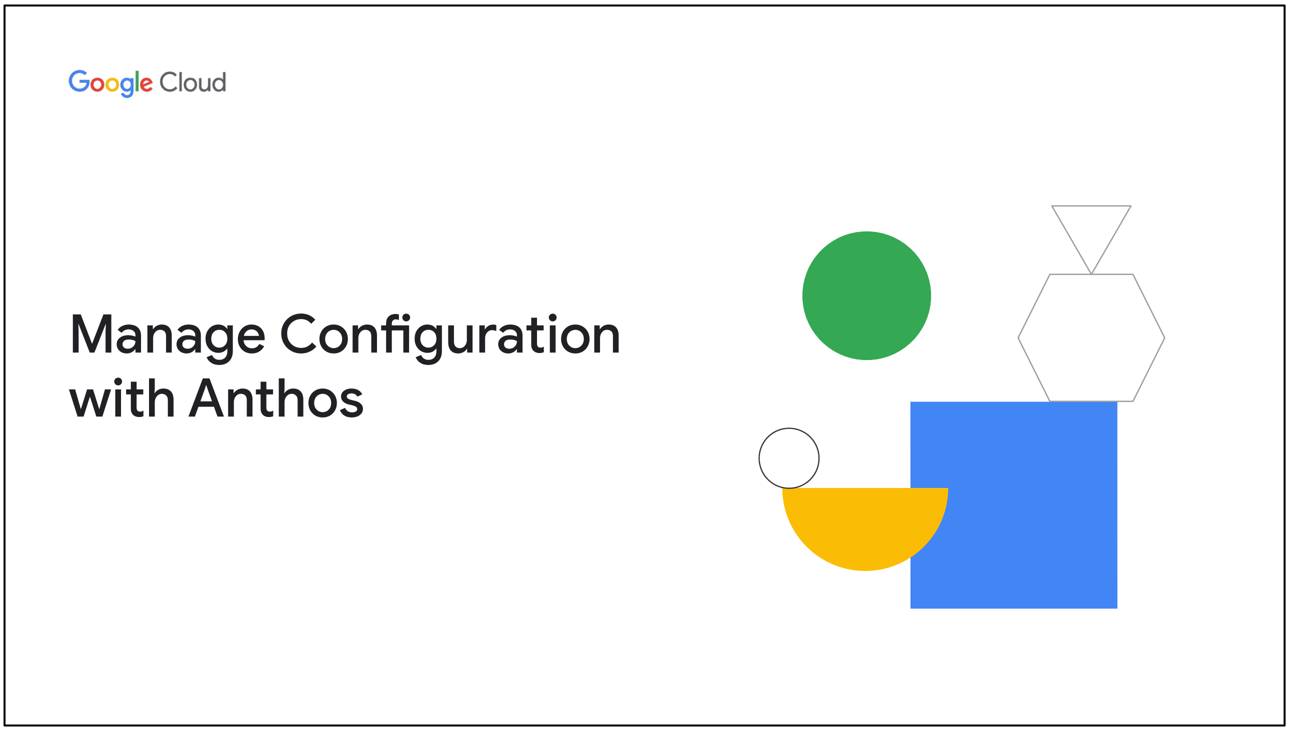
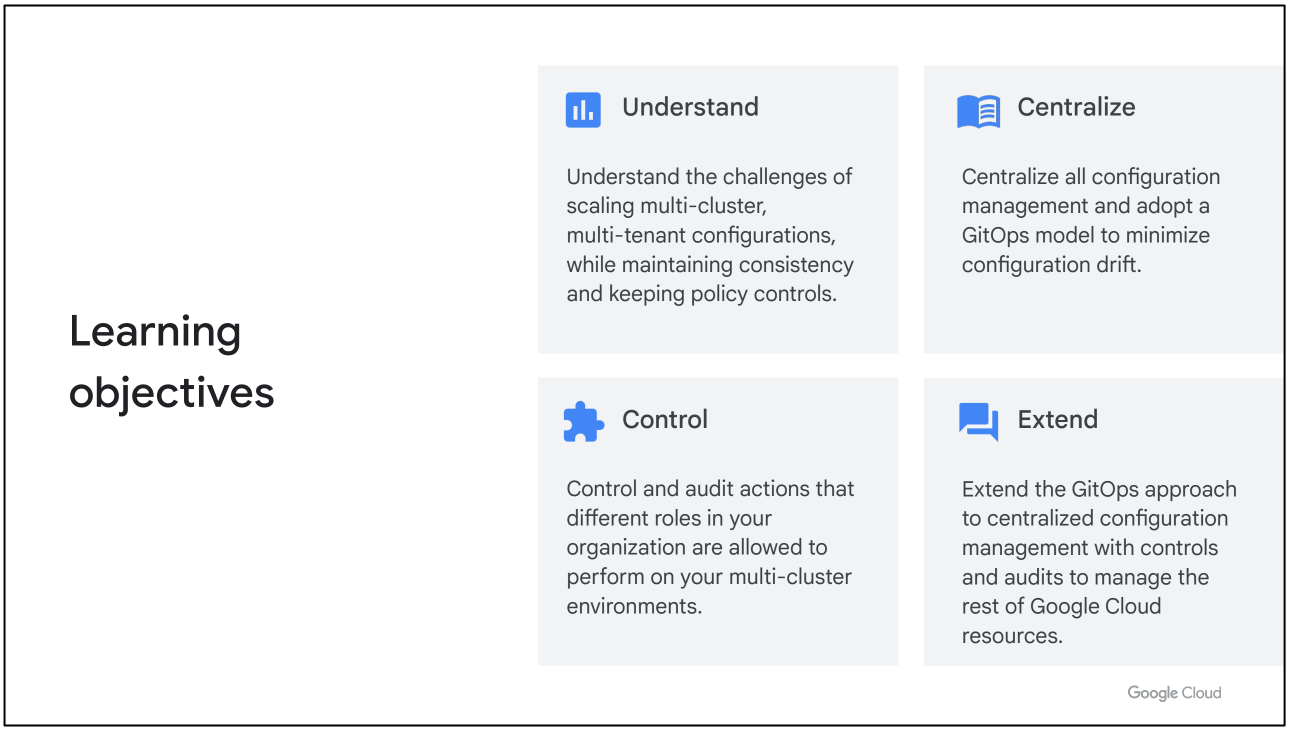


