Modern CI/CD with Anthos: Building a CI/CD system

This reference architecture provides you with a method and initial infrastructure to build a modern continuous integration/continuous delivery (CI/CD) system using tools such as Anthos, Skaffold, kustomize, Artifact Registry, and GitLab.

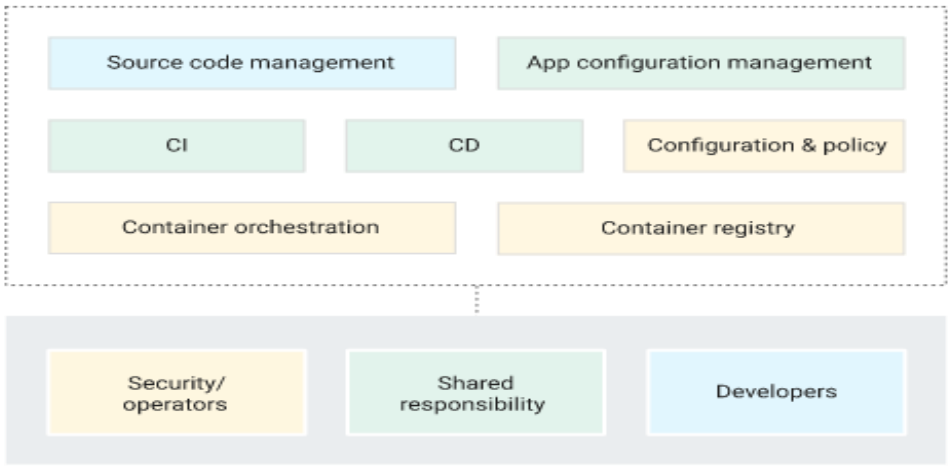
This document is part of a series:

- Modern CI/CD with Anthos: A software delivery framework
- Modern CI/CD with Anthos: Building a CI/CD system (this document)
- Modern CI/CD with Anthos: Applying the developer workflow

This document is intended for enterprise architects and application developers, as well as IT security, DevOps, and Site Reliability Engineering teams. Some experience with automated deployment tools and processes is useful for understanding the concepts in this document.

CI/CD workflow

To build out a modern CI/CD system, you first need to choose tools and services that perform the main functions of the system. This reference architecture focuses on implementing the core functions of a CI/CD system that are shown in the following diagram:



This reference implementation uses the following tools for each component:

- For source code management: GitLab

    - Stores application and configuration code.
    - Lets you review changes.
- For application configuration management: kustomize
    - Defines the desired configuration of an application.
    - Lets you re-use and extend configuration primitives or blueprints.
- For continuous integration: GitLab
    - Tests and validates source code.
    - Builds artifacts that the deployment environment consumes.
- For continuous delivery: GitLab
    - Defines the rollout process of code across environments.
    - Provides easy rollback for failed changes.
- For the infrastructure configuration and policy engine: Anthos Config Management
    - Provides a mechanism that you can use to define what is allowed to run in a given environment based on the policies of the organization.
- For container orchestration: Anthos clusters
    - Runs the artifacts that are built during CI.
    - Provides scaling, health checking, and rollout methodologies for workloads.
- For container registry: Artifact Registry
    - Stores the artifacts (container images) that are built during CI.

