Google Cloud

DevOps Automation

Priyanka Vergadia
Developer Advocate, Google Cloud

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.

Specifically, we will talk about 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 module introduces the Google Cloud tools for developing code and creating automated delivery pipelines that are provisioned on demand.

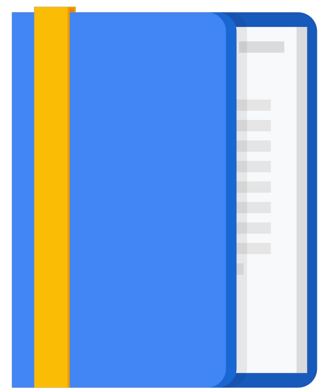
We will talk about using Cloud Source Repositories for source and version control, Cloud Build, including build triggers, for automating builds, and managing containers with Container Registry.

We'll finish by reviewing infrastructure as code tools like Deployment Manager and Terraform.

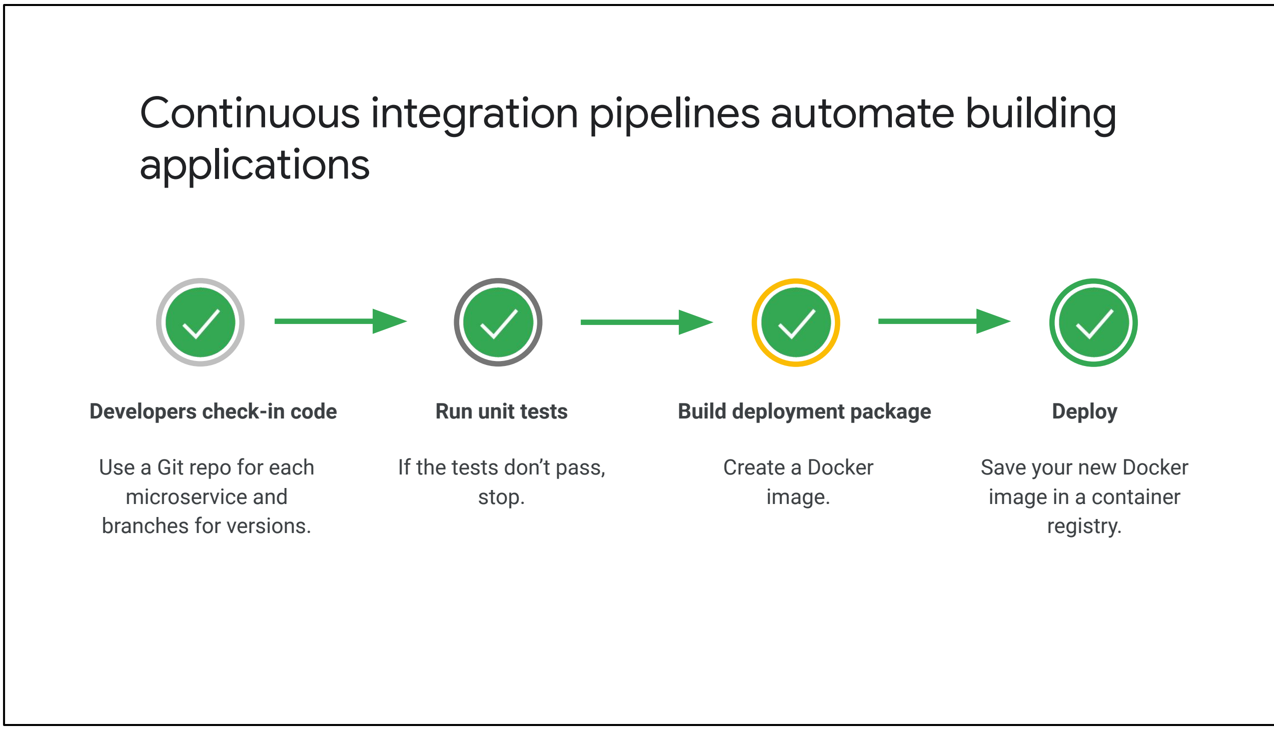
Agenda

Continuous Integration Pipelines

Infrastructure as Code



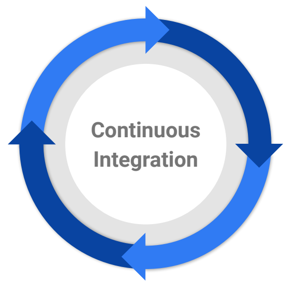
Let’s begin by talking about continuous integration pipelines.