Google Cloud

Manage Configuration with Anthos

Welcome to Managing Configurations with Anthos.



Learning objectives

Understand

Understand the challenges of scaling multi-cluster, multi-tenant configurations, while maintaining consistency and keeping policy controls.

Centralize

Centralize all configuration management and adopt a GitOps model to minimize configuration drift.

Control

Control and audit actions that different roles in your organization are allowed to perform on your multi-cluster environments.

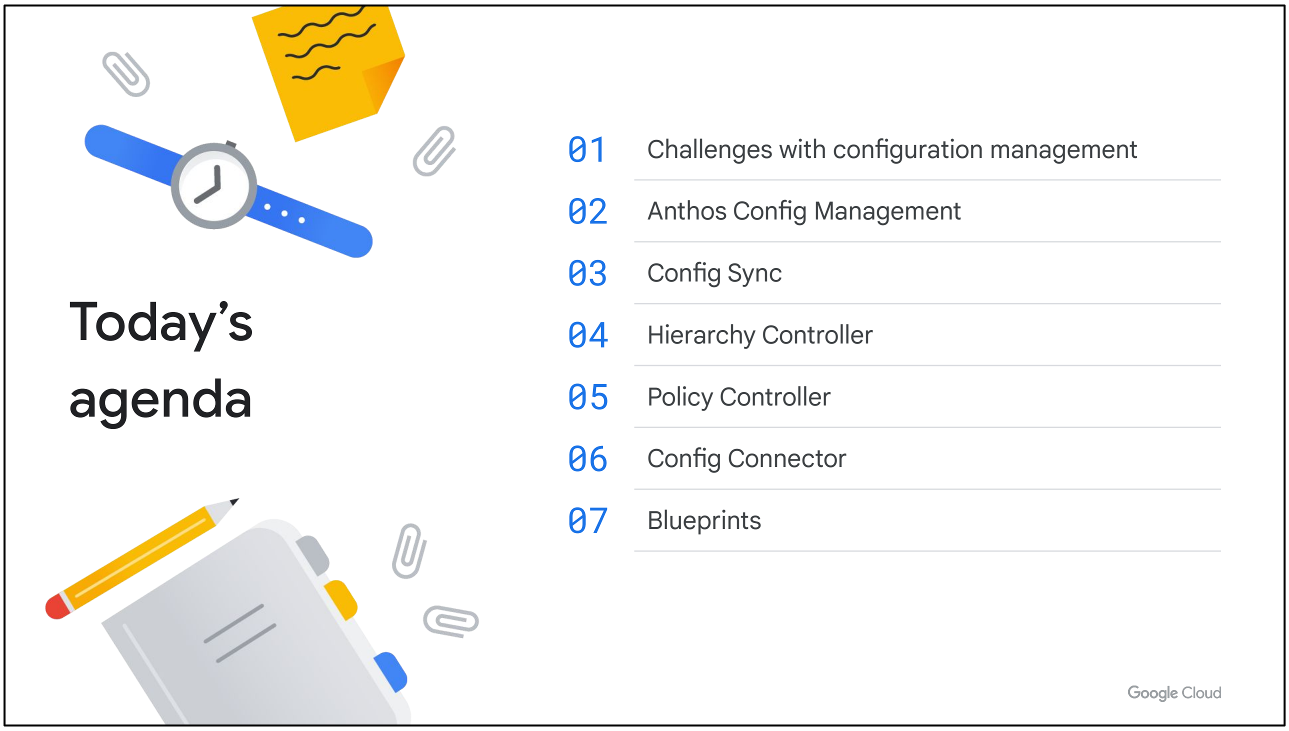
Extend

Extend the GitOps approach to centralized configuration management with controls and audits to manage the rest of Google Cloud resources.