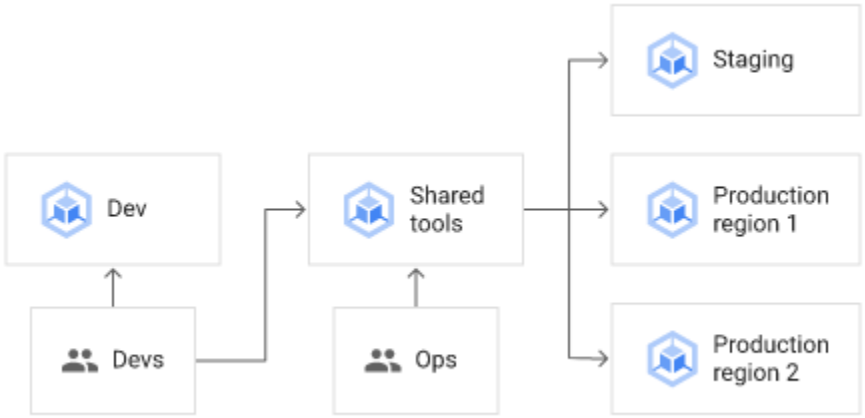
Architecture

This section describes the CI/CD components that you implement by using this reference architecture: infrastructure, code repositories, and application landing zones.

For a general discussion of these aspects of the CI/CD system, see Modern CI/CD with Anthos: A software delivery framework.

Platform infrastructure

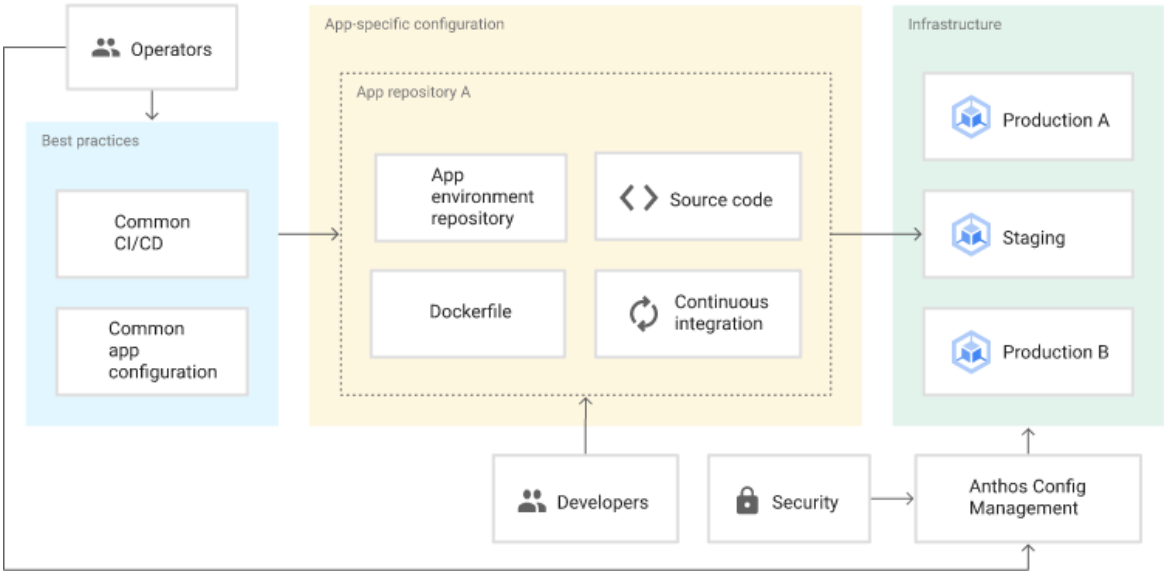
The infrastructure for this reference architecture consists of Kubernetes clusters to support development, shared tools, and application environments. The following diagram shows the logical layout of the clusters:



Code repositories

Using this reference architecture, you set up individual repositories for operators, developers, and security engineers.