Continuous integration pipelines automate building applications. This graphic shows a very simplistic view of a pipeline, which would be customized to meet your requirements. The process starts with checking code into a repository where all the unit tests are run. On successful passing of the tests, a deployment package is built as a Docker image. This image is then saved in a container registry from where it can be deployed. Each microservice should have its own repository.

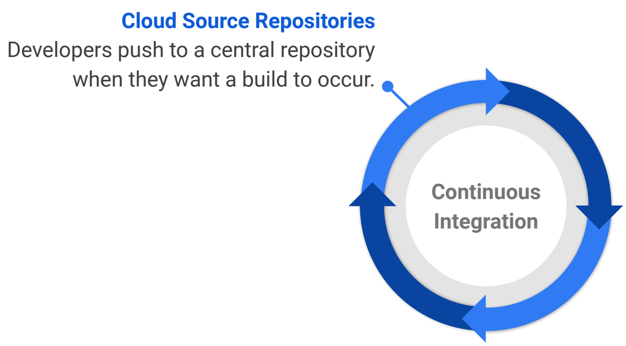
Typical extra steps include linting of code, quality analysis by tools such as SonarQube, integration tests, generating test reports, and image scanning.

Google provides the components required for a continuous integration pipeline



Google Cloud provides the components required to build a continuous integration pipeline. Let's go through each of those:

Google provides the components required for a continuous integration pipeline

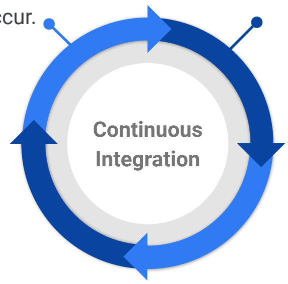


The **Cloud Source Repositories** service provides private Git repositories hosted on Google Cloud. These repositories let you develop and deploy an app or service in a space that provides collaboration and version control for your code. Cloud Source Repositories is integrated with Google Cloud, so it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

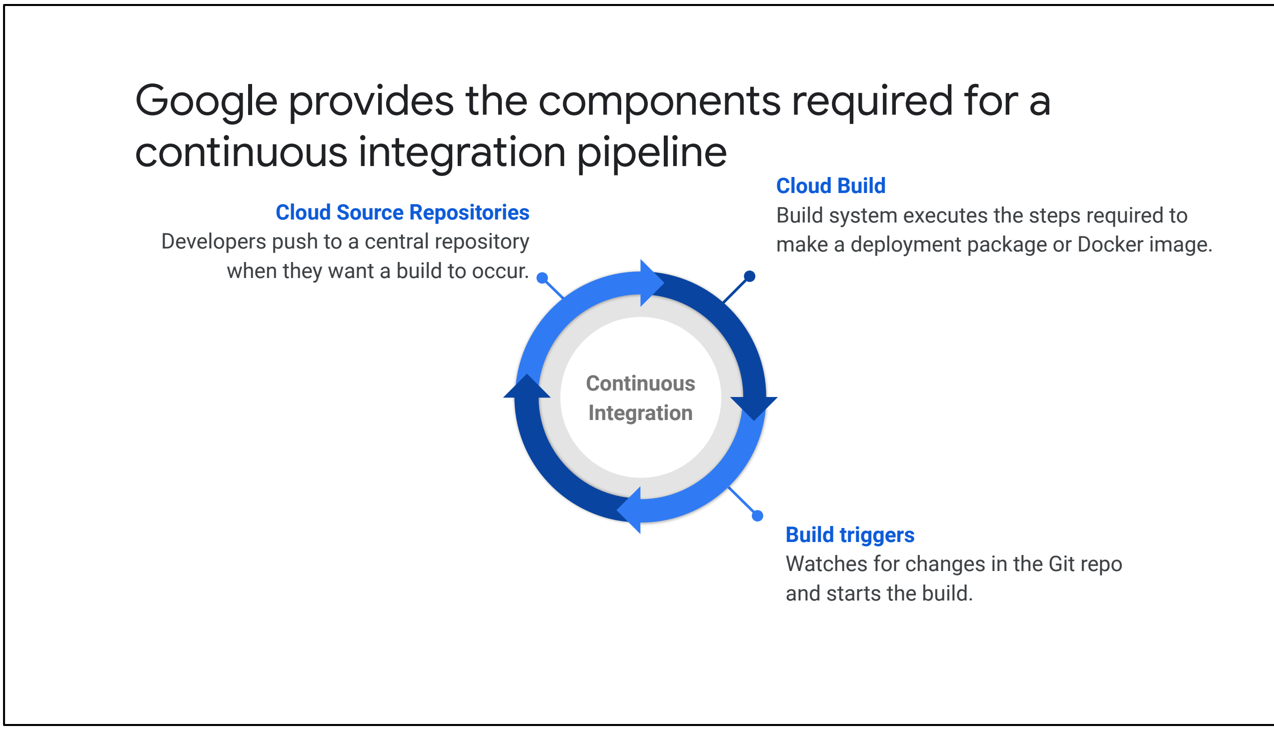
Google provides the components required for a continuous integration pipeline

