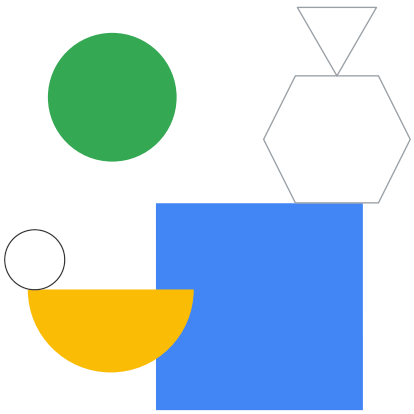
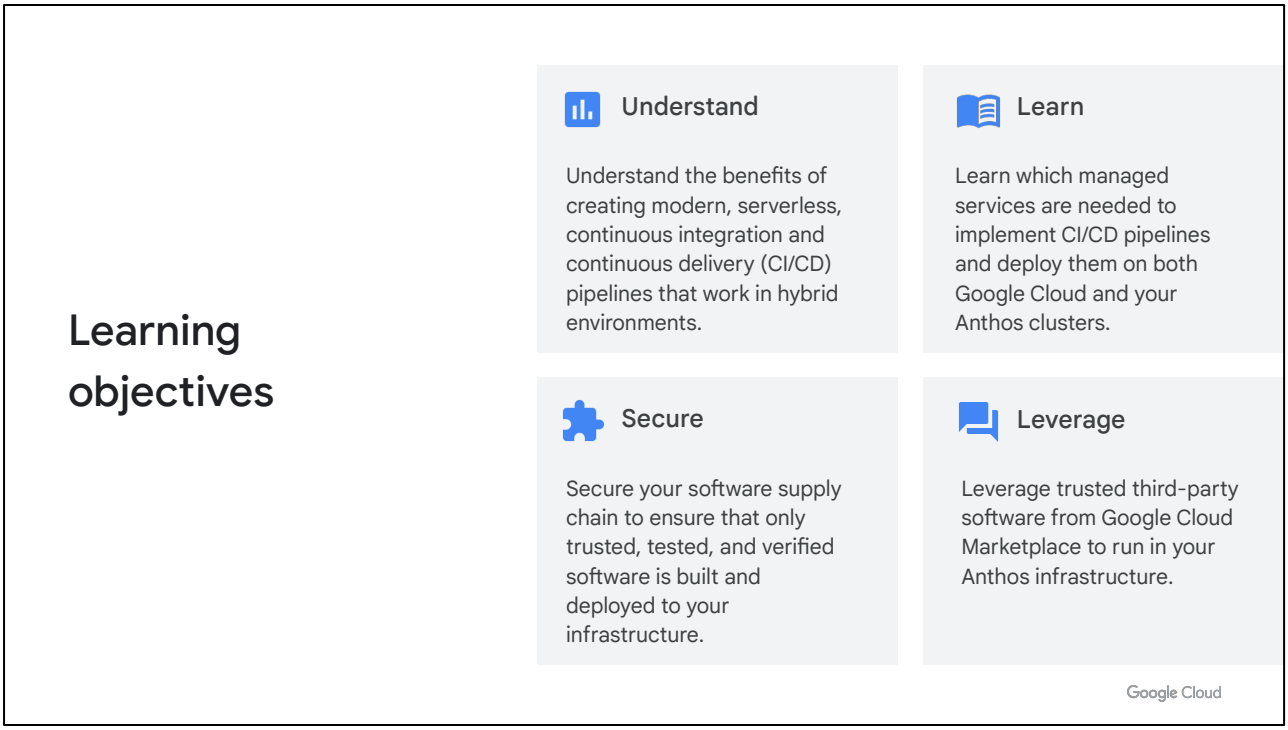
Google Cloud

Modern CI/CD for Anthos



Welcome to modern CI/CD for Anthos.

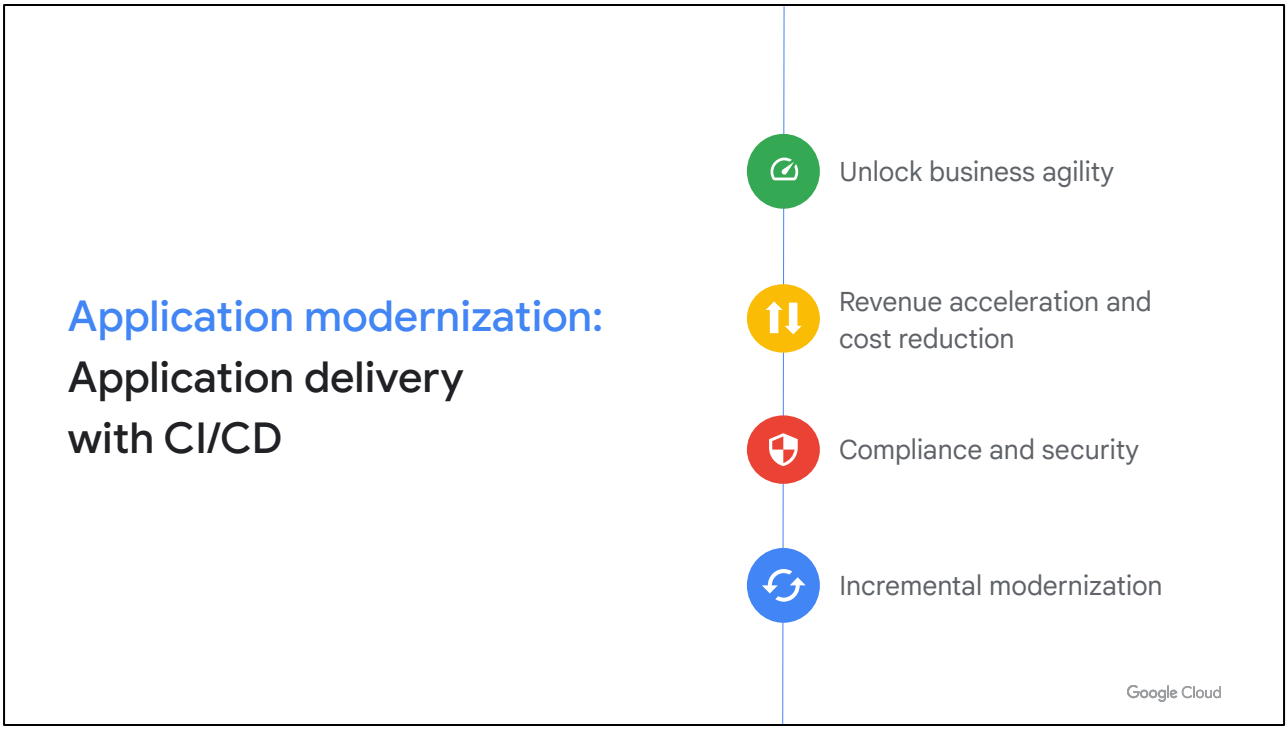


In this module, you:

- Understand the benefits of creating modern, serverless, continuous integration and continuous delivery (CI/CD) pipelines that work in hybrid environments.
- Learn the managed services needed to implement CI/CD pipelines and deploy them on both Google Cloud and your Anthos clusters.
- Secure your software supply chain to ensure that only trusted, tested, and verified software is built and deployed to your infrastructure.
- Leverage trusted third-party software from Google Cloud Marketplace to run in your Anthos infrastructure.

Here is the agenda for this module.

Here is the agenda for this module.



Application modernization is about:

1. Unlocking your companies' business agility via a modern application stack, infrastructure, and processes.
2. Fully managed, scalable infrastructure that helps you stay focused on core business.
3. Providing enterprise controls such as doing Kubernetes right, providing appropriate security, etc.

CI/CD is primarily about operationalizing those practices; providing developers with tools and automation to achieve application modernization outcomes repeatedly and at scale. We will see an incremental approach, with selective app modernization that fits with your existing DevOps tool chain.

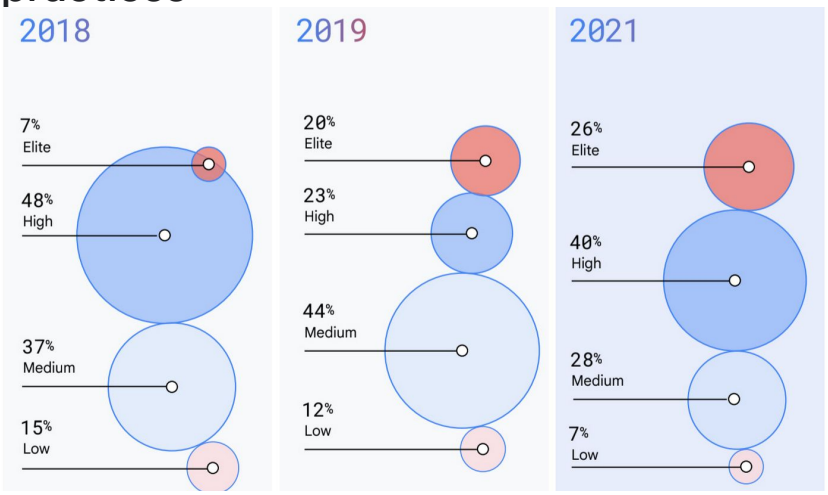
Accelerate State of DevOps reports metrics

Software delivery performance metric	Elite	High	Medium	Low
Development frequency	On-demand (multiple per day)	Weekly to monthly	Monthly to every 6 months	Less often than every 6 months
Lead time for changes	< 1 hour	1 day to 1 week	1 month to 6 months	6+ months
Time to restore service	< 1 hour	< 1 day	1 day < > 1 week	6+ months
Change failure rate	0-15%	16-30%	16-30%	16-30%

Google Cloud

Google Cloud's DevOps Research and Assessment (DORA) team creates an Accelerate State of DevOps report based on feedback from 32,000 professionals worldwide. They use four metrics to classify teams as elite, high, medium, or low performers based on their software delivery: deployment frequency, lead time for changes, mean-time-to-restore, and change fail rate. This year, we saw that elite performers continue to accelerate their pace of software delivery: they increased their lead time for changes from less than one day to less than one hour. Elite performers also deploy 973x more frequently than low performers, have a 6570x faster lead time to deploy, a 3x lower change failure rate, and an impressive 6570x faster time to recover from incidents when failure does happen.

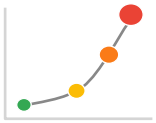
The report shows an acceleration of DevOps best practices



Google Cloud

The report shows an acceleration of DevOps best practices, with the highest performers continuing to raise the bar.

Challenges in the CI/CD journey



Scale

Scaling infrastructure costs
Environment variability
Visibility at scale
Onboarding velocity

Security

Separation of duties/concerns
Release promotion controls
Reliable rollback
Auditability and traceability

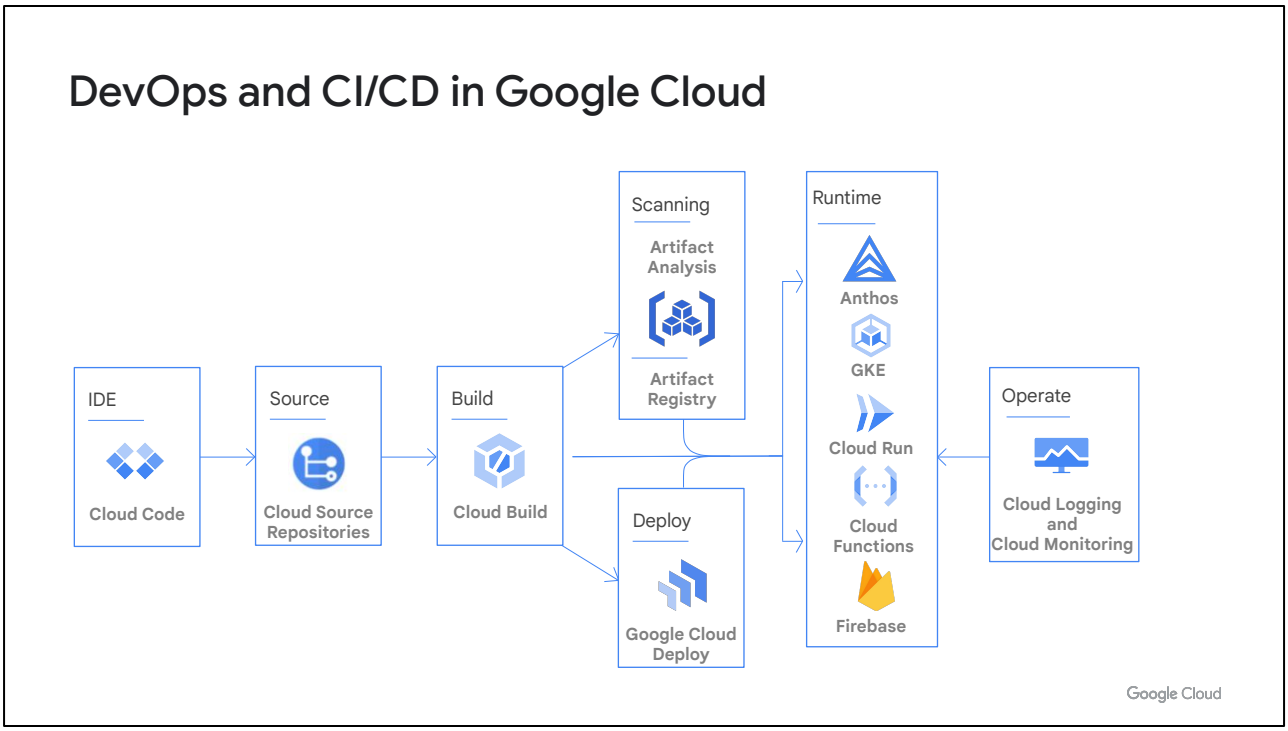
Integration

Integrate with your DevOps ecosystem
Observability integration

Google Cloud

Adopting DevOps and CI/CD best practices presents numerous challenges:

- Infrastructure costs: CI/CD sounds good but often comes with a high cost to create fixed replicas of production, staging, development, etc.
- Variations between target environments: No environment will be a perfect replica of production, so variations cause divergence and increased pipeline complexity.
- Visibility at scale: When you have dozens or hundreds of pipelines, how do you get an overall view of the health or activity of those pipelines?
- Developer onboarding: As your environment configuration becomes more complex, dev onboarding suffers.
- Separation of duties and concerns: A single CI/CD tool is problematic when you need to split access based on roles to conform to compliance (service now as dedicated “approval” interface).
- Observability integration: Involving Ops with logging and monitoring leads to increased coordination and dependencies, which in turn leads to higher mean time to recovery and more difficult root cause analysis.