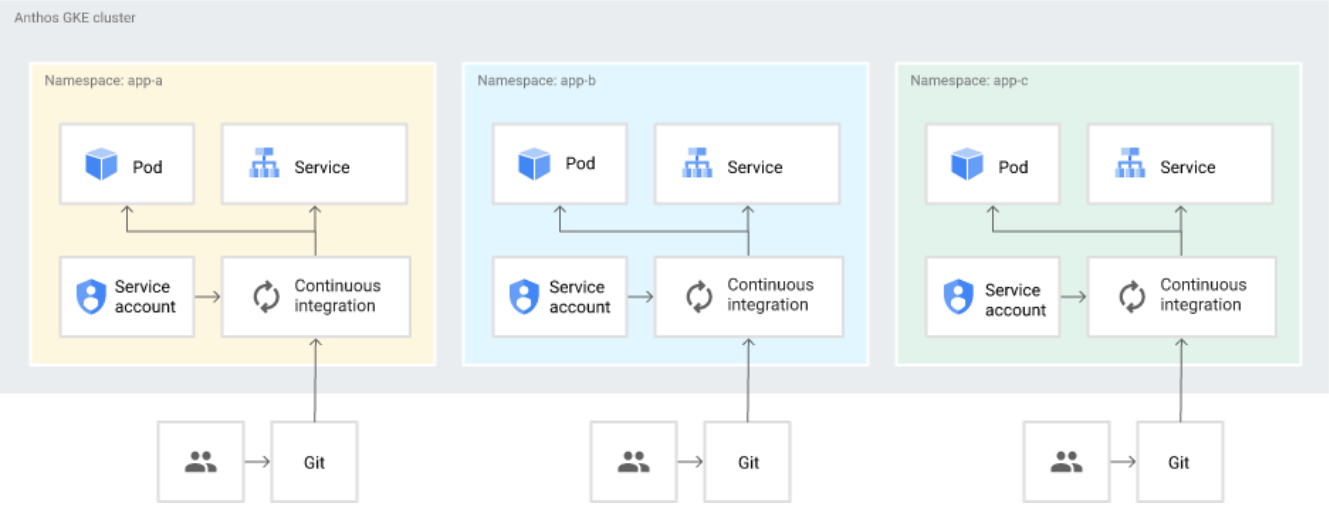
The following diagram shows the reference architecture implementation of the different code repositories and how the operations, development, and security teams interact with the repositories:



In this workflow, your operators can manage best practices for CI/CD and application configuration in the operator repository. When your developers can onboard applications in the development repository, they automatically get best practices, business logic for the application, and any specialized configuration necessary for their application to properly operate. Meanwhile, your operations and security team can manage the consistency and security of the platform in the configuration and policy repositories.

Application landing zones

The following diagram illustrates the important components of the landing zones used in this reference architecture:



Each namespace includes a service account that the CI/CD system uses to deploy Kubernetes resources such as Pods and Services. To follow the principle of least privilege, we recommend that you give the service account access only to its own namespace. You can define service account access in Anthos Config Management and implement it by using Kubernetes role-based access control (RBAC) roles and role bindings. With this model in place, teams can deploy any resources directly into the namespaces they manage but are prevented from overwriting or deleting resources from other namespaces.

Objectives

- Deploy the reference architecture infrastructure.
- Explore the infrastructure.
- Explore the code repositories and pipelines.
- Explore an example application landing zone.

Costs

This tutorial uses the following billable components of Google Cloud:

- Google Kubernetes Engine (GKE)
- Cloud SQL
- Memorystore

- Artifact Registry

To generate a cost estimate based on your projected usage, use the pricing calculator. New Google Cloud users might be eligible for a free trial.

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Clean up.

Before you begin

1. In the Google Cloud console, on the project selector page, select or create a Google Cloud project.

   Note: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

   Go to project selector
2. Make sure that billing is enabled for your Cloud project. Learn how to check if billing is enabled on a project.
3. In the Google Cloud console, activate Cloud Shell.