Google Cloud

In this module, we:

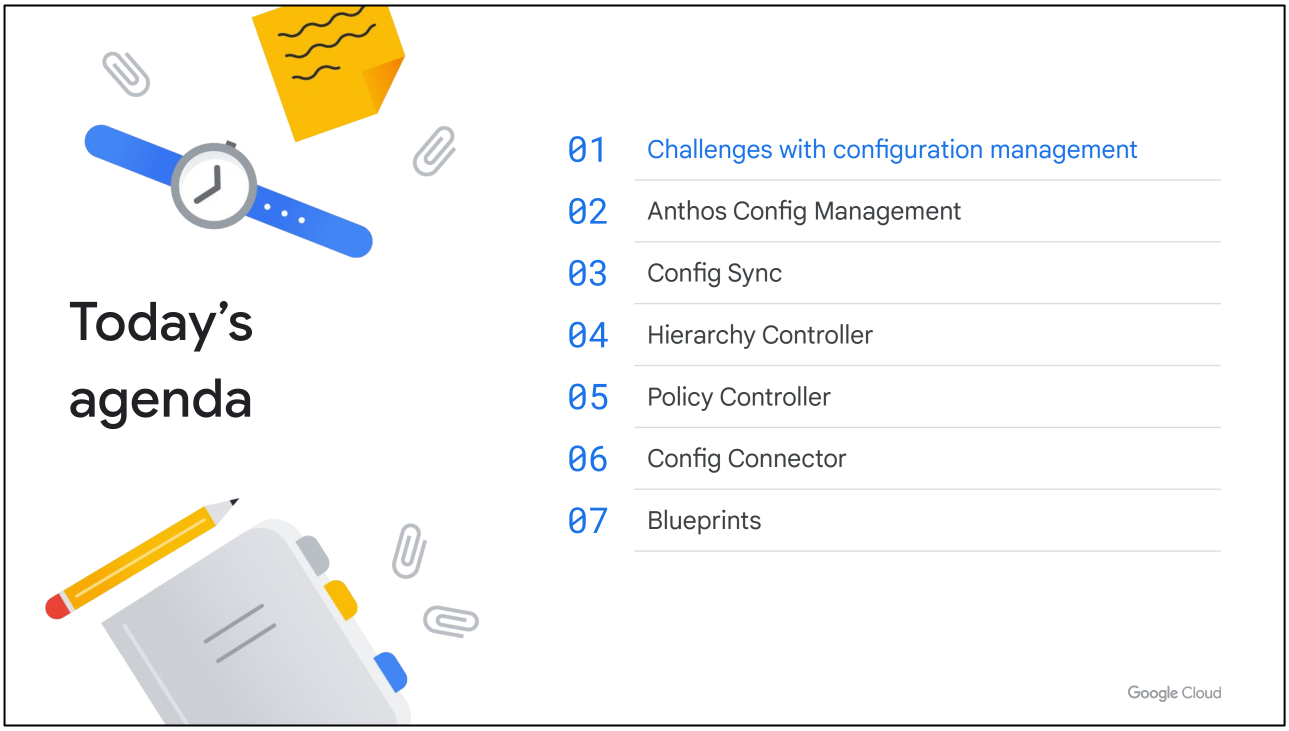
- Understand the challenges of scaling multi-cluster, multi-tenant configurations, while maintaining consistency and keeping policy controls.
- Centralize all configuration management and adopt a GitOps model to minimize configuration drift.
- Control and audit actions that different roles in your organization are allowed to perform on your multi-cluster environments.
- Extend the GitOps approach to centralized configuration management with controls and audits to manage the rest of Google Cloud resources.