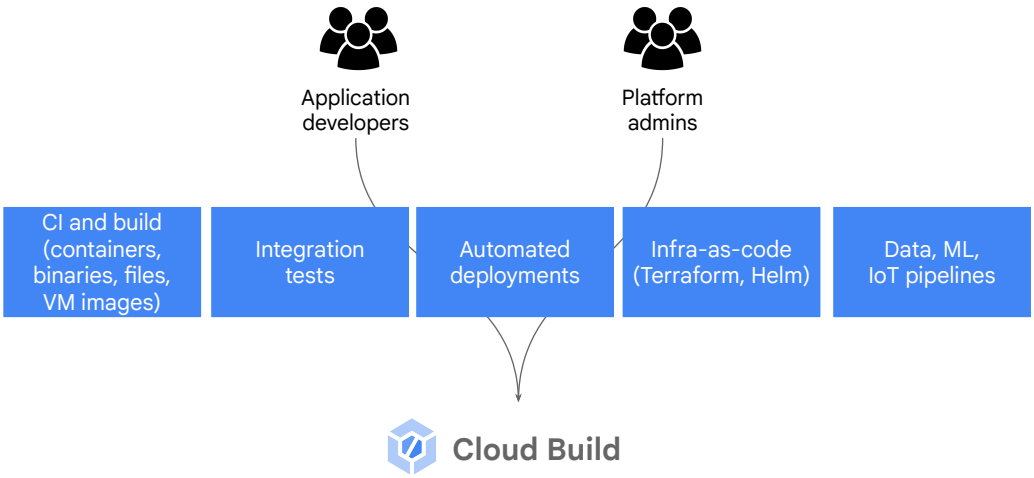


Google Cloud offers a complete, fully managed, serverless, DevOps pipeline with pluggable systems that can be replaced by any existing third-party tooling. Let's look at the individual components:

- Cloud Code is a plugin to IDEs like VS Code and IntelliJ that allows you to write, debug, and deploy your Kubernetes applications. It also has pre-configured samples so you can start running them in seconds.
- Cloud Source Repositories is a fully managed Git service that allows you to store your code and configure triggers to your CI/CD pipelines.
- Cloud Build is a completely serverless CICD platform. Build, test, and deploy applications in the cloud at any scale.
- Artifact Registry stores, manages, scans, and secures your container images and packages.
- Google Cloud Deploy is a declarative continuous delivery system.
- Finally, after your code is in production in a platform of your choice, you can use Cloud Logging and Cloud Monitoring for observing your applications.

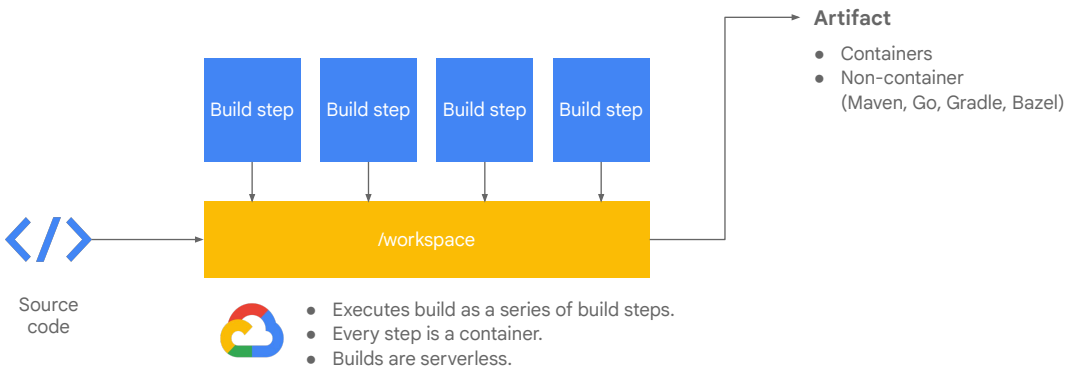
Let's look closer at the CI/CD components.

Cloud Build is an unopinionated, continuous automation platform



Cloud Build is an unopinionated continuous automation platform. You can set up continuous integration workflows by using triggers from Git repositories, storage buckets, and Pub/Sub notifications to start your artifact build process, which can create containers, binaries, files or VM images. Cloud Build is then also used for running unit and integration tests and deploying applications when all previous tasks are successful. Additionally, it is used as the runner for infrastructure as code solutions such as Terraform and Helm and can be used to orchestrate many types of services, from data to IoT pipelines.

Cloud Build workflow



Google Cloud

Cloud Build defines a workflow as a set of build steps that can occur sequentially or in parallel. Each step is executed by a container and occurs in a serverless, automated fashion. Workflows are usually triggered by a change in your code repository and produce an artifact such as a container or a binary.

Cloud builders

Containers with common languages and tools that you can use in build steps:

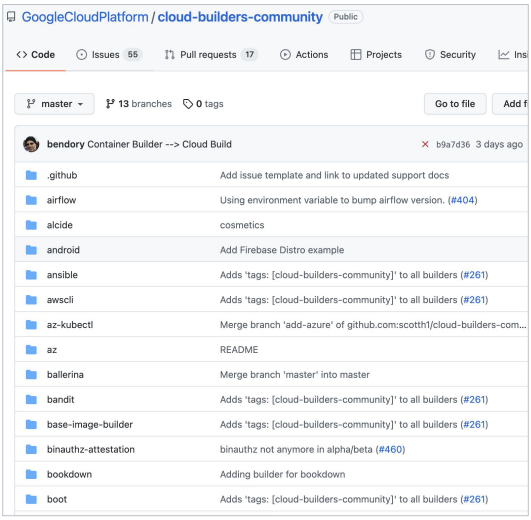
docker	gcr.io/cloud-builders/docker	docker example
go	gcr.io/cloud-builders/go	go example
gcloud	gcr.io/cloud-builders/gcloud	gcloud example
gradle	gcr.io/cloud-builders/gradle	gradle example
maven	gcr.io/cloud-builders/mvn	maven example
kubectl	gcr.io/cloud-builders/kubectl	kubectl example
npm	gcr.io/cloud-builders/npm	npm example

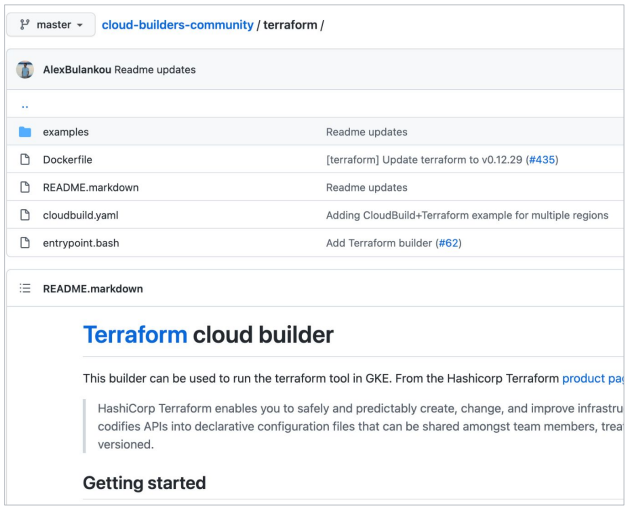
And many more: https://github.com/GoogleCloudPlatform/cloud-builders

Google Cloud

The containers that build, test, and deploy your application are pre-built and curated by Google and can be used without modification in your pipeline. Additionally, you can use any container as a build container in Cloud Build.

Use community-contributed builders in your project





Google Cloud

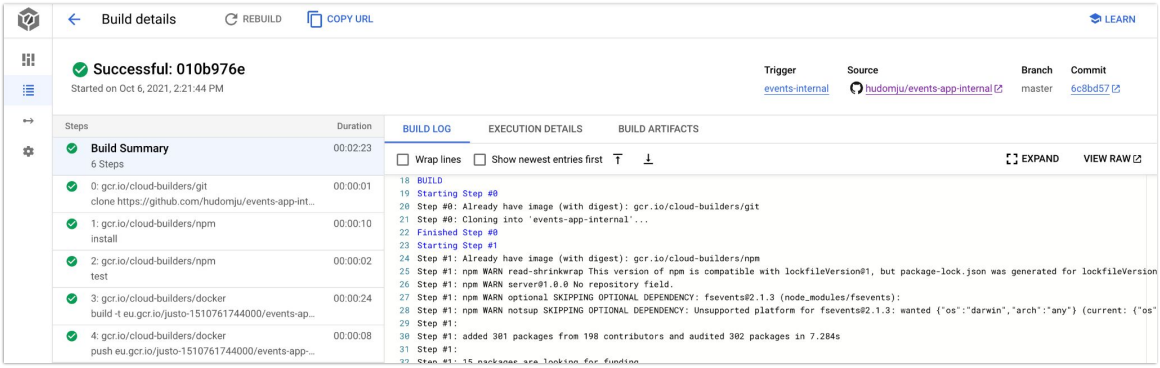
As the Google Cloud community of users built their own containers, Google created a curated repository where you can find community-contributed builders that you can use at no cost in your project.

cloudbuild.yaml file specifies the build steps

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  args: ['install']
- name: 'gcr.io/cloud-builders/npm'
  args: [test']
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/helloworld', '.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/$PROJECT_ID/helloworld']
```


The Cloud Build steps are defined in a configuration file, which takes the name of cloudbuild.yaml file by default. Here, you can see that we are using an NPM container builder to install and test the application and a Docker container builder to build the application container and push it to Container Registry.

Cloud Build runs these steps for you



When you submit your cloudbuild.yaml file to Cloud Build, the pipeline is created. Then you can execute it manually or via a trigger, and all the steps run serverlessly in the cloud for you.

Cloud Build pushes a container to Artifact Registry

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  args: ['install']
- name: 'gcr.io/cloud-builders/npm'
  args: [test']
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/helloworld', '.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/$PROJECT_ID/helloworld']
```

Artifact Registry securely stores and analyzes your containers.

Google Cloud

The last step of this continuous integration pipeline is pushing the container to Artifact Registry, which securely stores and analyzes your containers.

Use Cloud Build to deploy directly to your Anthos clusters via the Connect gateway

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  args: ['install']
- name: 'gcr.io/cloud-builders/npm'
  args: [test']
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/helloworld', '.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/$PROJECT_ID/helloworld']
- name: 'gcr.io/cloud-builders/gcloud'
  entrypoint: /bin/sh
  id: Deploy to Anthos cluster on VMware
  args:
  - '-c'
  - |
    set -x && \
    export KUBECONFIG="$(pwd)/gateway-kubeconfig" && \
    gcloud container hub memberships get-credentials my-vmware-cluster && \
    kubectl --kubeconfig gateway-kubeconfig apply -f myapp.yaml
```

Google Cloud

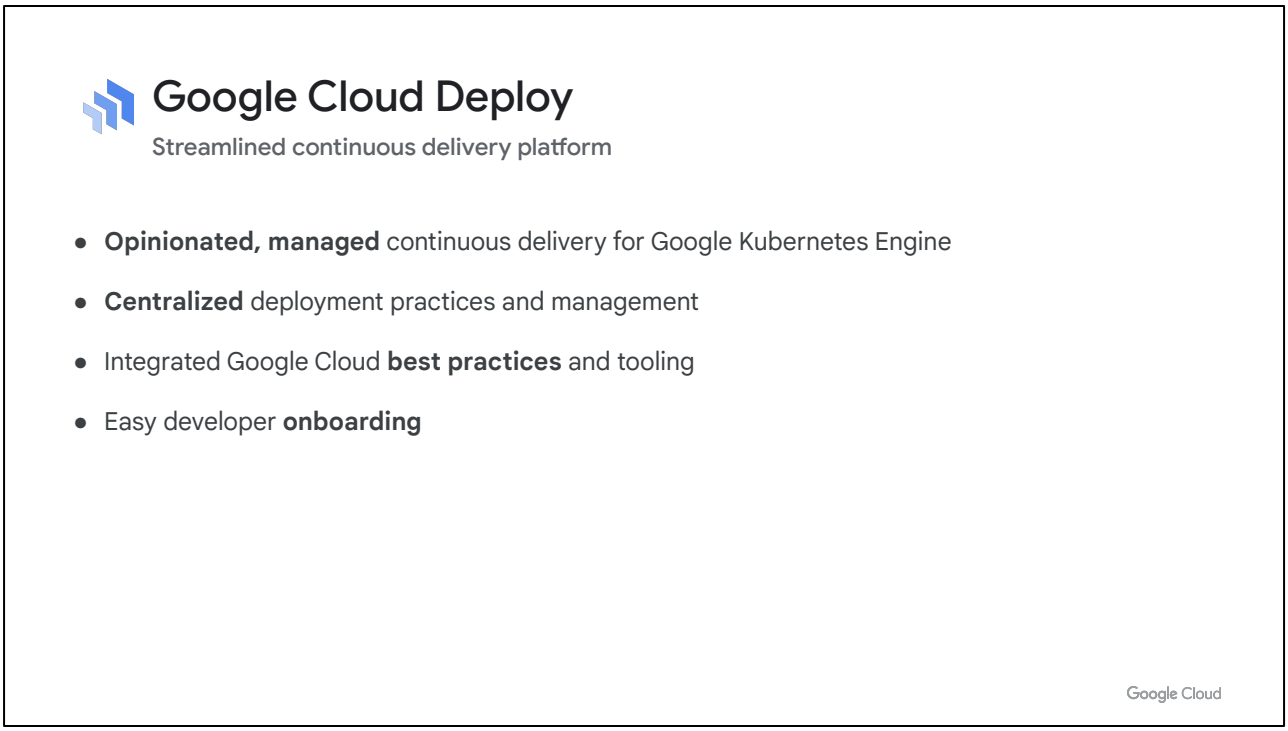
You can use Cloud Build to deploy directly to your target, thus providing a simple continuous deployment workflow. In this example, we execute the CI/CD pipeline in Google Cloud and deploy to an Anthos cluster on VMware that is located on-premises. This process relies on the Connect gateway, which acts as a proxy in Google Cloud and uses Anthos Connect to communicate with the Anthos cluster in the network on-premises.

Or trigger a continuous deployment pipeline with Google Cloud Deploy