   Activate Cloud Shell

Deploying the reference architecture

1. In Cloud Shell, clone the Git repository:

   ```
   git clone https://github.com/GoogleCloudPlatform/solutions-modern-cicd-anthos.git
   cd solutions-modern-cicd-anthos
   ```

2. Set the environment variables for this project:

   ```
   export PROJECT_ID=PROJECT_ID
   export PROJECT_NUMBER=$(gcloud projects describe ${PROJECT_ID} --format 'value(projectNumber)')
   export REGION="us-central1"

   gcloud config set compute/region ${REGION}
   gcloud config set core/project ${PROJECT_ID}
   ```

   Replace *PROJECT_ID* with your Cloud project ID.
3. Enable the Cloud Build, Anthos, Service Usage, Cloud Key Management Service (Cloud KMS), Binary Authorization, Secret Manager, and Container Analysis APIs:

```
gcloud services enable cloudbuild.googleapis.com
gcloud services enable anthos.googleapis.com
gcloud services enable serviceusage.googleapis.com
gcloud services enable cloudkms.googleapis.com
gcloud services enable binaryauthorization.googleapis.com
gcloud services enable secretmanager.googleapis.com
gcloud services enable containeranalysis.googleapis.com
```

4. Update Identity and Access Management (IAM) roles and permissions for the Cloud Build service account:

```
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
    --member
serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com \
    --role roles/owner
gcloud projects add-iam-policy-binding ${PROJECT_ID} \
    --member
serviceAccount:${PROJECT_NUMBER}@cloudbuild.gserviceaccount.com \
    --role roles/containeranalysis.admin
```

5. Run Cloud Build to deploy the clusters:

```
gcloud builds submit --substitutions=_PROJECT_ID=${PROJECT_ID}
```

   This process takes about 30 minutes. When it finishes, you can explore the infrastructure.

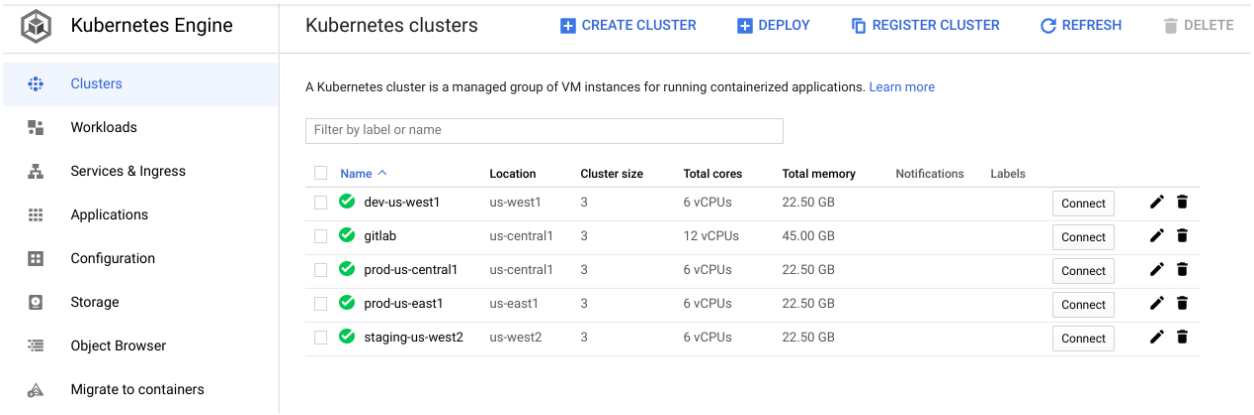
Exploring the infrastructure

In this section, you explore the main components of the CI/CD system, including the infrastructure, code repositories, and a sample landing zone.

- In the Google Cloud console, go to the **Kubernetes clusters** page.

  Go to the Kubernetes clusters page

  This page lists the clusters that are used for the development (dev-us-west1), shared tools (gitlab), and application environments (staging-us-west2, prod-us-central1, prod-us-east1):



Development cluster

The development cluster (dev-us-west1) gives your developers access to a namespace that they can use to iterate on their applications. We recommend that teams use tools like Skaffold that provide an iterative workflow by actively monitoring the code under development and reapplying it to the development environments as changes are made. This iteration loop is similar to hot reloading, but instead of being programming language-specific, the loop works with any application that you can build with a Docker image. You can run the loop inside a Kubernetes cluster.