Today's agenda

Google Cloud

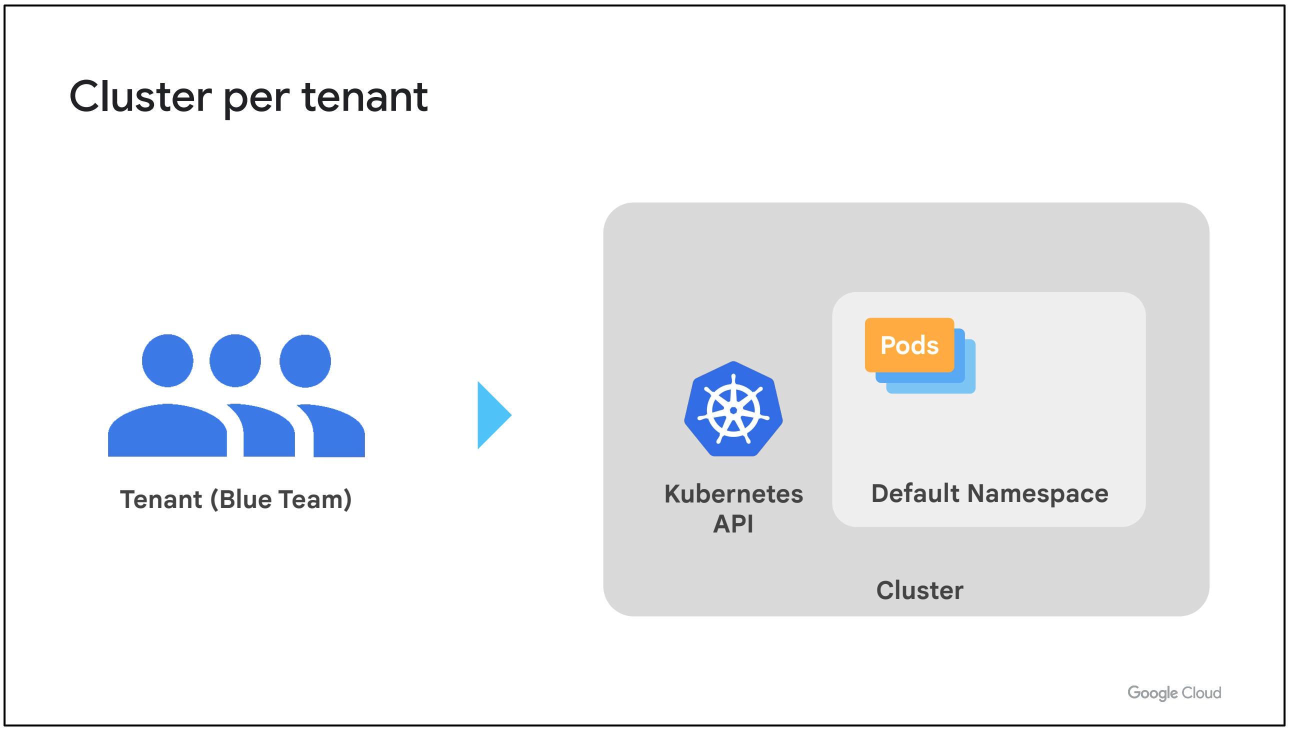
Here is our agenda.