```
steps:
- name: 'gcr.io/cloud-builders/npm'
  args: ['install']
- name: 'gcr.io/cloud-builders/npm'
  args: [test']
- name: 'gcr.io/cloud-builders/docker'
  args: ['build', '-t', 'gcr.io/$PROJECT_ID/helloworld', '.']
- name: 'gcr.io/cloud-builders/docker'
  args: ['push', 'gcr.io/$PROJECT_ID/helloworld']
- name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: gcloud
  args: [
        "beta", "deploy", "releases", "create", "rel-\${SHORT_SHA}",
        "--delivery-pipeline", "helloworld-pipeline",
        "--region", "us-central1",
        "--annotations", "commitId=\${REVISION_ID}",
        "--images", "helloworld=gcr.io/$PROJECT_ID/helloworld"
        ]
```

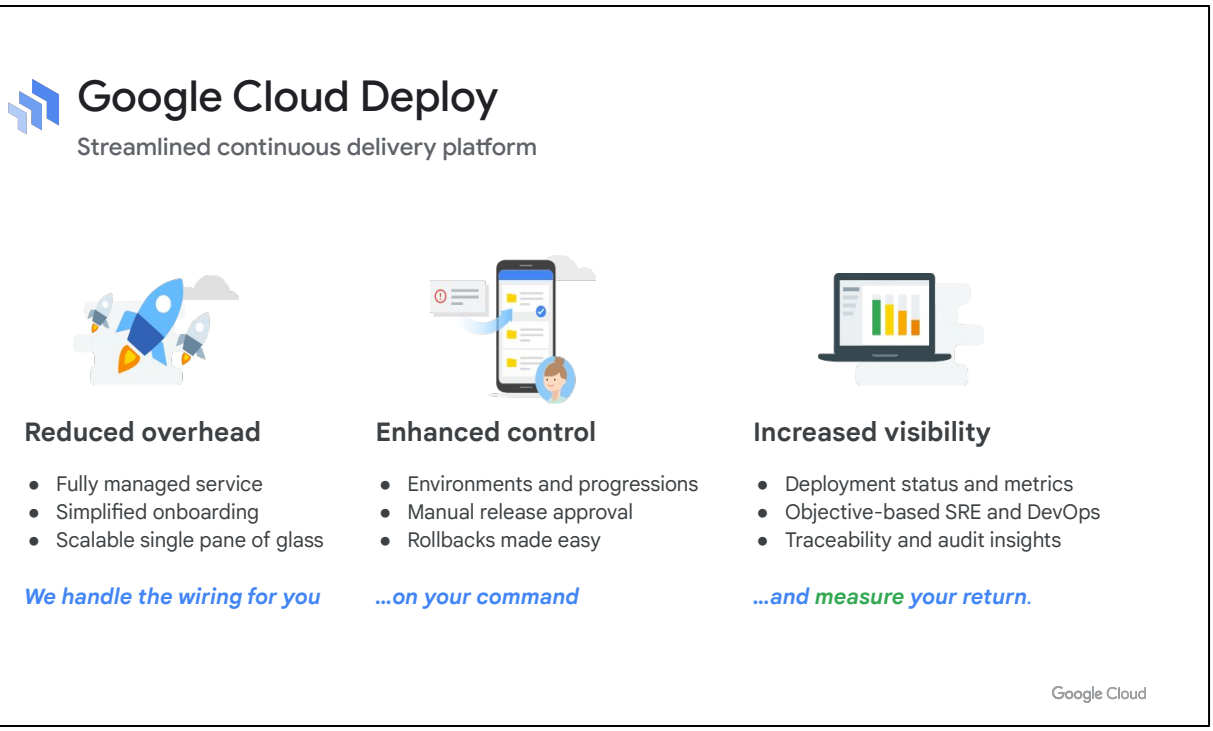
Google Cloud

If you want to use a more advanced continuous deployment pipeline, you can trigger a Google Cloud Deploy release instead.



Google Cloud Deploy is a streamlined continuous delivery platform that is:

- Fully managed, so that you can stay focused on value creation.
- Opinionated in the form of best practices and tooling.
- Consistent with a single interface across delivery pipelines.
- Fully integrated with Google Cloud best practices and tools like logging and Identity and Access Management.
- Easy for developers to adopt.



Google Cloud Deploy characteristics include:

Reduced overhead

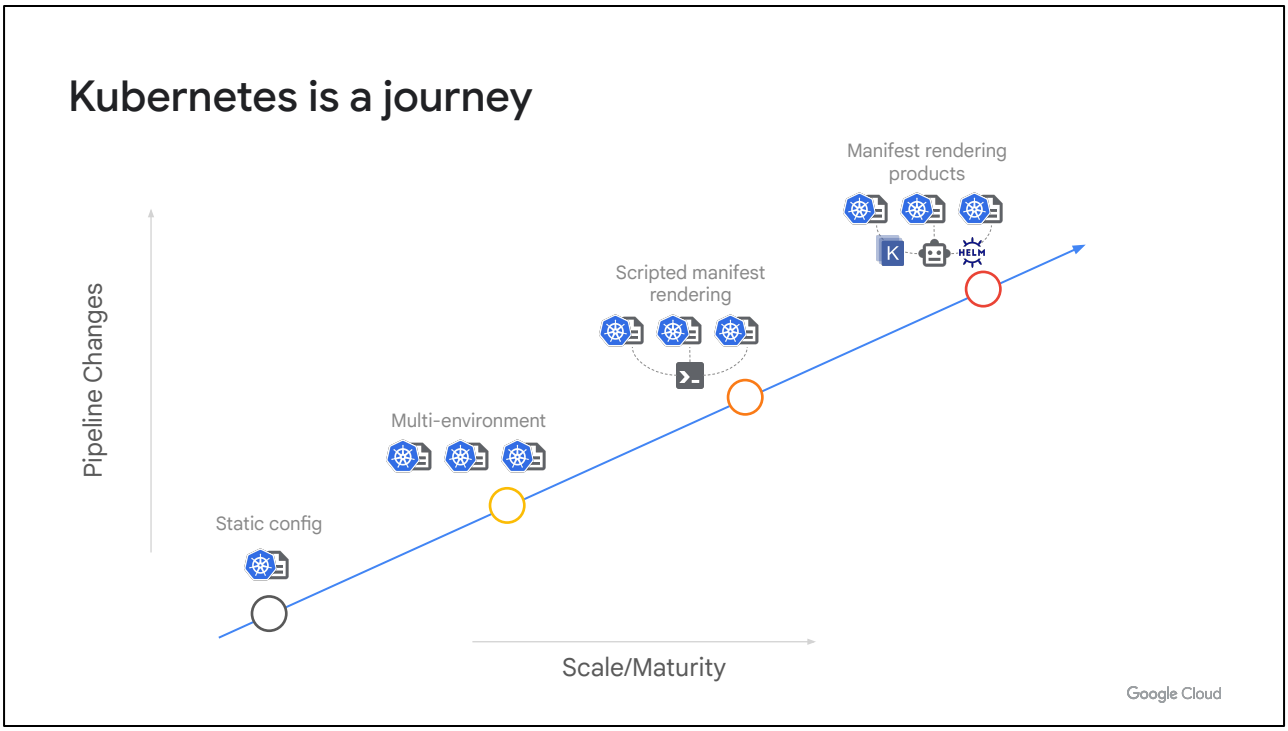
- No servers to manage or patch
- Simplified onboarding: just point to the source repository
- Scalable design, which scales for many apps across multiple environments

Enhanced control

- Provide consistent lifecycle progressions.
- Implement manual release approvals.
- Enable painless, error-free rollbacks.

Increased visibility

- Gain insights on deployment metrics.
- Drive objective-based SRE and DevOps processes.
- Provide traceability and audit insights.



When talking to customers on their journey to deploying to Kubernetes both in the cloud and on-premises with Anthos, Google realized that there are different maturity stages.

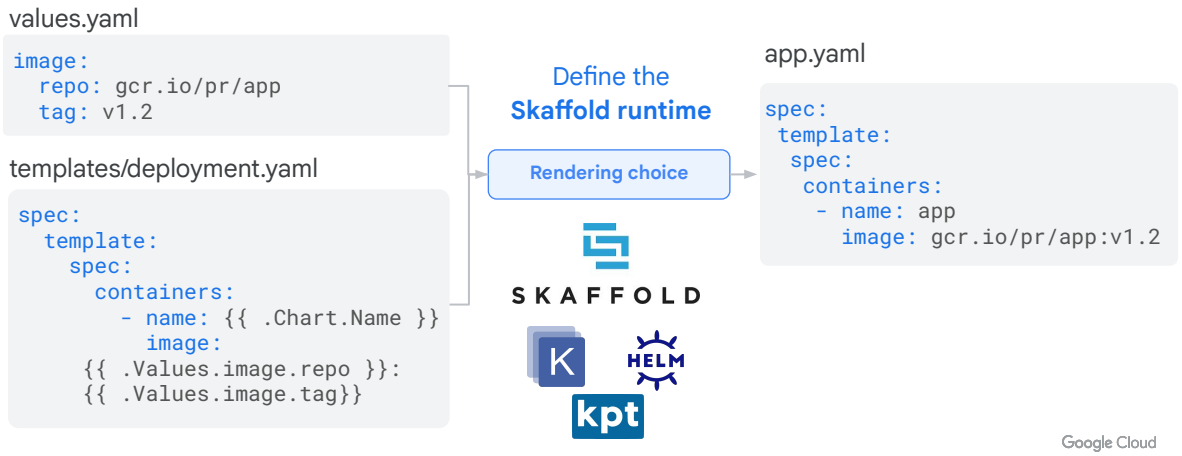
- Everyone starts out with a static configuration. At some point, diversity of environments leads to config per environment, which turns into an **inflexible responsibility model**. As this iteration happens, tight coupling in config and ownership leads to dependencies between the Dev and Ops platforms in terms of permissions and responsibilities.

- Companies then start to provide custom scripting to render manifests per environment, which creates brittle deployment pipelines.

- As Kubernetes scale and maturity advances within the organization, they discover and adopt Kubernetes-native manifest rendering tools such as:
  - Helm
  - Kustomize
  - Kpt

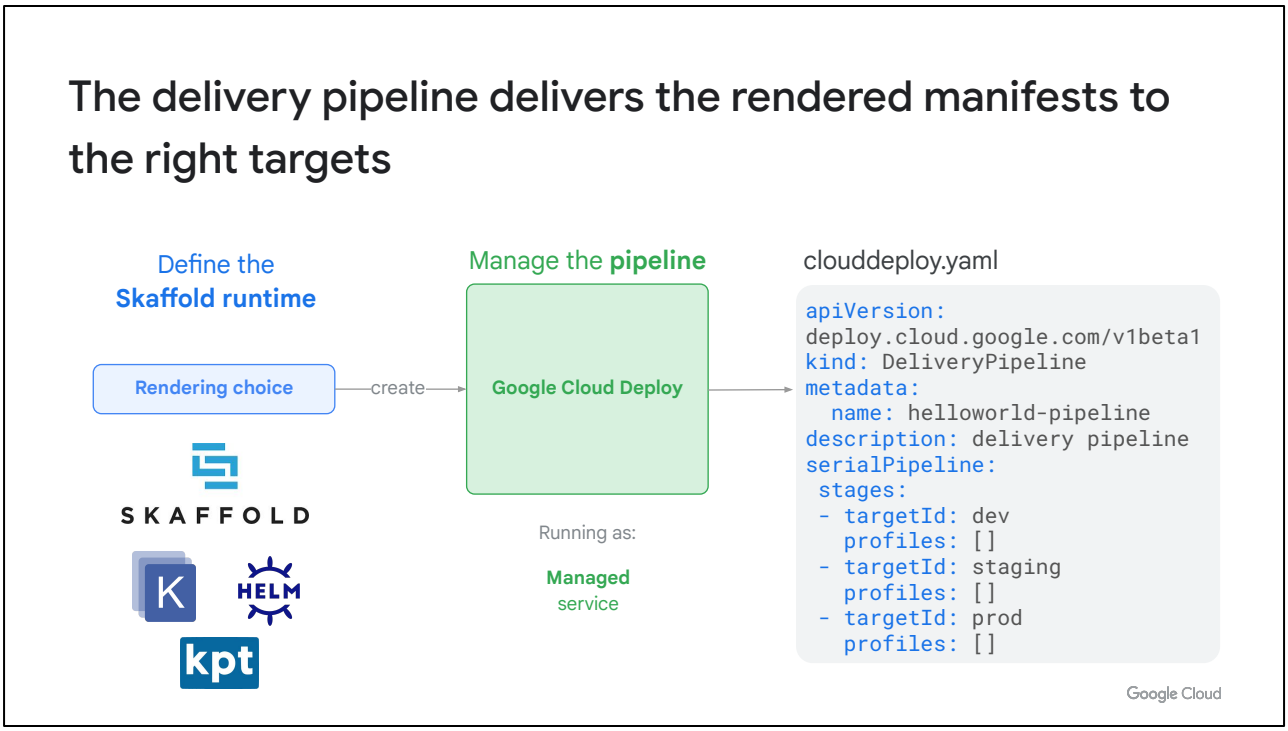
- However, the space moves quickly, and new technology and tools are constantly evolving.

Google Cloud Deploy uses Skaffold to render Kubernetes-native applications



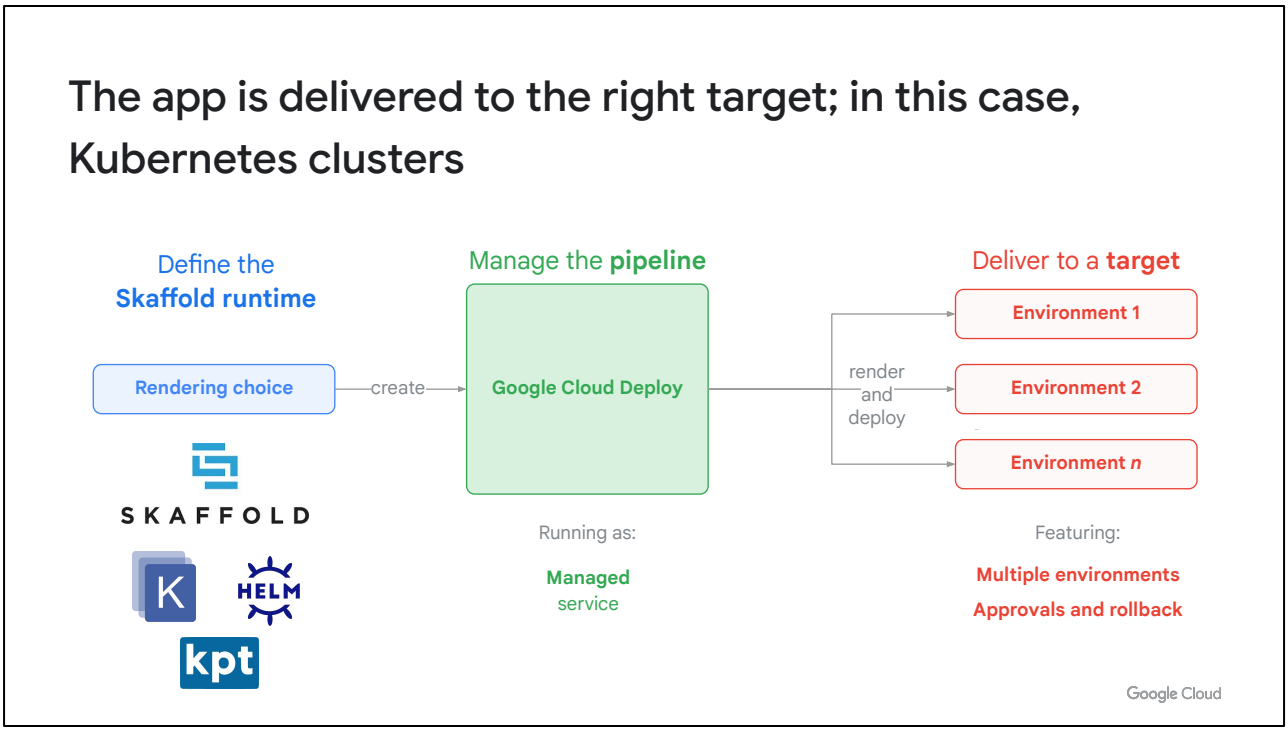
Google Cloud Deploy uses Skaffold to separate rendering tools from the delivery pipeline, which allows you to use a rendering tool of your choice. Skaffold configuration defines which manifests are rendered and how. This makes manifest rendering flexible without affecting how you define your delivery pipeline.

The most common tools for packaging, customizing, validating, and applying Kubernetes resources are kpt (pronounced “kept”), Kustomize, and Helm, which Skaffold leverages to render the Kubernetes manifests. The example on this slide shows a Helm package with a values file and a deployment.yaml with placeholders to be replaced with values. Depending on the environment that we deploy to, we would use a different set of values. For example, the configuration for production might be different from the one for development. Scaffold uses profiles to match the configuration with the deployment target so that a development Kubernetes cluster receives the templates rendered with the development values.

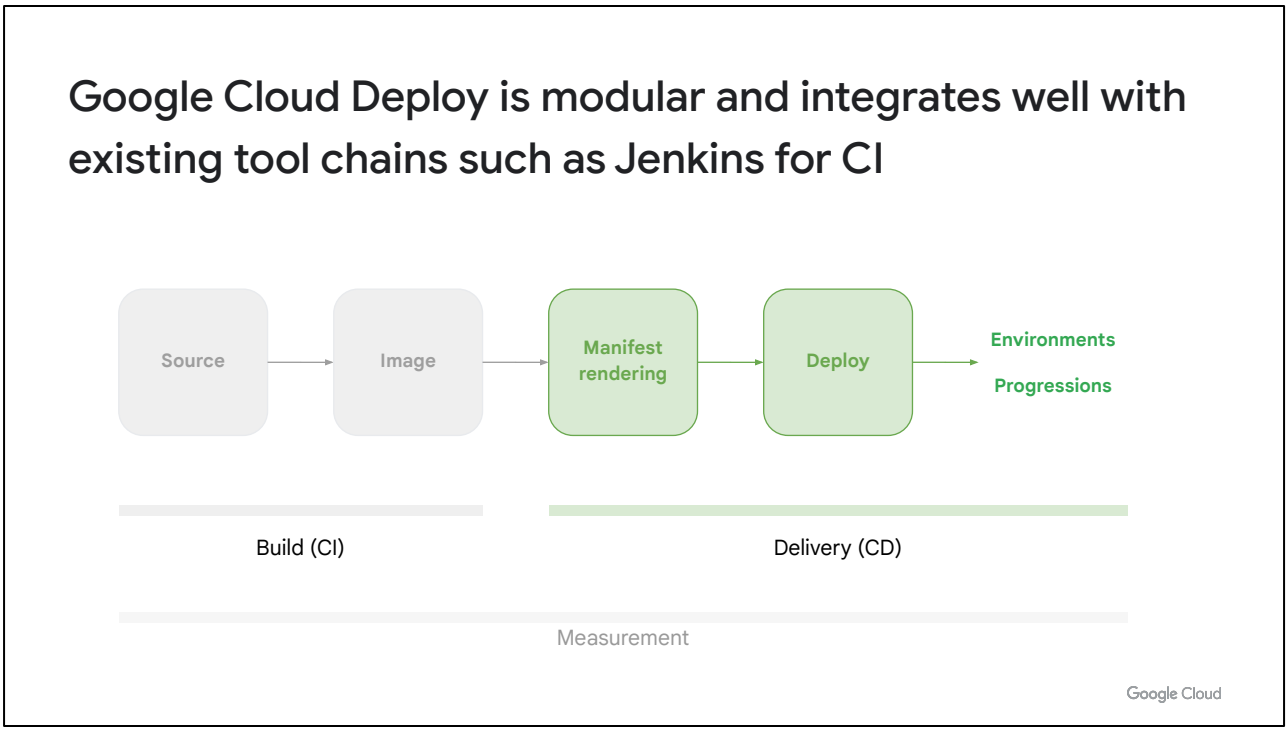


The delivery pipeline delivers the rendered manifests to the right targets.

In Google Cloud Deploy, the delivery pipeline is a representation of the workflow that delivers an application to each target in a deployment progression; that is, from dev to staging to production. The delivery pipeline is defined in a YAML configuration file, typically clouddeploy.yaml.

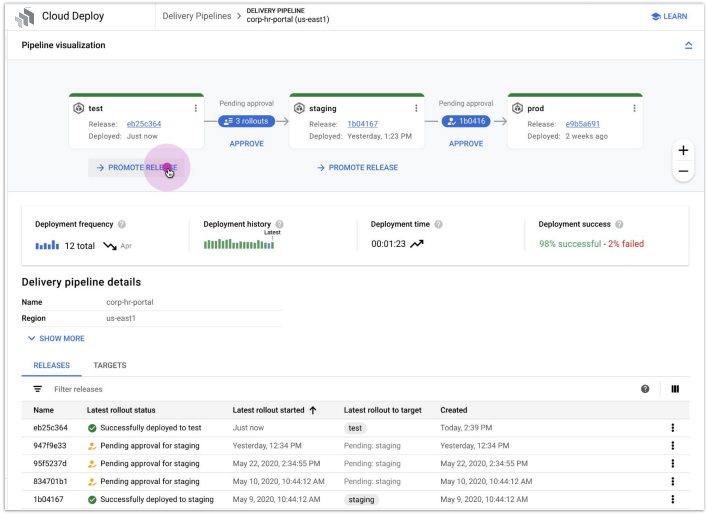


Finally, the app is delivered to the right Target; in this case, a Kubernetes cluster. Targets are also represented in a yaml format, and you can specify whether the rollout requires a manual approval. After an application has been deployed into a cluster, Google Cloud Deploy offers the ability to roll back to the previous version if a mistake occurred or to promote the release to the next environment, for example, from staging to production.



Google Cloud Deploy is modular and integrates well with existing tool chains such as Jenkins for CI. Google Cloud Deploy focuses on continuous deployment. It has the rendering or hydration of the Kubernetes manifest component and the actual deployment, that is, the image rollout to an environment. That rollout is hermetically sealed for a possible later rollback. Also, every rollout provides promotion pipelines between environments, with optional approval gates.

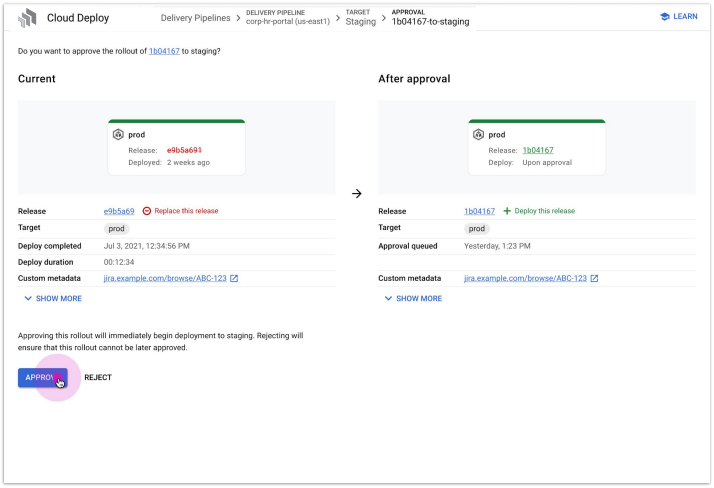
Monitor your delivery pipelines



Google Cloud

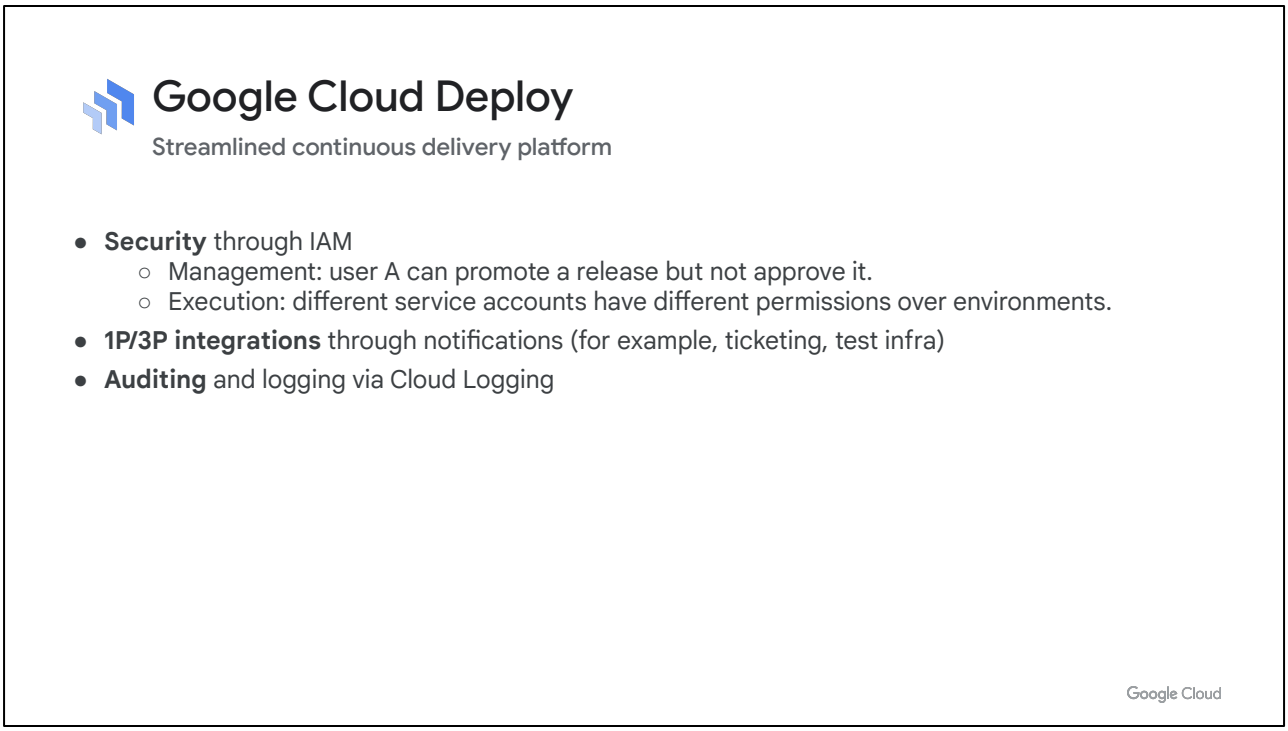
Google Cloud Deploy provides an interactive visualization of the pipeline, with the DORA Delivery Metrics that we mentioned before, so that you can measure development frequency, deployment time, and deployment success.

Include an approval process before deploying



Google Cloud