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

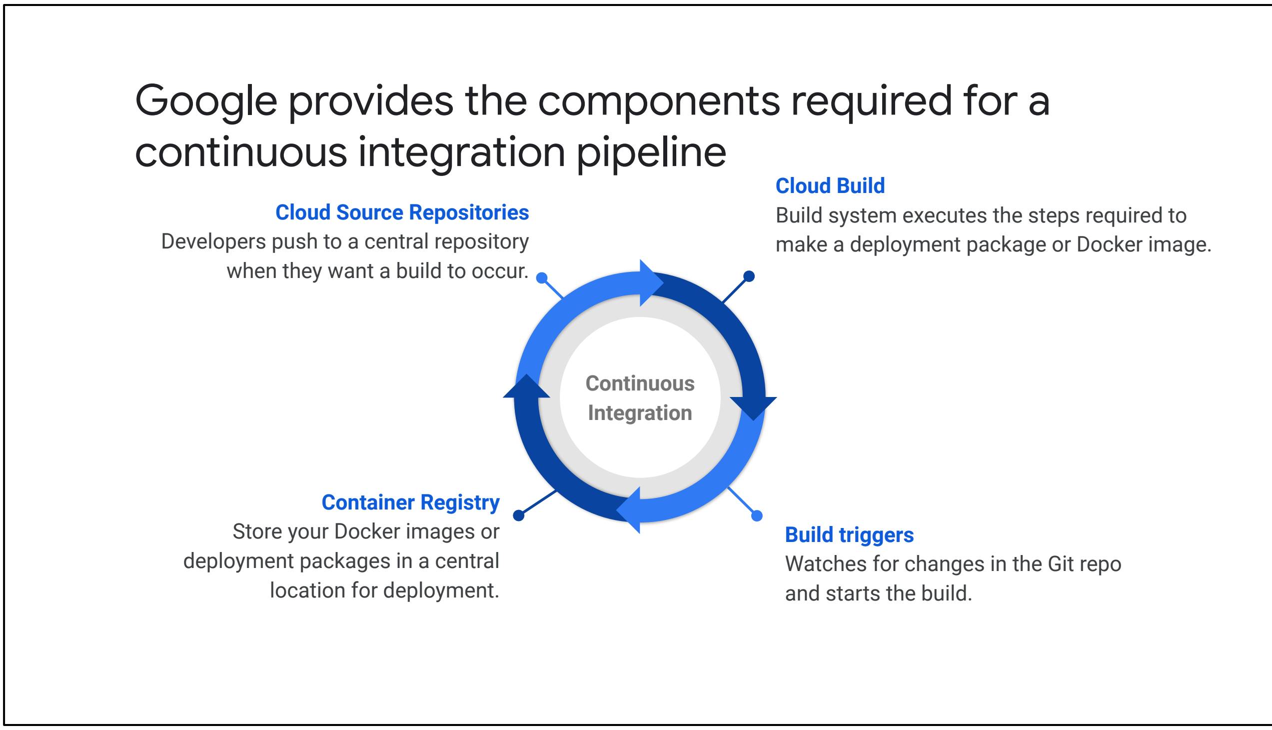
Cloud Build
Build system executes the steps required to make a deployment package or Docker image.



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.



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.

