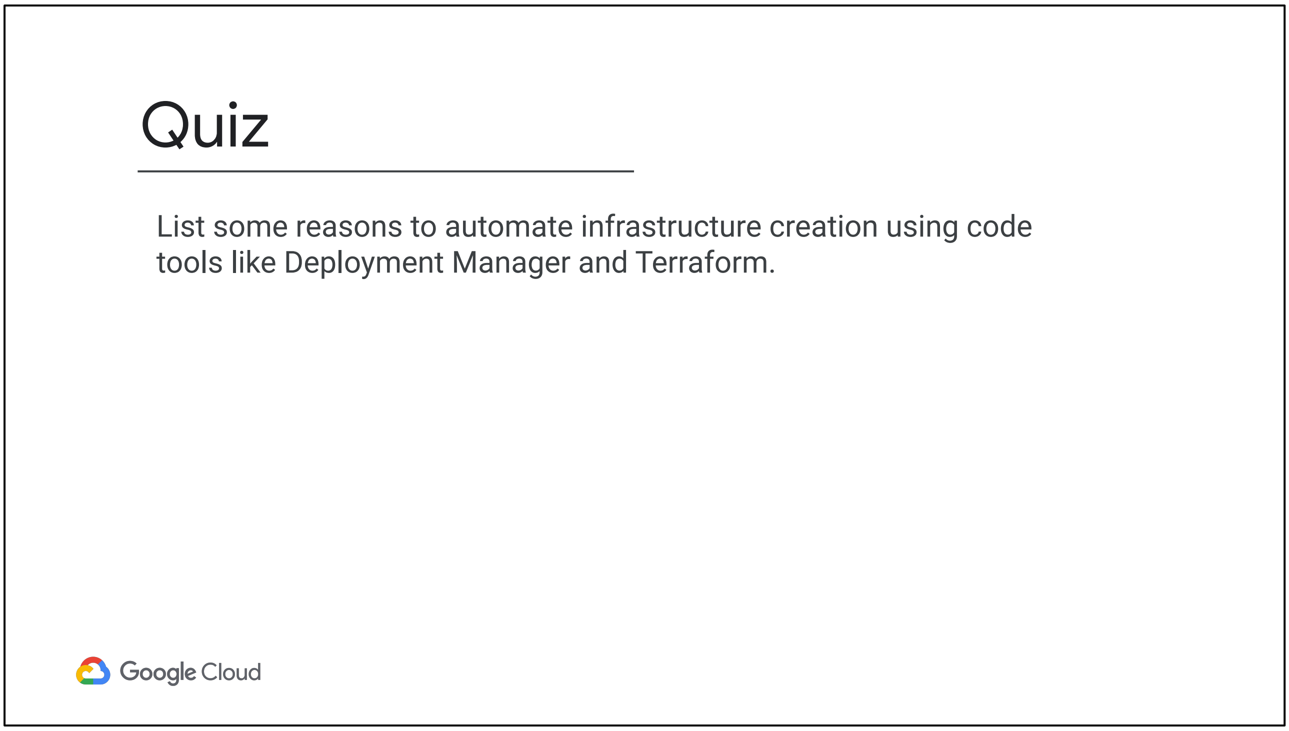
B. Cloud Build

C. Container Registry

[✓ D. All of the above]

Google Cloud

All the above answers are correct. Source Repositories provides a private Git repository, Cloud Build builds containers, and Container 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.