Also, after your application has gone through a testing or development environment, you can set up approvals so that the decision makers can approve or deny the release rollout into production.



Google Cloud Deploy uses IAM for security, so that:

- Management can:
  - Granularly define on a Google Cloud Deploy resource level which user can access what; for example, who can promote but not approve, or who can create or approve a release.
  - Define a specific resource through conditional IAM; for example, delivery pipeline ‘application-A’ can only be promoted by user A.
- On execution or actuation, you can use different service accounts for render and deploy operations per pipeline, which might target different environments, and thus limit access.

Third-party integration support is available through notifications so that you can do approvals from inside your existing tool or other ticketing solutions.

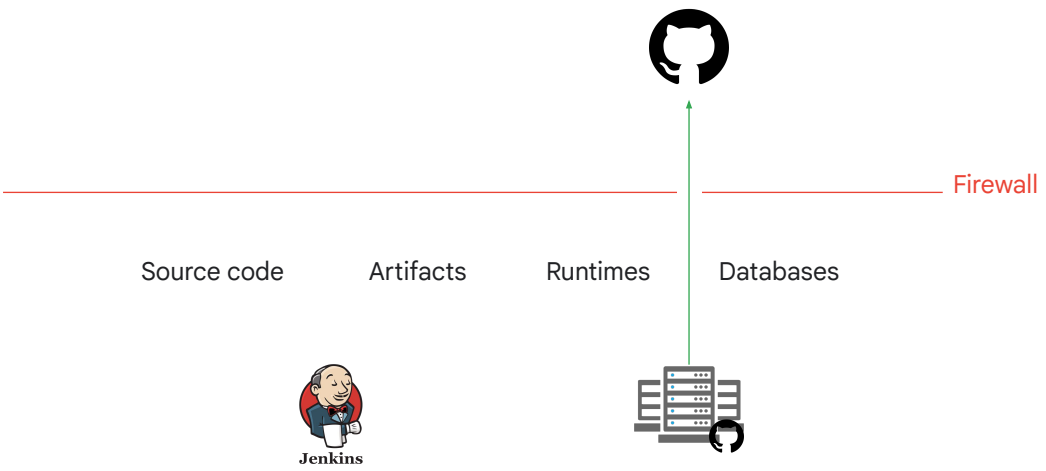
Google Cloud Deploy integrates with Cloud Logging, including Cloud Audit Logs, so an immutable record of who does what and when is always available.

CI/CD is almost never done in isolation. CI/CD tools integrate with source code, artifact repositories, runtimes, and other services to automate developer workflows.

The majority of enterprises have resources running in a private network, whether in a VPC, on-premises, or on other cloud provider. Running behind a firewall, Cloud Build cannot securely access these resources unless access is created in the firewall, which no one wants to do and we'd advise against anyway.

So, let's talk about how to deploy these CI/CD tools on a private network.

Existing solutions require self-hosting your CI/CD infrastructure



Google Cloud

Enterprises have approached this problem in two ways:

- Option 1: Self-host an open-source CI/CD solution on their own servers. This is typically the model for Jenkins.
- Option 2: Bring their own runner architectures, where they run an agent on a local server or cluster. This is what GitLab, GitHub Actions, Azure DevOps Pipelines, CircleCI, and others do.

Neither of these options is what customers actually want though.

“Hosting our own CI server in the cloud isn’t enough; it just shifts our problems to the cloud. [Instead], we need to eliminate maintenance altogether, which is why we see cloud-native CI as the best option.”

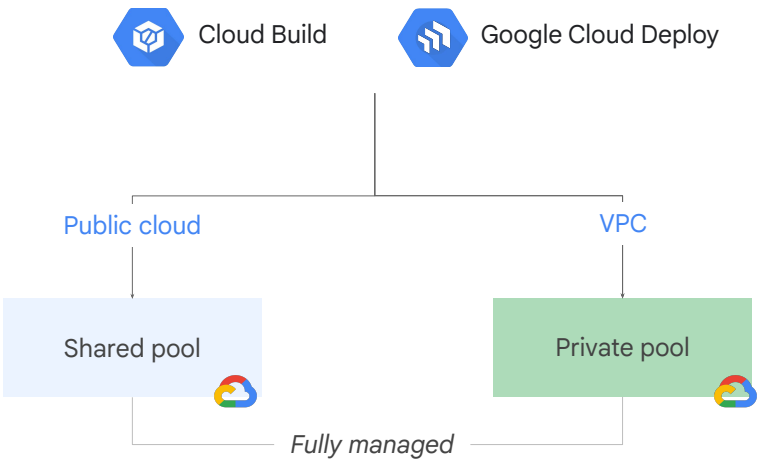
Reference customer

Forrester Wave Cloud Native CI 2019

Google Cloud

Unlike source code, artifacts, runtimes, and databases, which customers definitely want to keep on-premises, the predominant feedback is that they don’t feel this way about CI/CD tooling. They would prefer to move to a fully managed solution that is secure and works without their having to move those other resources to the cloud.

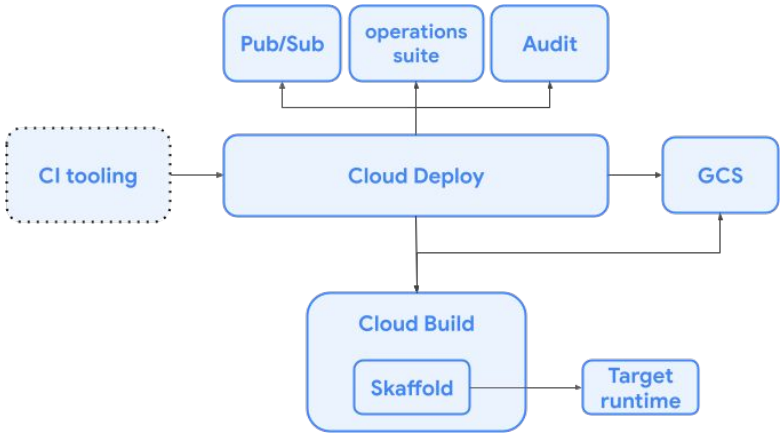
CI/CD in a private pool runs inside your VPC



Google Cloud

Cloud Build and Google Cloud Deploy offer two modes of running their worker pools in Google Cloud: in a shared pool and in a private pool. Shared pools are global and can access the internet; private pools are regional and run in a peered VPC network. Both are fully managed and share the same APIs.

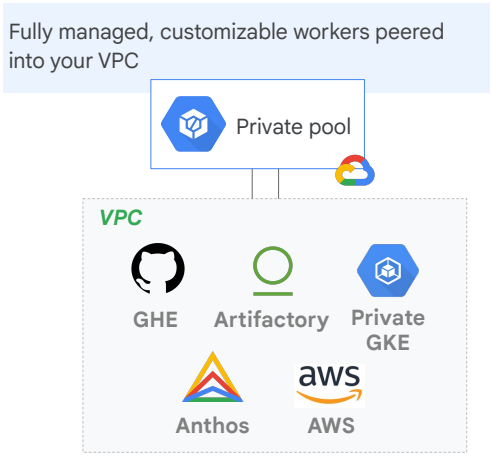
Google Cloud Deploy uses the backend from Cloud Build to perform rendering and deployments



Google Cloud

Google Cloud Deploy uses the backend from Cloud Build to perform rendering and deployments. You use the UI or CLI tools as the control plane, but the actual execution takes place in Cloud Build. This is important because all features available to Cloud Build are applicable to Google Cloud Deploy. For example, when one service is launched in regions, the other will also be available there.

CI/CD in a private pool connects with your VPC



Features	Shared Pool	Private Pool
VPC		X
VPC-SC		X
No public IP		X
Static IP ranges		X
Warm machine types	5	15
Max concurrency	30	100+
Regions	global	25

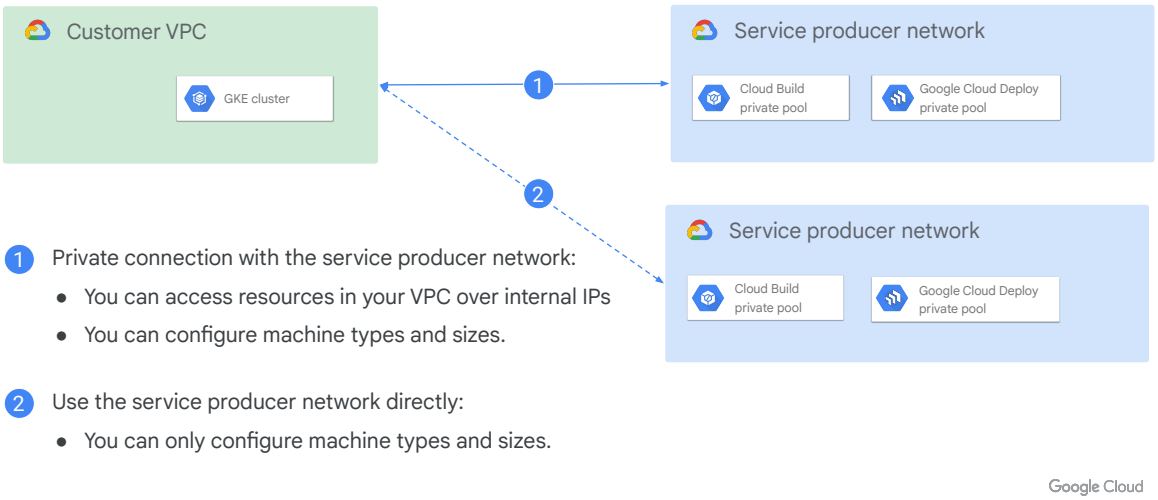
Google Cloud

Private pools allow you greater control, scale, and security. You can place the CI/CD worker pool inside a peered VPC and apply service controls to your perimeter, thus allowing only a static set of private IP addresses to deploy to your clusters. You can specify the types of machines that will perform the build process, keep a warm pool of machines to avoid long queue times, and scale to over 100 concurrent builds. Finally, if you have data locality requirements, private pools are a great option because you can choose the country to run your CI/CD pipelines in.

Notice also that Cloud Build provides integrations with other platforms such as GitHub Enterprise or Artifactory so you can continue using part of your existing tooling.

Let's look at the process of setting up private pools in Google Cloud.

Two options to configure private pools



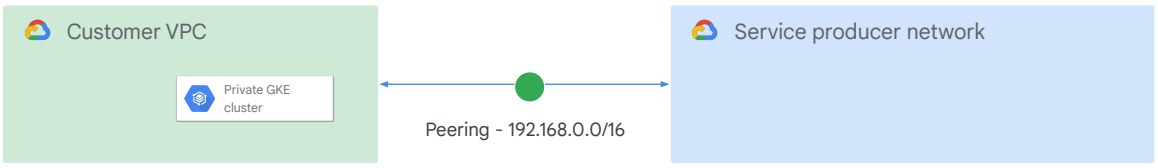
1. Private connection with the service producer network:
   - You can access resources in your VPC over internal IPs
   - You can configure machine types and sizes.
2. Use the service producer network directly:
   - You can only configure machine types and sizes.

Google Cloud

The private pools are hosted in a Google-owned Virtual Private Cloud, or VPC, network called the *service producer network*. When setting up a private pool, you can decide to peer your VPC with Google's network to communicate with your VPC resources over private IP addresses. Otherwise, you can also choose to not peer your networks, and the private pool will use a public endpoint to communicate with your VPC resources.

If you are only interested in configuring machine types and sizes or deploying in a specific region, you don't have to configure the network.

1. Set up the network for a private pool



1. Reserve a CIDR range for the service producer VPC.