In the next document in this series, Modern CI/CD with Anthos: Applying the developer workflow, you use Skaffold to create the development loop.

Shared tools cluster

Any software delivery system uses a mixture of tools to support the software development lifecycle. Following the principle of least privilege, the reference implementation provides a dedicated cluster for storing tools (gitlab). We recommend that you deploy each tool to its own namespace.

In this reference implementation, you use GitLab for source code management and continuous integration. You install GitLab in the tools cluster by using the GitLab on GKE Terraform module.

Application environment clusters

Also included in the reference architecture are clusters (staging-us-west2, prod-us-central1, prod-us-east1) to run your applications for both pre-production (staging) and production deployments. You should deploy your applications to at least one cluster in each environment. For geo-redundancy or high-availability (HA) systems, we recommend that you add multiple clusters to each environment. For all clusters where applications are deployed, it's ideal to use regional clusters. This approach insulates your applications from zone-level failures and any interruptions caused by cluster or node pool upgrades.

We recommend that you use Anthos Config Management to sync the configuration of cluster resources such as namespaces, quotas, and RBAC. For more details on how to manage those resources, see Configuration and policy repositories later in this document.

Exploring the code repositories

In this section, you explore the code repositories.

Log in to the GitLab instance

1. In Cloud Shell, get the GitLab URL:

   ```
   echo "https://gitlab.endpoints.${PROJECT_ID}.cloud.goog"
   ```

   Copy the URL because you need it for a later step.
2. Retrieve the GitLab User and Password, which are stored in Secrets Manager:

   ```
   export GITLAB_USER=$(gcloud secrets versions access latest --
   secret="gitlab-user")
   export GITLAB_PASSWORD=$(gcloud secrets versions access latest --
   secret="gitlab-password")

   echo "User: ${GITLAB_USER}"
   echo "Password: ${GITLAB_PASSWORD}"
   ```

   Copy these credentials because you need them for a later step.
3. In a web browser, go to the GitLab URL that you copied earlier.
4. Using the User and Password credentials that you copied, log in to your GitLab instance.