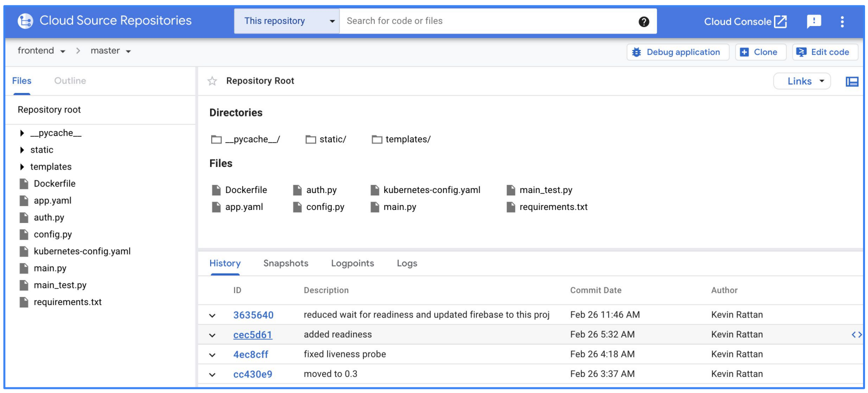


Container Registry is a single place for your team to manage Docker images or deployment packages, perform vulnerability analysis, and decide who can access what with fine-grained access control.

Let’s go through each of these services in more detail.

Cloud Source Repositories provides managed Git repositories

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



Cloud Source Repositories provides managed Git repositories.

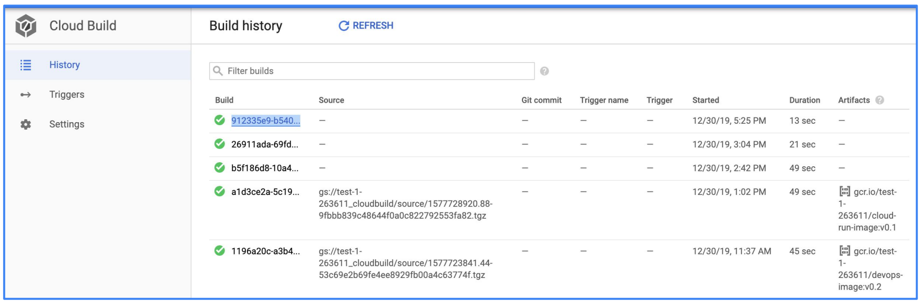
You can use Cloud IAM to add team members to your project and to grant them permissions to create, view, and update repositories.

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 .`



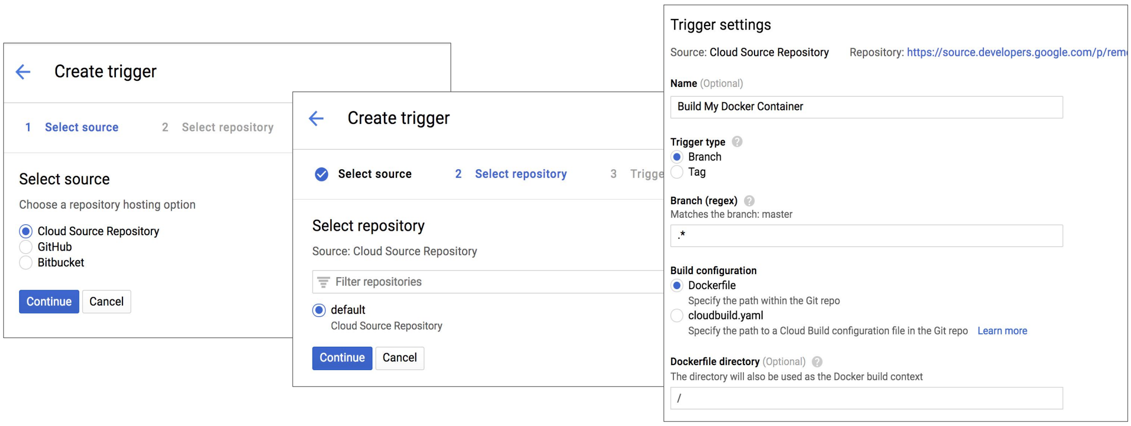
Cloud Build lets you build software quickly across all languages. It is a Google-hosted Docker build service and is an alternative to using the Docker build. The CLI can be used to submit a build using gcloud. An example is shown on this slide.

“gcloud build submits” submits the build and will run as a remote build. The “--tag” is the tag to use when the image is created. The tag must use the gcr.io or *.gcr.io.* namespace. Your source must contain a Dockerfile if you use the tag.

Finally, the . represents the location of the source to build.