```
gcloud compute addresses \
    create CI-CD-Pool \
    --global \
    --purpose=VPC_PEERING \
    --addresses=192.168.0.0 \
    --prefix-length=16 \
    --description="Pool peering" \
    --network="The_Network"
```

2. Create a private connection between the service producer network and your VPC network.

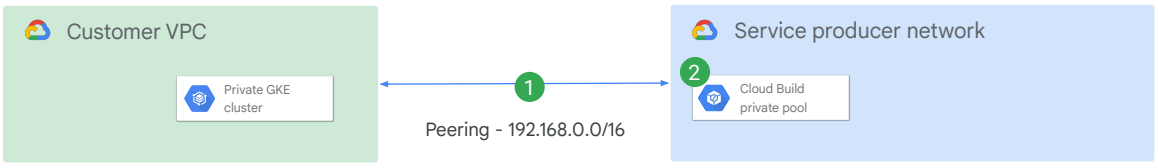
```
gcloud services vpc-peerings connect \
    --service=servicenetworking.googleapis.com \
    --ranges=CI-CD-Pool \
    --network="The_Network" \
    --network="wp123"
```

Google Cloud

If you want the private pool to communicate with your VPC resources over private IP addresses, you must connect your networks. For example, if you are running a private GKE cluster, you must perform the following steps.

Create a VPC network in your project. Then, reserve a CIDR range that does not conflict with any ranges in your subnets for Google's service producer network. Finally, using the reserved CIDR range, create a VPC peering between your VPC and Google's service producer network.

2. Create a new private pool



2 Create the private pool.