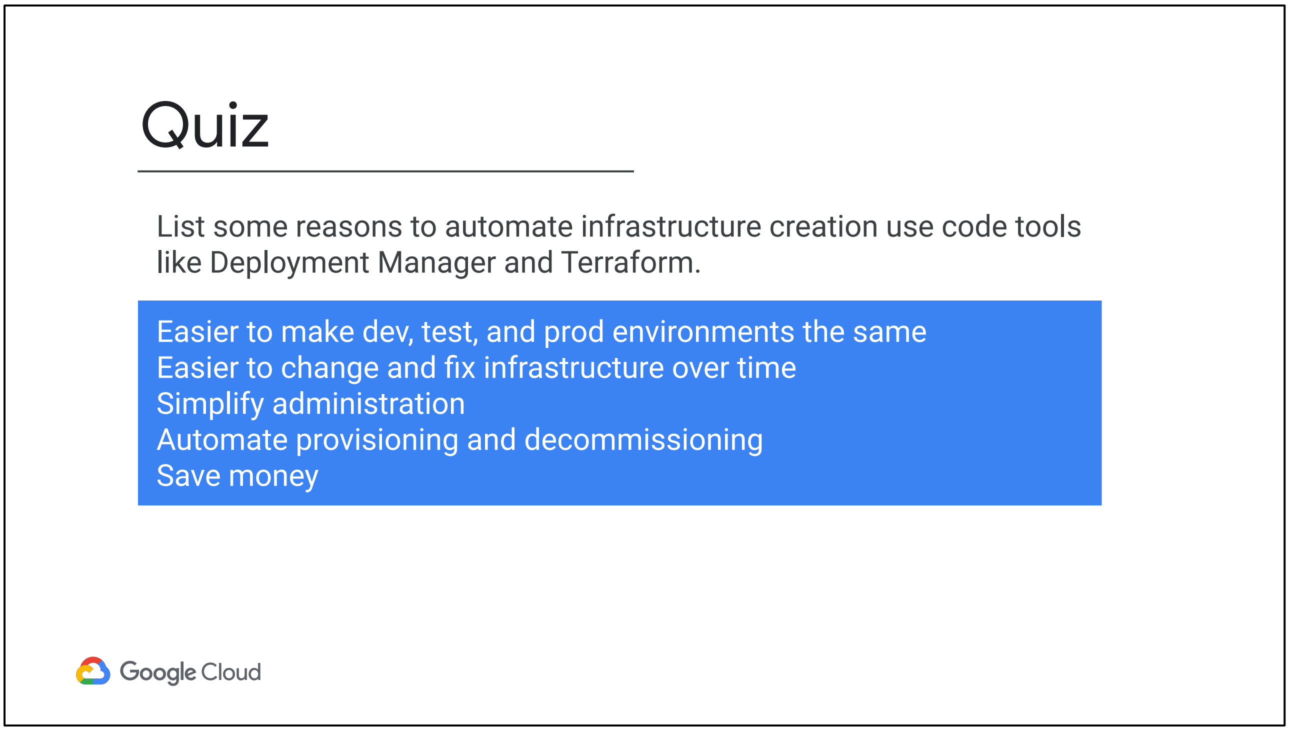


Quiz

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

Google Cloud

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



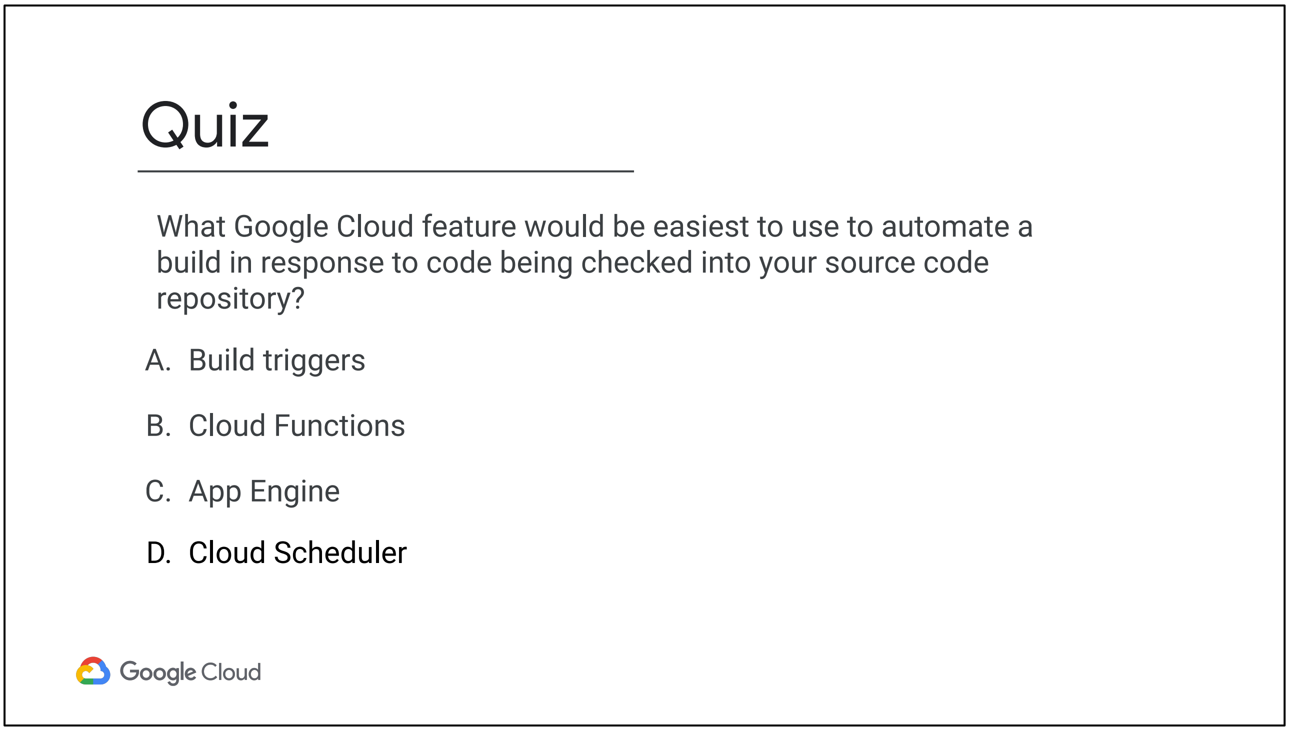
Quiz

List some reasons to automate infrastructure creation use code tools like Deployment Manager and 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

Google Cloud

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.