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

Supports Maven, custom builds, and Docker



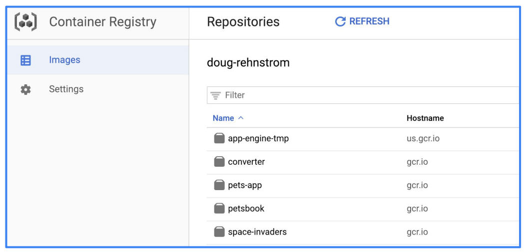
Build triggers watch a repository and build a container whenever code is pushed. Google's build triggers support Maven, custom builds, and Docker.

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 with the branch or tag value to match.

The build configuration can be specified either in a Dockerfile or a Cloud Build file. The configuration required is shown on this slide. First, a source is selected. This can be Cloud Source Repositories, Github, or Bitbucket. In the next stage, a source repository is selected, followed by the trigger settings. The trigger settings include information like the branch or tag to use for trigger, and the build configuration, for example the Dockerfile or Cloud Build file.

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`



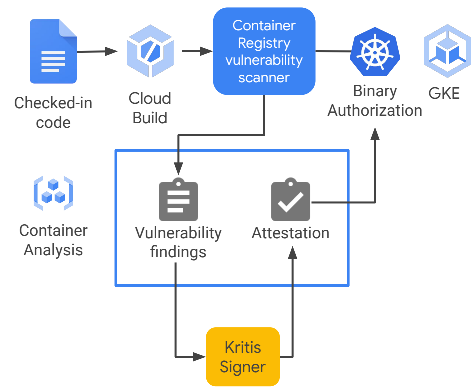
Container Registry is a Google Cloud-hosted Docker repository. Images built using Cloud Build are automatically saved in Container Registry. Images are tagged with a prefix, as shown on this slide.

It is also possible to push and pull images using the standard Docker commands. So to push an image, use:
docker push gcr.io/your-project-id/image-name

To pull an image use:
docker pull gcr.io/your-project-id/image-name

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

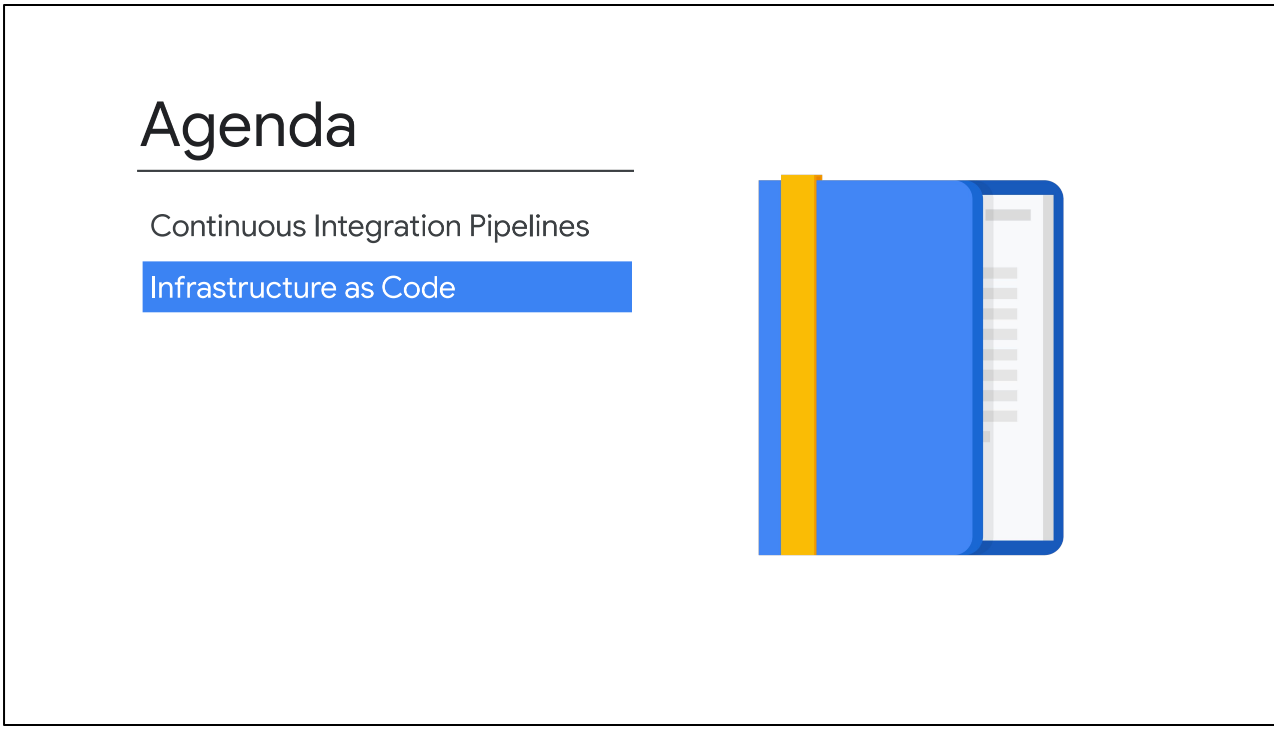
- Enable binary authorization on GKE cluster.
- Add a policy that requires signed images.
- When an image is built by Cloud Build an "attestor" verifies that it was from a trusted repository (Source Repositories, for example).
- Container Registry includes a vulnerability scanner that scans containers.