```
gcloud builds worker-pools \
    create CI-CD-Pool \
    --config-from-file=conf.yaml \
    --region=us-east-1
```

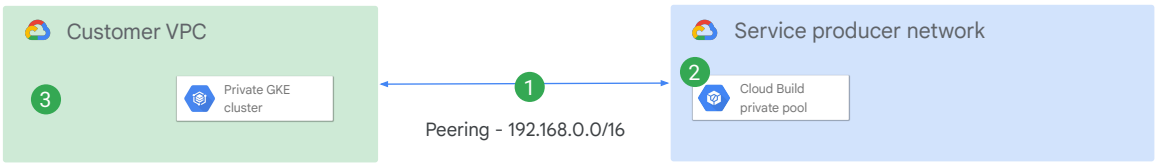
conf.yaml

```
privatePoolV1Config:
  networkConfig:
    egressOption: NO_PUBLIC_EGRESS
    peeredNetwork: CI-CD-Pool
  workerConfig:
    diskSizeGb: '512'
    machineType: e2-standard-2
```

Google Cloud

Regardless of whether you created the peering, the next step is to create the private pool on Google's service producer network. To do that, you need a configuration file that specifies the type of egress, whether it should use a peered connection, and the configuration for the worker nodes, such as the disk size and machine type. When the template is ready, run the "gcloud builds worker-pools create" command to build the private pool in the selected region.

3. Trigger a build for the private pool



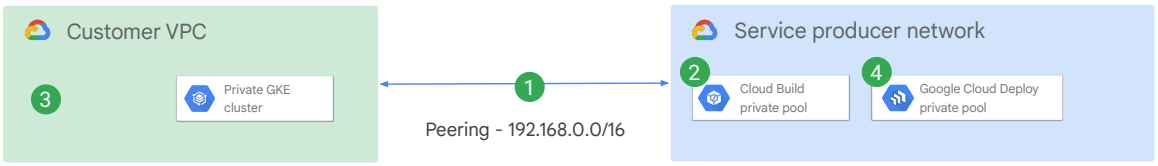
3 Submit a new build job.