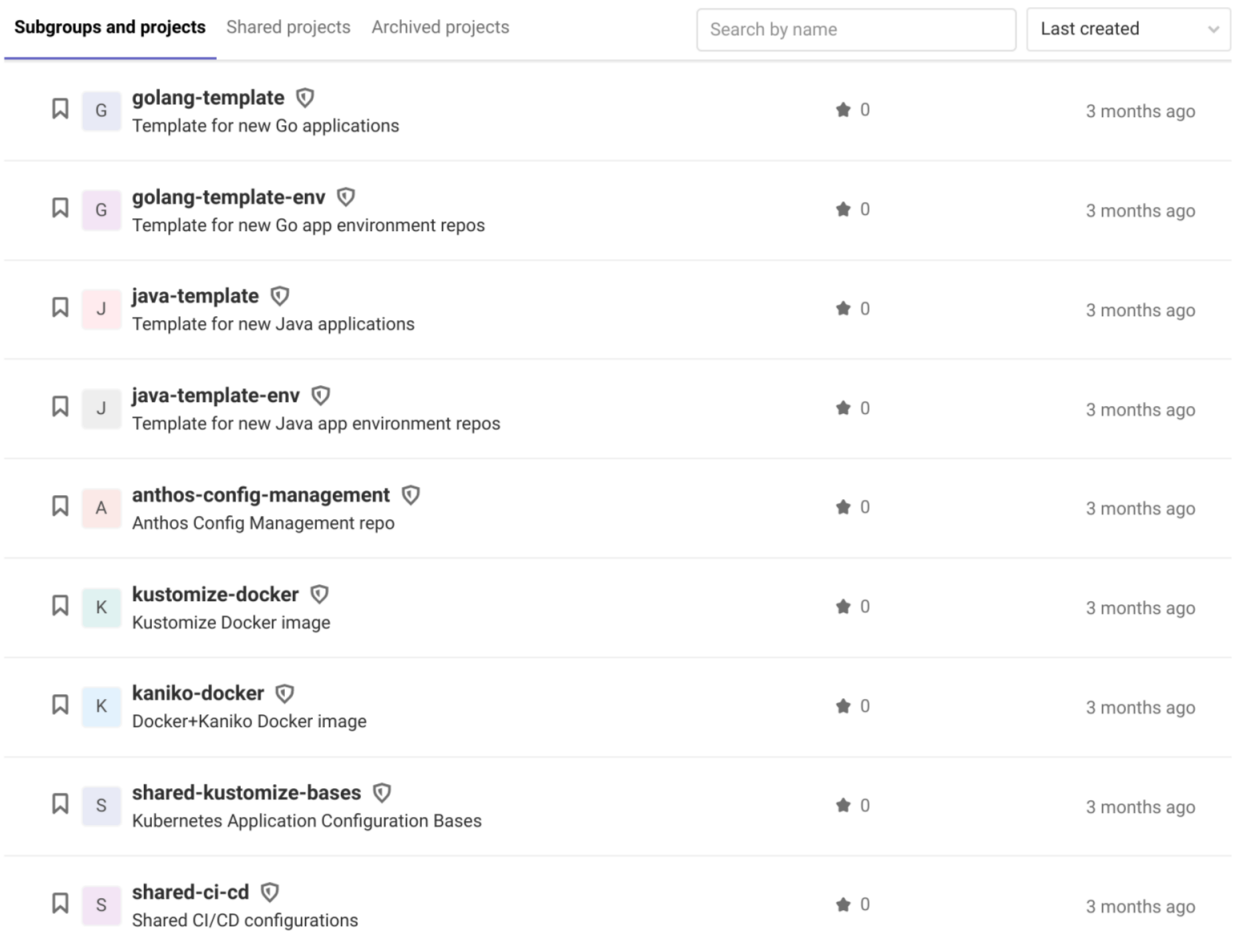
   The **Projects** page for your GitLab instance is displayed.

Explore the operator, starter, and configuration repositories

The operator, starter, and configuration repositories are where operators and platform administrators define the common best practices for building on and operating the platform. These repositories are all located in the platform-admins group.

1. In the GitLab instance, click **Groups**, and then select **Your Groups**.
2. Click **platform-admins**.

   A list of repositories is displayed:



Operator repositories

The reference architecture includes the shared-kustomize-bases and shared-ci-cd operator repositories.

- The shared-kustomize-bases repository contains the base Kubernetes manifests for the applications running in Kubernetes on the platform. Operators can update the manifests as needed, which gets picked up automatically without application teams updating individual application configurations.
- The shared-ci-cd repository stores the best practices for running CI and CD steps on the platform. Similar to the application configurations, operators can update and add stages to the best practices, and individual application pipelines are automatically updated.

In this reference implementation, operators use kustomize to manage base configurations in the shared-kustomize-bases repository. Developers are then free to extend the manifests with application-specific changes (such as resource names and configuration files) in their application repository. The kustomize tool supports configuration as data. With this

methodology, kustomize inputs and outputs are Kubernetes resources. You can use the outputs from one modification of the manifests for another modification.

The following diagram illustrates a base configuration for a Spring Boot application:

The configuration as data model in kustomize has a major benefit: when operators update the base configuration, the updates are automatically consumed by the developer's deployment pipeline on its next run without any changes on the developer's end.

For more information about using kustomize to manage Kubernetes manifests, see the kustomize documentation.

Starter repositories

In the starter repositories, your operators can codify and document best practices such as CI, metrics collection, logging, and security for applications. Included in the reference are examples of starter repositories for Go and Java applications.

The golang-template and java-template starter repositories contain boilerplate code that you can use to create new applications. The golang-template-env and java-template-env repositories contain the base configuration needed for CD. Developers and operators use these starter repositories when creating new applications.