Quiz

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

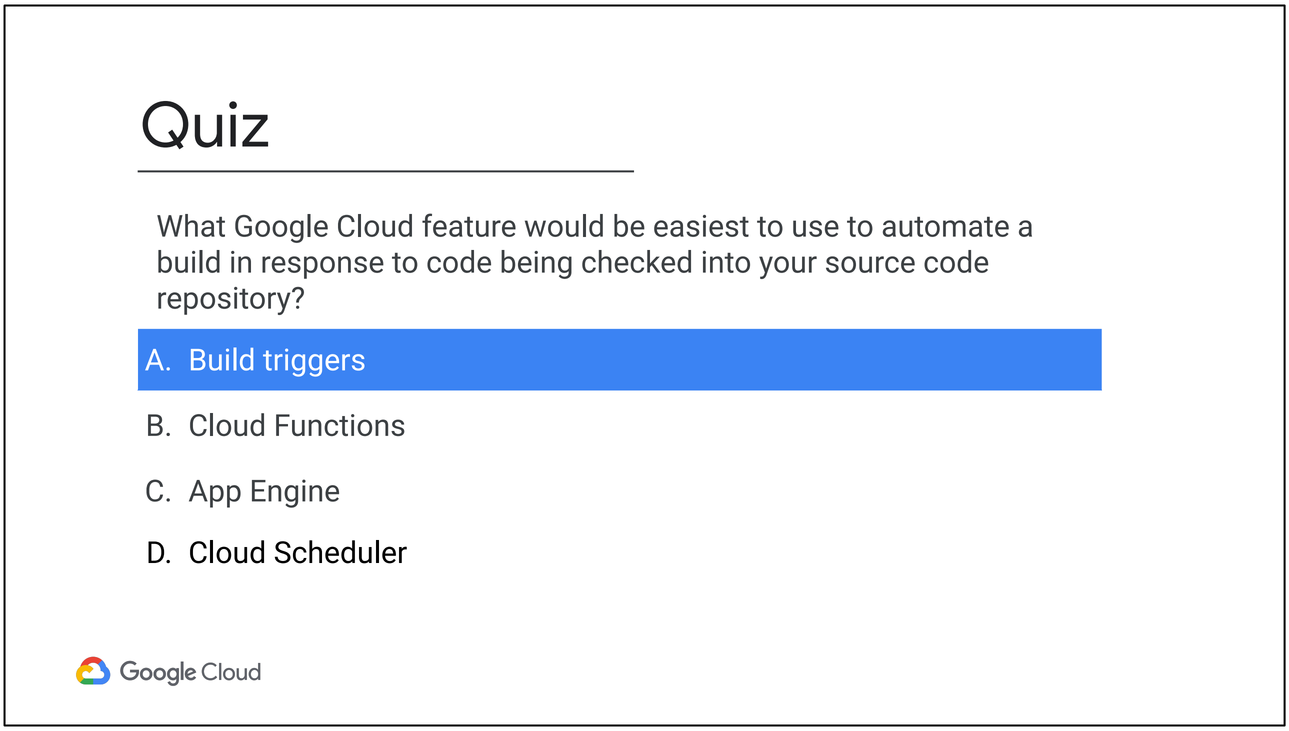
B. Cloud Functions

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 Functions
C. App Engine
D. Cloud Scheduler



Quiz

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 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 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.

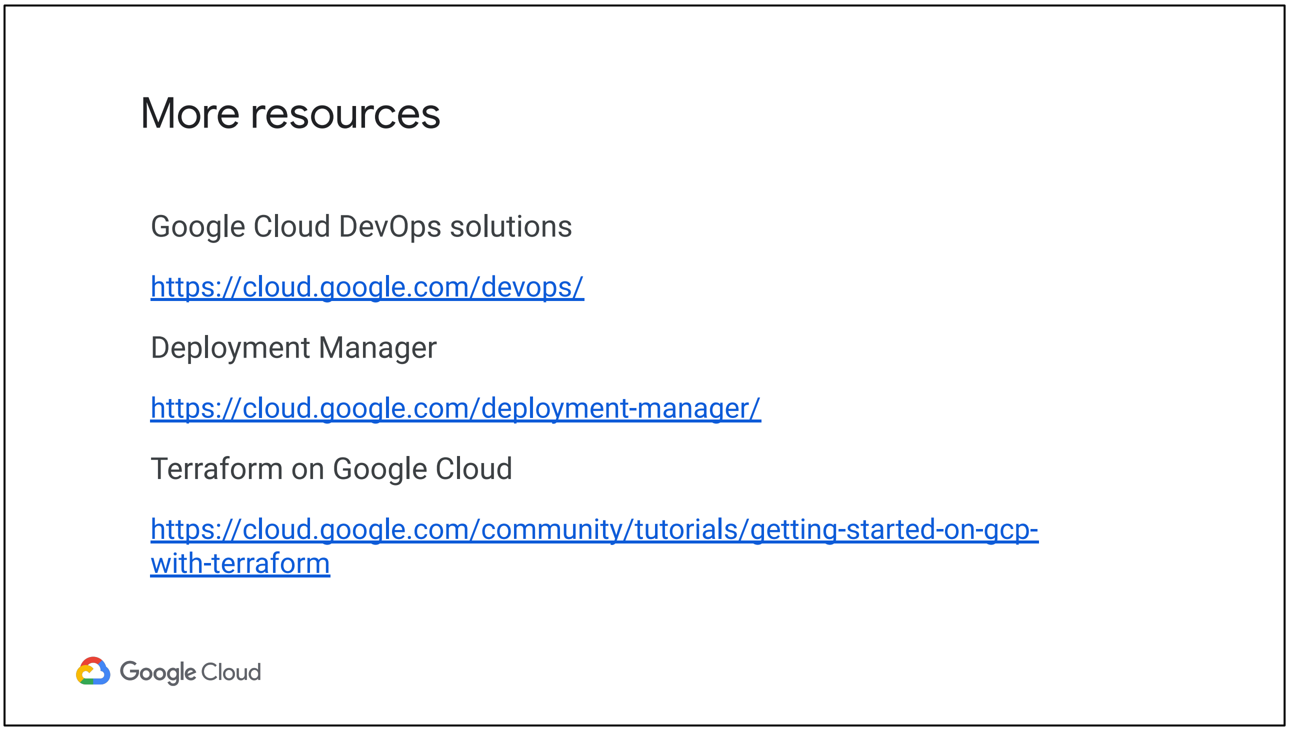
Review

DevOps Automation

Google Cloud

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 Container 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 Deployment Manager and Terraform.



More resources

Google Cloud DevOps solutions

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

Deployment Manager

https://cloud.google.com/deployment-manager/

Terraform on Google Cloud

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

Google Cloud

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

Google Cloud