```
gcloud builds submit \
    --config=cloudbuild.yaml \
    --worker-pool=projects/p123/locations/us-central1/workerPools/wp123
```

Google Cloud

Now you can start a new build with a Git trigger or by manually specifying the worker pool in a flag. You can view the progress in the Google Cloud Console in the same way as when you used a shared pool.

4. Configure Google Cloud Deploy to use the private pool



4 Configure Google Cloud Deploy to use the private pool from Cloud Build.

clouddeploy.yaml

```
executionConfigs:
  - privatePool:
      workerPool: "projects/p123/locations/us-central1/workerPools/wp123"
    usages:
    - RENDER
    - DEPLOY
```

Google Cloud

Remember that Google Cloud Deploy ultimately runs inside Cloud Build; therefore, you can specify the worker pool that it should use. You can specify whether you want to use the same pool as for the build pipeline and also decide whether the pool should apply for both the RENDER and DEPLOY pipelines or only one of them.

Today's agenda

01 CI/CD in Google Cloud

02 CI/CD in a private network

03 CI/CD in on-premises and multi-cloud environments

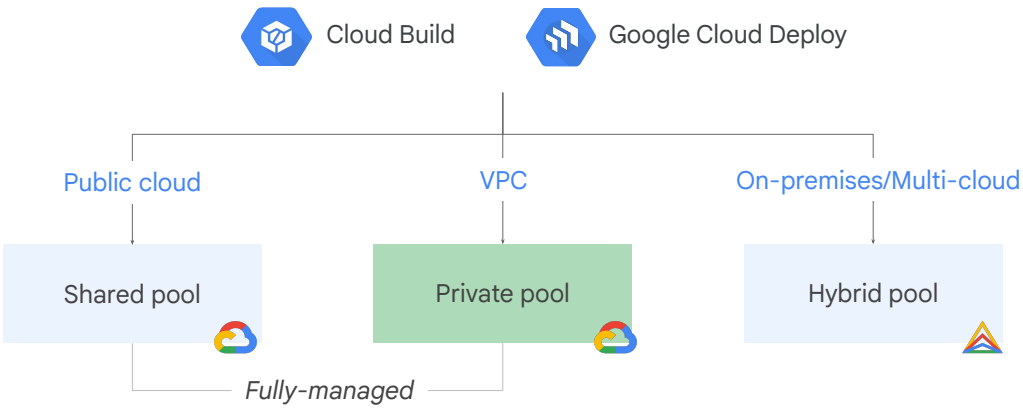
04 Securing the software supply chain

05 Deploying third-party software

Google Cloud

Let's discuss how to use these CI/CD tools when working with on-premises and multi-cloud environments.

CI/CD in a hybrid pool runs on your Anthos infrastructure on-premises or in multi-cloud environments



CI/CD in a hybrid pool runs on your Anthos infrastructure on-premises or in multi-cloud environments, which means that the Cloud Build runner is executed on your own clusters.

CI/CD in a hybrid pool offers the highest flexibility

Unified CI/CD across on-premises and cloud running on Tekton

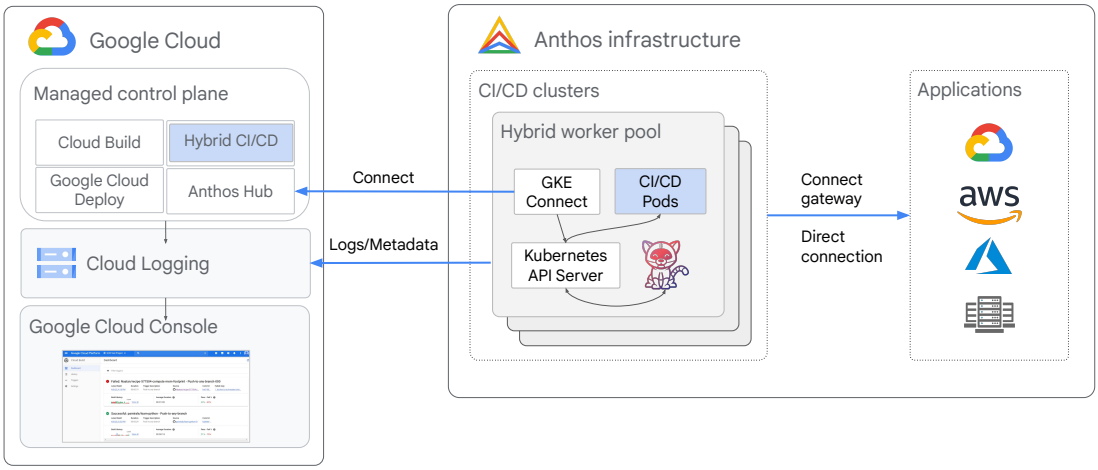
Features	Private Pool	Hybrid Pool
Run on your VPC	❌	✅
Run on-premises	❌	✅
Run on AWS/Azure	❌	✅
Access to cluster	❌	✅

Google Cloud

Hybrid pool offers the highest flexibility, because you can run your CI/CD pipelines on any Anthos environment, whether you are in a cloud environment in Google Cloud, Azure, or AWS, or your own premises running on VMware or bare metal. Cloud Build uses the open source Tekton project in the background, and hybrid pools allow Anthos customers to run Tekton CI/CD pipelines inside Kubernetes. These pipelines are 100% compatible with Cloud Build and Google Cloud Deploy, so you don't have to change your code depending on where you run and can have a unified pipeline across environments.

If you have restrictions on the location or network where you run your CI/CD pipeline or want to leverage your existing infrastructure, Cloud Build hybrid pools are the best solution for you.

Architecture for hybrid CI/CD



Let's discuss hybrid architecture for CI/CD. Going from left to right in the slide, the managed control plane here represents components of Cloud Build, Google Cloud Deploy, and the Anthos Hub.

The hybrid CI/CD controller is responsible for creating the CI/CD worker pools on user clusters. These clusters must be associated to an Anthos Fleet and registered to the Hub. The cluster has:

- Anthos connect
- Tekton

Although the underlying resources here are running on your cluster, Google deploys and manages Tekton.

Tekton dynamically creates CI/CD pods to execute CI/CD workloads. These pods can deliver applications to other Anthos connected clusters across various environments over the Connect gateway or, if they are in the same network, via a direct connection.

Tekton provides powerful CI/CD capabilities with portability so you can use Cloud Build with shared or private pools if you want a managed service on Google Cloud or use hybrid pools to run a managed service on top of Kubernetes.

Today's agenda

01 CI/CD in Google Cloud

02 CI/CD in a private network

03 CI/CD in on-premises and multi-cloud environments

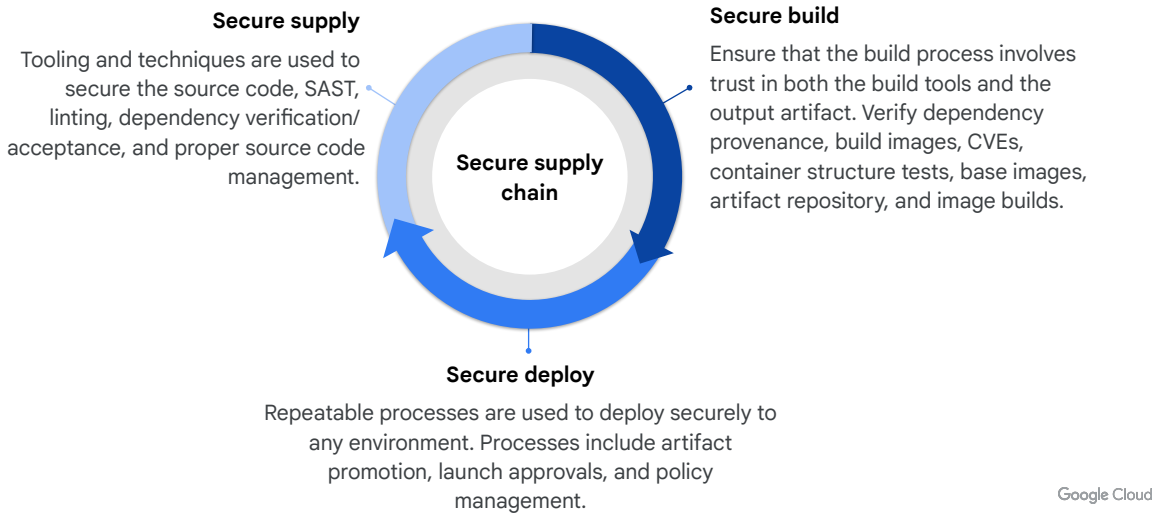
04 Securing the software supply chain

05 Deploying third-party software

Google Cloud

We talked about how to continuously build and deploy software into your hybrid and multi-cloud environments. Let's discuss how to secure this process.

Securing the software supply chain



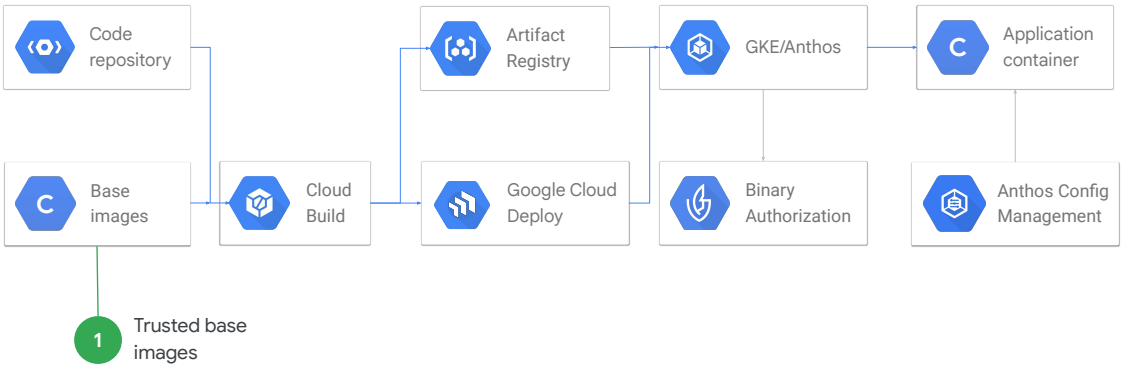
When we talk about securing the software supply chain, we mean securing the supply, securing the build, and securing the deployment.

Securing supply refers to the tooling and techniques to secure the source code, SAST, linting, dependency verification and acceptance, and proper source code management.

Securing the build means ensuring that the build process involves trust in both the build tools and the output artifact. Verify dependency provenance, build images, CVEs, container structure tests, base images, artifact repository, and image builds.

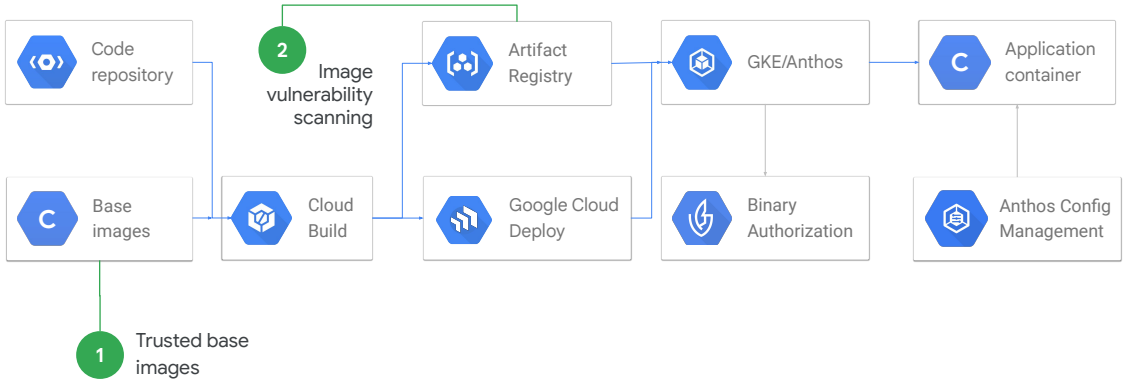
Securing the deployment describes the repeatable processes used to deploy securely to any environment: artifact promotion across the several stages, such as development or production, approvals of those launches, and policy management for the software deployed in the environments.

Deploy trusted base images



First, deploy trusted base images only. There are different ways to establish trust for your images. For example, Google provides managed base images that you can use. Alternatively, you can use distribution images, or you can create your own based on those distribution images or Google's.

Enable the vulnerability scanning in Artifact Registry



Google Cloud

Second, enable the vulnerability scanning feature in Container Registry to guard against vulnerabilities based on CVE feeds.

Artifact Registry offers more than container storage

- Achieve data locality with regional repositories
- Grant permissions at the repo level
- Use customer-managed encryption keys (CMEK)
- Secure with a VPC service perimeter
- Use audit logging for access controls

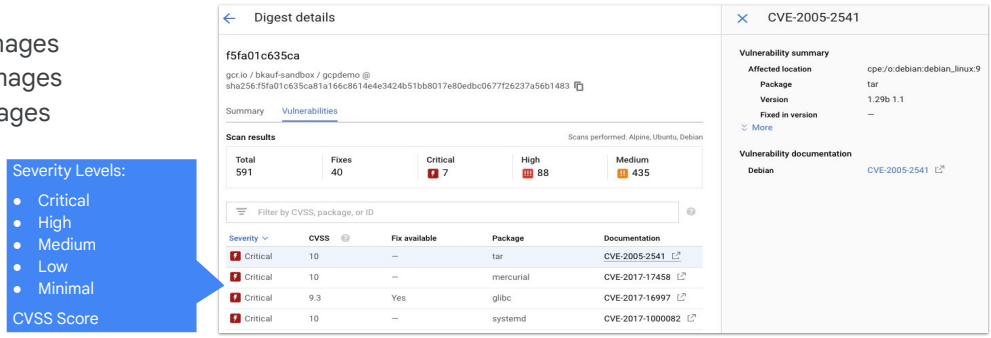
Google Cloud

Artifact Registry offers more than container storage; it provides a range of security features:

- Regional repositories reduce network egress and keep all the CI/CD pipeline in the same region, thus achieving data locality.
- Grant permissions at the repo level, with clear separation between accessing and publishing artifacts.
- Use customer-managed encryption keys, or CMEK, so that all your data is encrypted with your own keys.
- Secure with a VPC service perimeter to add an additional layer of security around your artifacts.
- Use audit logging to control who accesses artifacts.

Artifact Registry provides container analyses and vulnerability scanning

- Scans images and installed packages for Common Vulnerabilities and Exposures (CVEs) when:
  - An image is added to the registry.
  - There is an update to the database.
- Works for:
  - Debian images
  - Ubuntu images
  - Alpine images



Google Cloud

The Container Analysis API allows you to store metadata information associated with a resource. This metadata can be later retrieved to audit your software supply chain.

Container Analysis is built on top of Grafeas, an open source component metadata API that can work as a centralized authority for tracking and enforcing policies. Build, auditing, and compliance tools can use Grafeas to store, query, and retrieve comprehensive metadata about software components.

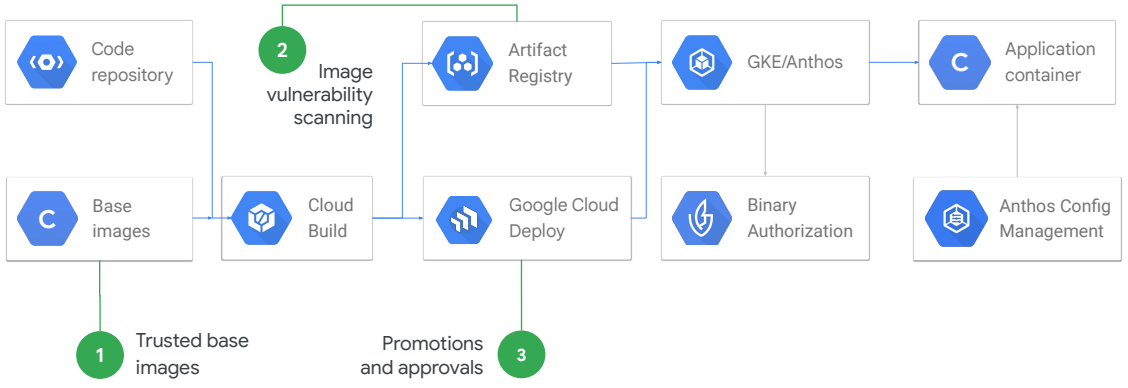
Because Grafeas is open source, you are not committed to a specific vendor. Grafeas uses a unique software identifier to associate metadata. It decouples the artifact storage so you can store metadata about components from many different repositories. The same principles apply to Container Analysis: you can use it as a centralized universal metadata store for software components in Artifact Registry or any other location.

When an image is stored in Artifact Registry, the registry automatically scans it for known Common Vulnerabilities and Exposures, or CVEs. Artifact Registry examines images and packages installed in images and works for Debian, Ubuntu, and Alpine images.

Images are scanned when:

- They are added to the registry.
- The vulnerability database is updated.

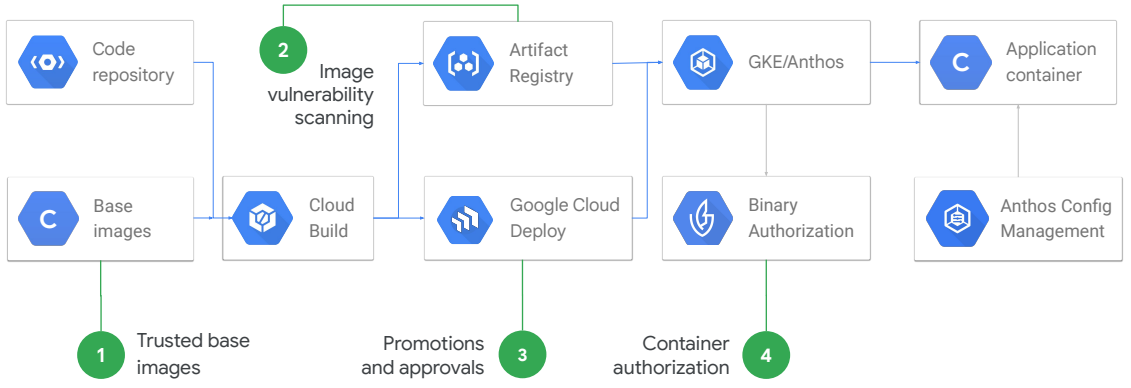
Approve releases to production



Google Cloud

As we saw before, you can add different IAM permissions for promoters of artifacts so that they can say they are ready to make a deployment, and for approvers, who are other stakeholders, who can decide when changes should go live.

Use Binary Authorization to prove your build went through all stages



Use Binary Authorization to prove that your build went through all stages of your pipeline.

Binary Authorization

Run only what you trust. This ensures that only properly signed containers are deployed to production.

- Deploy policy enforcement integrated into your GKE and Anthos clusters.
- Supports signature verification and image allow/deny lists.
- **What to verify at deploy time:**
  - Images are built by trusted CI/CD pipeline.
  - Images pass security/quality tests (vulnerability scans, QA, etc.).
  - Only explicitly allowed third-party images are deployed.

Google Cloud

Run only what you trust. Binary Authorization ensures that only properly signed containers are deployed to production.

- You can deploy policy enforcement integrated into your GKE and Anthos clusters.
- Binary Authorization supports signature verification and image allow and deny lists.
- There can be multiple verifications at deploy time, such as:
  - Images are built by a trusted CI/CD pipeline.
  - Images pass security and quality tests; for example. vulnerability scans or QA tests.
  - Only explicitly allowed third-party images are deployed.

An image is attested by the desired CI/CD stages before deployment

Google Cloud

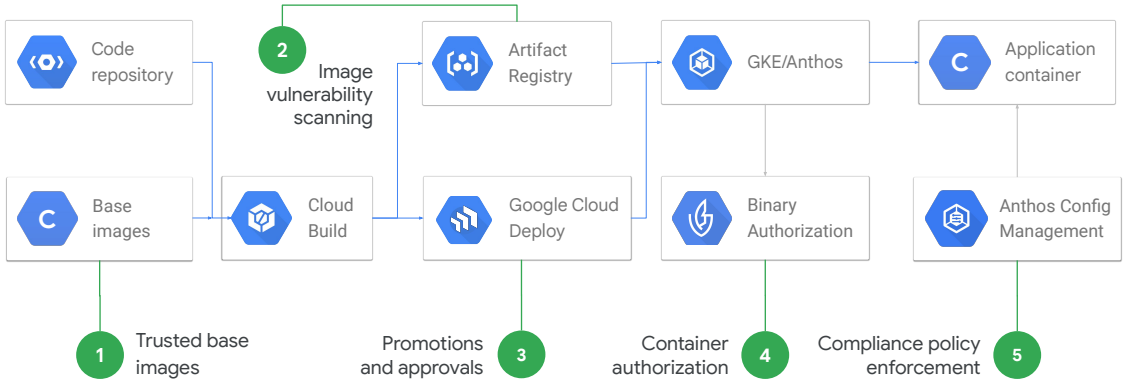
Binary Authorization formalizes and codifies an organization's deployment requirements by integrating with the desired CI/CD stages to produce signatures, or “attestations,” as an image passes through. Binary Authorization acts as an admission controller by preventing images that do not meet these criteria from being deployed. In Kubernetes Engine, this is implemented with the pod creation API, using the ImagePolicy admission controller.

Binary Authorization also integrates with Cloud Audit Logging to record failed pod creation attempts for later review.

A common source of production vulnerability is unpatched third-party software. These images are not built in-house, so they don’t have the necessary attestations from a trusted CI/CD process.

This is why, in addition to signature-based verification, Binary Authorization policy also supports using name patterns to whitelist images. A user can specify a repository, path, or particular set of images that are allowed to deploy. You can have a centralized whitelist of all the third-party images and easily identify, locate, and update when a third-party image becomes outdated or vulnerable.

Anthos Config Management enforces compliance policies

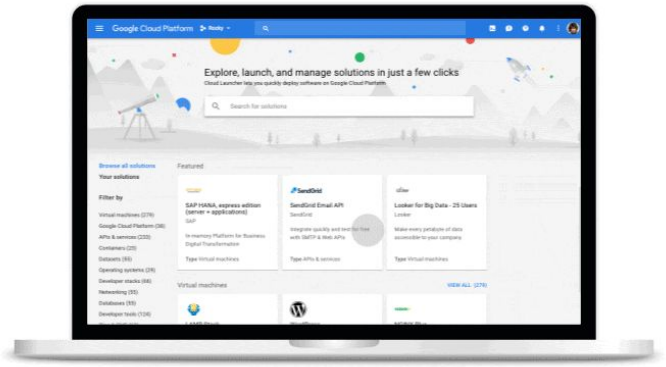


Finally, Anthos Config Management enforces compliance policies. It uses an admission controller inside your cluster, which ultimately allows applications and configurations to be deployed in your cluster.

In addition to building, securing, and deploying their own software, an enterprise wants to leverage third-party solutions to accelerate their development process while maintaining high levels of security, stability, and scalability. In this section, we learn how to do this with Anthos and Google Cloud.

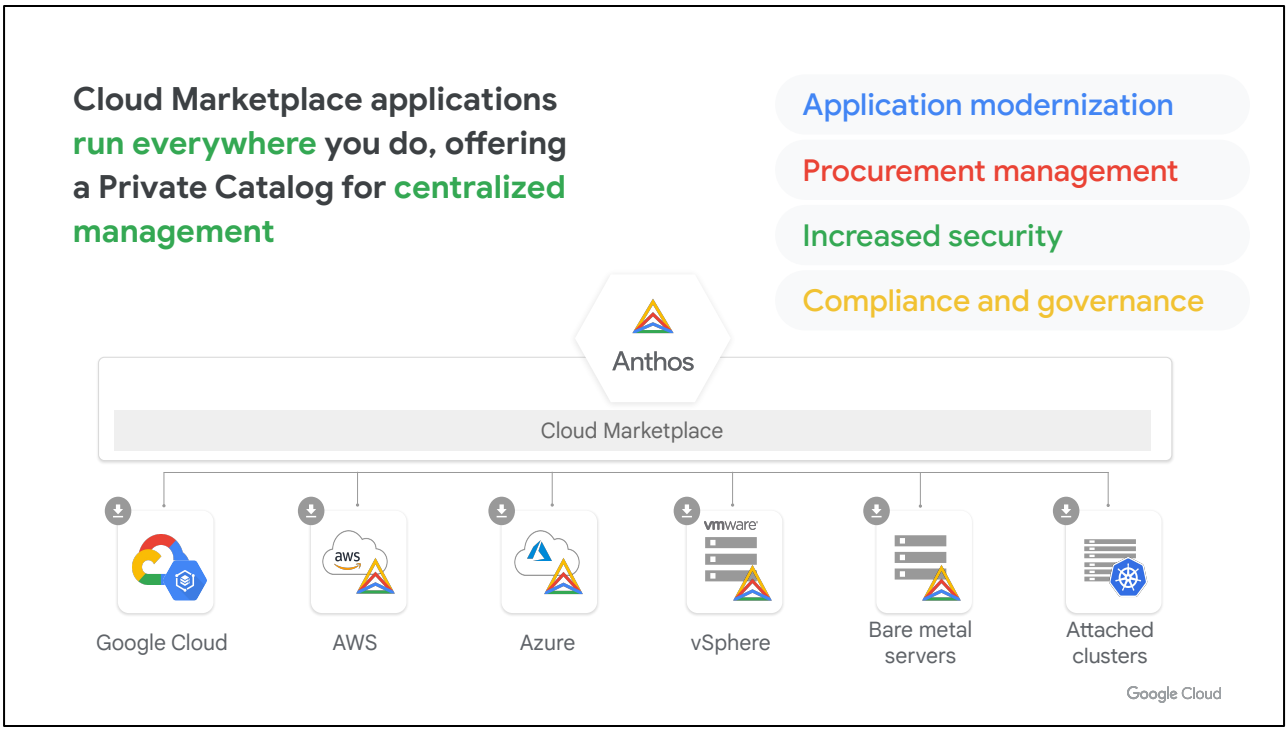
Google Cloud Marketplace is a great opportunity for both clients and partners

- Helps clients find and deploy enterprise-grade cloud solutions.
- Helps partners package, sell, manage, and upgrade their offerings.



Google Cloud

Google Cloud Marketplace is an online repository with third-party curated products that helps clients with the discovery, purchase, and fulfillment of enterprise-grade cloud solutions. Cloud Marketplace helps clients provide controls over internal apps and pay for third-party products in a single bill and helps partners package, sell, manage, and upgrade their applications and services.



Cloud Marketplace solutions run on your Anthos clusters, whether they are in Google Cloud, on-premises, or even in multi-cloud locations, thus allowing you to modernize your third-party application management by bringing the applications onto the same platform where you manage everything else.

Cloud Marketplace also offers a Private Catalog so that you can pre-approve solutions that can be procured from the Marketplace. That way, delivery times for your development teams are reduced.

All applications in Cloud Marketplace are Google-vetted against security vulnerabilities.

You can also negotiate with partners, benefit from flexible billing models, and retire Google Cloud committed spend with most transactions through Google Cloud Marketplace. All spend on solutions purchased from Google Cloud Marketplace is added into your existing Google Cloud invoice, so you receive just one bill from Google.

The Cloud Marketplace ecosystem is experiencing significant growth. Many third-party vendors have curated solutions for Google Cloud Marketplace, which they support to ensure simplified deployment and enhanced user experience. All of the vendors listed on the slide currently have Kubernetes applications in Cloud Marketplace.

Many of the Google Cloud Marketplace applications are Google Kubernetes Engine ready. Anthos-enabled Kubernetes apps available on Google Cloud Marketplace can be deployed on the GKE and Anthos clusters on-premises and multi-cloud locations, which enables clients to benefit from cloud-agnostic metering and billing. Additionally, clients can delegate to Google the management of updates to the Kubernetes platform that runs the Marketplace applications.

Offer your application on Google Cloud Marketplace

1. Open Cloud Marketplace, and register as a seller.
2. Create a public Git repo for application resources.
3. Create an Artifact Registry for your container images.
4. Set up a Google Cloud project for Google engineers to test.
5. Register the environment with Cloud Marketplace.
6. Select a pricing scheme.

Google Cloud

As you saw, many partners are offering their applications on Cloud Marketplace to create new revenue streams, and you can too. To do so, follow these steps:

- Open the Marketplace and register as a seller.
- Create a public Git repo for application resources. It can be hosted on GitHub, Cloud Source Repositories, or your own server.
- The containers must be hosted in Cloud Artifact Repository, and container analysis must be enabled. Also, as a best practice, Google recommends starting with one of the Cloud Marketplace certified container images.
- Set up a Google Cloud Project for Google engineers to test your application.
- Register your project with Cloud Marketplace.
- Select a pricing scheme.

Select a pricing scheme

- Free: user only pays for Google Cloud or other resources used elsewhere
- Bring Your Own License (BYOL): customer pays partner directly
- Google not involved with non-Google Cloud billing
- Usage-based with configurable metrics:
  - Time
  - Data processed
  - Storage time
  - Custom
- Pricing structure: single-rate or tiered
- Free trials allowable
- Private pricing for client-specific quotes

Google Cloud

You can offer your solution for free, where the client only pays for any resources consumed in the cloud, with a Bring Your Own License, or BYOL, where the customer pays the partner directly, or set up usage-based billing, where you can set up a configurable metric including time, data processed, storage time, or a custom variable. For usage-based, you must either configure a billing agent as a sidecar or directly call the billing APIs to gather usage by the client. Additional pricing options include tiered pricing, offering discounts on volume, free trials, and private pricing for specific clients.

Offer your application on Google Cloud Marketplace

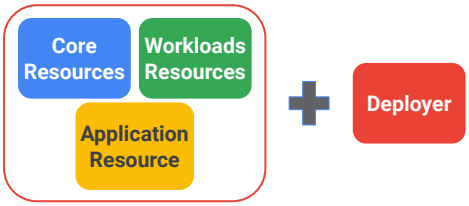
1. Open Cloud Marketplace, and register as a seller.
2. Create a public Git repo for application resources.
3. Create an Artifact Registry for your container images.
4. Set up a Google Cloud project for Google engineers to test.
5. Register the environment with Cloud Marketplace.
6. Select a pricing scheme.
7. **Create the application package**.

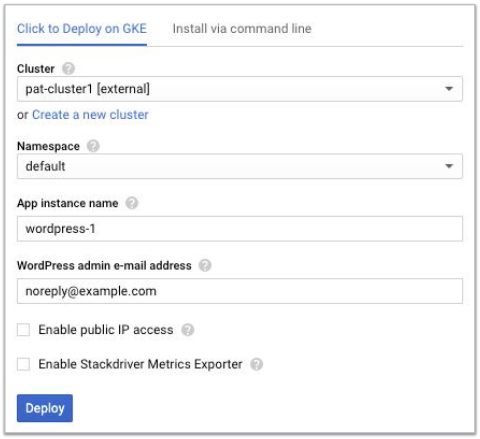
Google Cloud

- Create the application package
- Submit the app to the Marketplace
- Maintain

Create an application package in a repository

- LICENSE contains open-source licenses.
- Deployer schema or a Helm chart with parameters:
  - Cluster, namespace, app name, image, and license
  - Storage Classes for your PersistentVolumeClaims
  - Other custom attributes





Google Cloud

All files from the solution must be packaged in order to manage all resources as one application. For that, you must use either a Helm Chart or a Deployer Schema, and add a license file with all the open-source licenses that you use. Additionally, when you create your package, some fields must be configurable. For example:

- The name of the cluster, namespace, app name, container image, and license secret must always be provided as a parameter.
- If you are providing a stateful application that uses storage in the cluster, you must provide the StorageClass as a parameter. Customers will deploy your application to on-premises and multi-cloud environments, and you cannot predict the kind of storage that will be available.
- Finally, you can provide any other customization attributes that you find useful, and they will all be rendered together in the Cloud Marketplace UI.

Offer your application on Google Cloud Marketplace

1. Open Cloud Marketplace, and register as a seller.
2. Create a public Git repo for application resources.
3. Create an Artifact Registry for your container images.
4. Set up a Google Cloud project for Google engineers to test.
5. Register the environment with Cloud Marketplace.
6. Select a pricing scheme.
7. Create the application package.
8. **Submit the app to Cloud Marketplace**.
9. **Profit**.

Google Cloud

- Submit the app to Cloud Marketplace and profit!

Submit the app to Cloud Marketplace

- Solution will need a name and ID
  - Name may be changed, but ID is fixed
- Under Solution Type, select Kubernetes app
- Will also need a series of identifiers for company, product, image, and version
  - Will be used to create product and container image URLs
  - Must be URL-friendly