In the next document in this series, Modern CI/CD with Anthos: Applying the developer workflow, you use the golang-template and golang-template-env repositories to create a new application.

Configuration and policy repositories

Included in the reference is an implementation of a configuration and policy repository using Anthos Config Management. The anthos-config-management repository contains the configuration and policies that you deploy across the application environment clusters. The configuration defined and stored by platform admins in these repositories is important to ensuring the platform has a consistent look and feel to the operations and development teams.

This following sections discuss how the reference architecture implements configuration and policy repositories in more detail.

Configuration

In this reference implementation, you use Anthos Config Management to centrally manage the configuration of clusters in the platform and enforce policies. Centralized management lets you propagate configuration changes throughout the system.

Using Anthos Config Management, your organization can register its clusters to sync their configuration from a Git repository, a process known as *GitOps*. When you add new clusters, the clusters automatically sync to the latest configuration and continually reconcile the state of the cluster with the configuration in case anyone introduces out-of- band changes.

For more information on Anthos Config Management, see its documentation.

Policy

In this reference implementation, Anthos Config Management leverages the Policy Controller, which is based on Open Policy Agent, to intercept and validate each request to the Kubernetes clusters in the platform. You can create policies by using the Rego policy language, which lets you fully control not only the types of resources submitted to the cluster but also their configuration.

The architecture in the following diagram shows a request flow for using Policy Controller to create a resource:

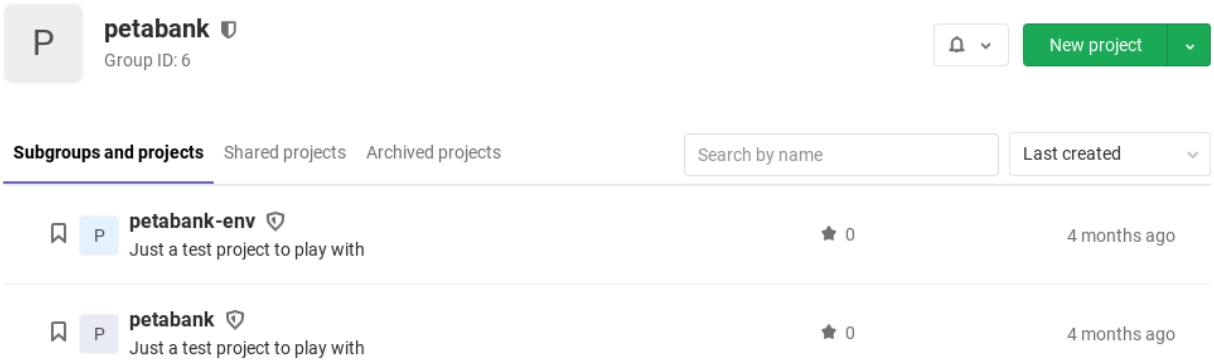
You create and define rules in the Anthos Config Management repository, and these changes are applied to the cluster. After that, new resource requests from either the CLI or API clients are validated against the constraints by the Policy Controller.

For more information about managing policies with Anthos Config Management, see the Policy Controller Overview.

Explore the application repositories

1. In the GitLab instance, click **Groups**, and then select **Your Groups**.
2. Click **petabank**.

   The application configuration and application code repositories for the petabank application are shown:



In this reference architecture, each application has two repositories: a code repository and a configuration repository.

The application code repository contains the following:

- Application source code
- A Dockerfile that describes how to build and run the application
- The CI/CD pipeline definition that uses shared tasks built by operators
- kustomize patches for each application environment

The application configuration repository contains the fully rendered Kubernetes manifests that you need to deploy the application. This repository contains a branch for each environment that the application will be deployed to.