Now binary authorization allows you to enforce deployment of only trusted containers into GKE. Binary authorization is a Google Cloud service and is based on the Kritis specification. For this to work, you must enable binary authorization on your GKE cluster where your deployment will be made. A policy is required to sign the images. When an image is built by Cloud Build, an attestor verifies that it was from a trusted repository; for example, Source Repositories. Container Registry includes a vulnerability scanner that scans containers. A typical workflow is shown in the diagram.

Checkin of code triggers a Cloud Build. As part of the build, Container Registry will perform a vulnerability scan when a new image is uploaded. The scanner publishes messages to Pub/Sub. The Kritis Signer listens to Pub/Sub notifications from a container registry vulnerability scanner and makes an attestation if the image scanning 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.



Let’s now move on to consider infrastructure as code.

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.

Moving to the cloud requires a mindset change. The on-demand, pay-per-use model of cloud computing is a different model to traditional on-premises infrastructure provisioning. A typical on-premises model would be to buy machines and keep these running continuously. The compute infrastructure is typically built from fewer, larger machines. From an accounting view, the machines are capital expenditure that depreciates over time.

When using the cloud, resources are rented instead of purchased, and as a result we want to turn the machines off as soon as they are not required to save on costs. The approach is to typically have lots of smaller machines—scale out instead of scale up—and to expect and engineer for failure. From an accounting view, the machines are a monthly operating expense.

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.

In other words, in the cloud, all infrastructure needs to be disposable. The key to this is infrastructure as code (IaC), which allows for the provisioning, configuration, and deployment activities to 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. The automation can be achieved using scripts or declarative tools such as Terraform which we will discuss later.

It is really important that no time is spent trying to fix broken machines or installing patches or upgrades. These will lead to problems recreating the environments at a later date. If a machine requires maintenance, remove it and create a new one instead.

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.

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 it 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 provides Deployment Manager where 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
- And Packer

Let's take a closer look at Deployment Manager and Terraform.

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
```

Cloud Deployment Manager defines infrastructure using YAML files.

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.

The example YAML snippet to the right shows a section from a Deployment Manager configuration. The configuration must list resources, which in this example is a virtual machine, including the type of VM, zone, disks and network interface.

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}”
}
```

Terraform is similar to Deployment Manager but can be used on multiple public and private clouds. Terraform is already installed in Cloud Shell.

The example configuration file shown on the right begins by indicating that the provider is Google Cloud. What follows is the configuration of a Compute Engine instance and its disk. The output section allows for the IP address of the provisioned instance to be obtained from the deployment.

Lab

Building a DevOps Pipeline

In this first lab, you will build a DevOps pipeline using Cloud Source Repositories, Cloud Build, and Container Registry. Specifically, you will first create a Git repository. You will then write a simple Python application and add it to your repository. After that you will test your web application in Cloud Shell and then define a Docker build.

Once you define the build, you will use Cloud Build to create a Docker image and store the image in Container Registry. Then, you will see how to automate builds using triggers. Once you have the trigger, you will test it by making a change to your program and pushing that change to your Git repo.

Lab review

Building a DevOps Pipeline

In this lab, you learned how to use Google Cloud tools to create a simple and automated continuous integration pipeline. You used Cloud Source Repositories to create a Git repository and then used Cloud Build and triggers to automate the creation of your Docker images when code was checked into the repo.

When Cloud Build created your Docker images, it stored them in Container Registry. You saw how to access those images and test them in a Compute Engine VM.

You can stay for a lab walkthrough, but remember that Google Cloud's user interface can change, so your environment might look slightly different.

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 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.