```
https://console.cloud.google.com/partner/solutions?project=PROJECT_ID
```

Google Cloud

When you submit the application to Cloud Marketplace, the solution will need a name and ID, where the name may be changed but the ID is fixed.

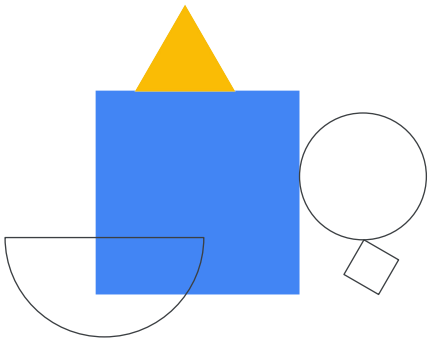
Under Solution Type, select Kubernetes app.

You will also need a series of identifiers for company, product, image, and version, which will be used to create product and container image URLs, so all fields must be URL-friendly.

Lab intro

60 min

Creating CI/CD Pipelines in Google Cloud for Anthos clusters

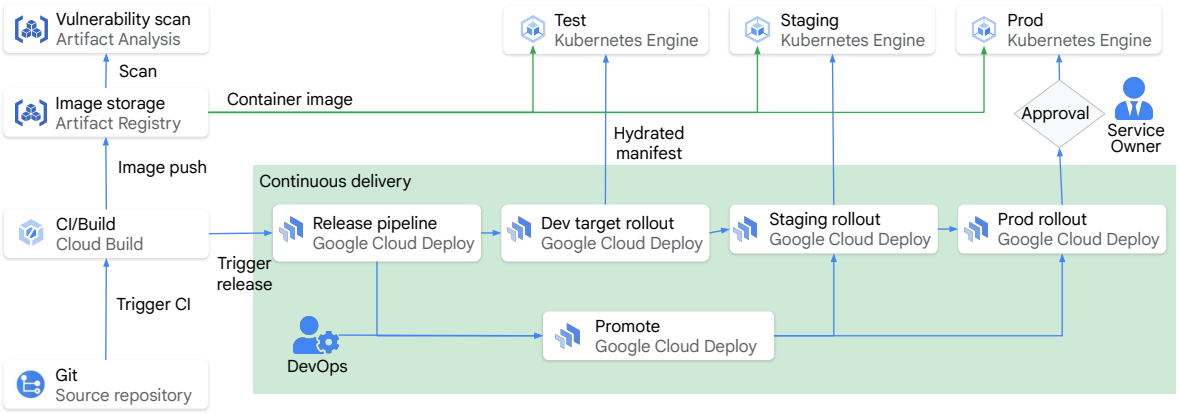


Google Cloud

Objectives:

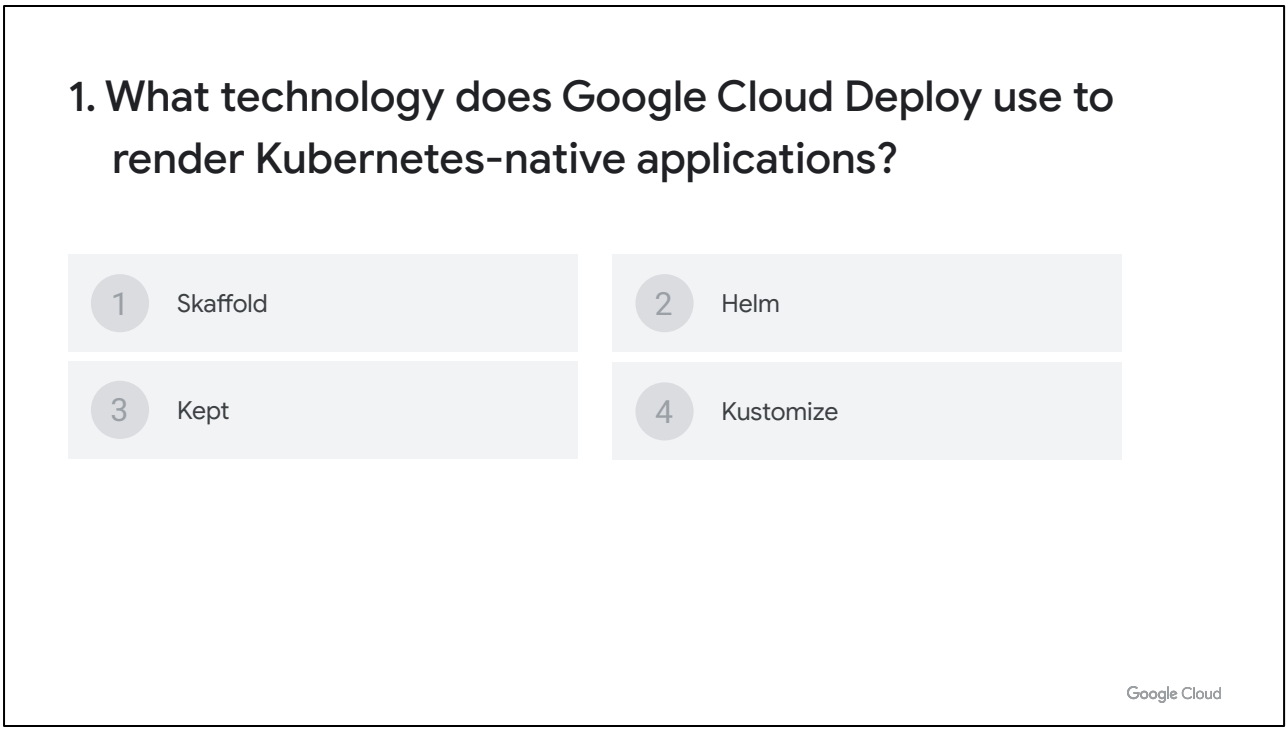
- Discover and assess the workloads for migration to containers with the Migration Fit Tool.
- Migrate and deploy the workloads into a GKE cluster using Migrate for Anthos and GKE
- Optimize the deployment

Lab 121: Creating CI/CD Pipelines in Google Cloud for Anthos clusters



Google Cloud

Questions and answers



1. What technology does Google Cloud Deploy use to render Kubernetes-native applications?

1. What technology does Google Cloud Deploy use to render Kubernetes-native applications?

1. Skaffold
2. Helm
3. Kept
4. Kustomize

Google Cloud

Skaffold is a rendering technology. Helm, Kept, and Kustomize are responsible for packaging, customizing, validating, and applying Kubernetes resources.

2. What option is used by Google Cloud Deploy and Cloud Build to run CI/CD pipelines in customer VPCs?

1. Shared pools
2. Hybrid pools
3. Tekton
4. Private pools

2. What option is used by Google Cloud Deploy and Cloud Build to run CI/CD pipelines in customer VPCs?

1. Shared pools
2. Hybrid pools
3. Tekton
4. Private pools

Google Cloud

Google Cloud Deploy and Cloud Build use private pools to run CI/CD pipelines in customer VPCs with a machine of their choice. Hybrid pools allow you to run CI/CD pipelines on your Anthos clusters, and shared pools run on Google Cloud infrastructure. Tekton is the open source technology behind Cloud Build.

3. Why would you use Binary Authorization?

1. Verify that images are built by trusted CI/CD pipelines.
2. Images pass security and quality tests, such as vulnerability scans or QA tests.
3. Only deploy explicitly allowed third-party images.
4. All the above

Google Cloud

3. Why would you use Binary Authorization?

3. Why would you use Binary Authorization?

1. Verify that images are built by trusted CI/CD pipelines.
2. Images pass security and quality tests, such as vulnerability scans or QA tests.
3. Only deploy explicitly allowed third-party images.
4. All the above

Google Cloud

These are all great reasons for using Binary Authorization.

4. What are the benefits of selling your software through Google Cloud Marketplace?

1	Customers can run your application on GKE and Anthos clusters.	2	Add new revenue streams.
3	Simplify customers' billing and use modern billing models such as pay-per-use.	4	All the above

Google Cloud

4. What are the benefits of selling your software through Google Cloud Marketplace?

4. What are the benefits of selling your software through Google Cloud Marketplace?

1	Customers can run your application on GKE and Anthos clusters.	2	Add new revenue streams.
3	Simplify customers' billing and offer modern billing models such as pay-per-use.	4	All the above

Google Cloud

All the above