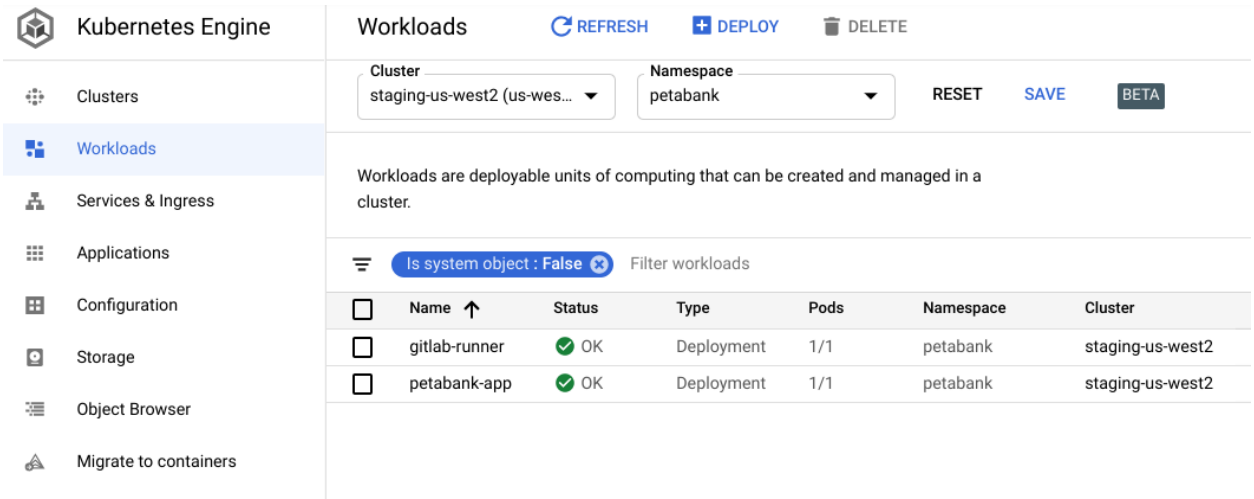
Exploring the application landing zones

The reference architecture infrastructure also includes examples of application landing zones. In this section, you explore a sample landing zone. For more information about landing zones, see Modern CI/CD with Anthos: A software delivery framework.

1. In the Google Cloud console, go to the GKE **Workloads** page.

   Go to the Workloads page
2. In the **Cluster** drop-down menu, select `staging-us-west2`.
3. In the **Namespace** drop-down menu, select **petabank**.

   A list of Kubernetes resources is displayed for the petabank application:



4. To view the Kubernetes Service resource for the petabank application, click **Services & Ingress**.

The namespace contains the resources for the petabank application, a GitLab runner for the CI/CD tasks, and a Kubernetes Deployment and Service for the petabank application. Following the principle of least privilege, RBAC and network policies are defined in the anthos-config-management repository that is stored in GitLab. You can apply the configuration and policies using Anthos Config Management.

Applying the reference architecture

Now that you've explored the reference architecture, you can explore a developer workflow that is based on this implementation. In the next document in this series, Modern CI/CD with Anthos: Applying the developer workflow, you create a new application, add a feature, and then deploy the application to the staging and production environments.

Clean up

If you want to try the next document in this series, Modern CI/CD with Anthos: Applying the developer workflow, do not delete the project or resources associated with this reference architecture. Otherwise, to avoid incurring charges to your Google Cloud account for the resources that you used in the reference architecture, you can delete the project or manually remove the resources.

Delete the project

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

2. In the Google Cloud console, go to the **Manage resources** page.

   Go to Manage resources

3. In the project list, select the project that you want to delete, and then click **Delete**.
4. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Manually remove the resources

- In Cloud Shell, remove the infrastructure:

```
gcloud builds submit --substitutions=_PROJECT_ID=${PROJECT_ID} --
config cloudbuild-destroy.yaml
gcloud endpoints services delete
gitlab.endpoints.${PROJECT_ID}.cloud.goog
gcloud endpoints services delete
registry.endpoints.${PROJECT_ID}.cloud.goog
```