Today's agenda

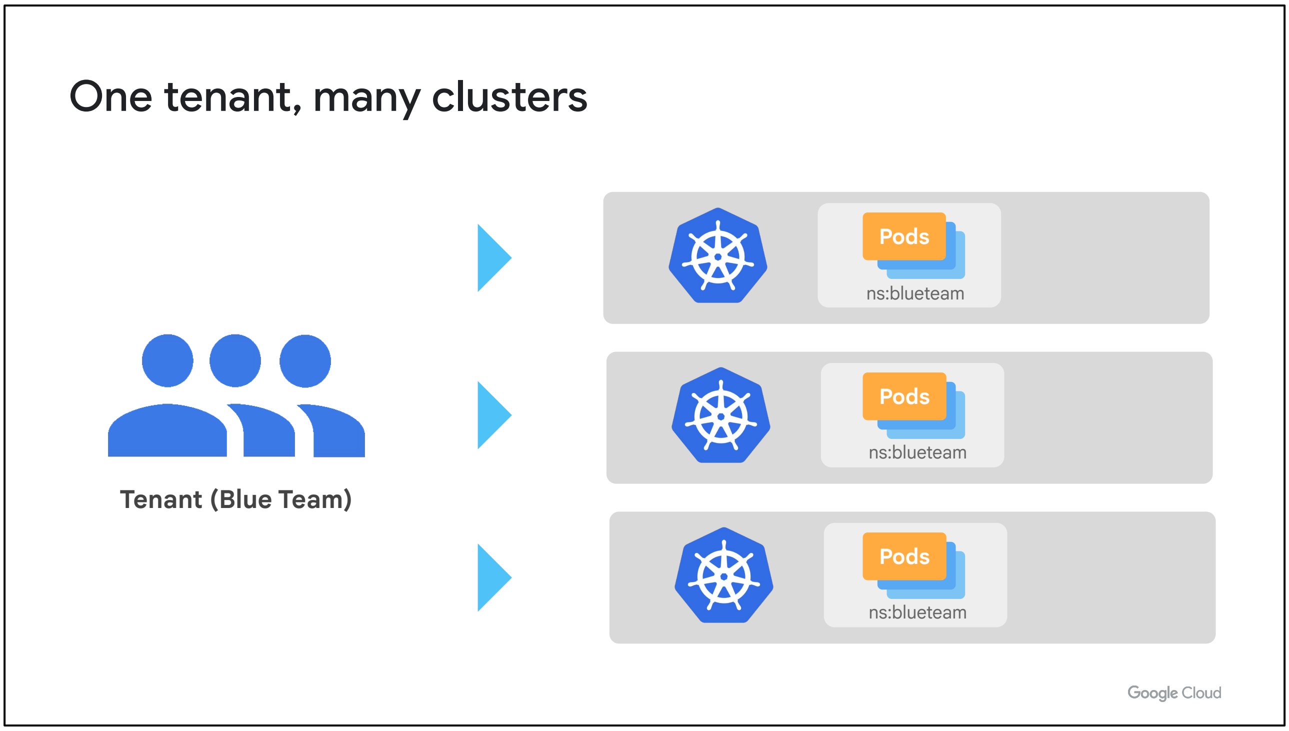
Google Cloud

We start with the challenges with configuration management.



Cluster per tenant

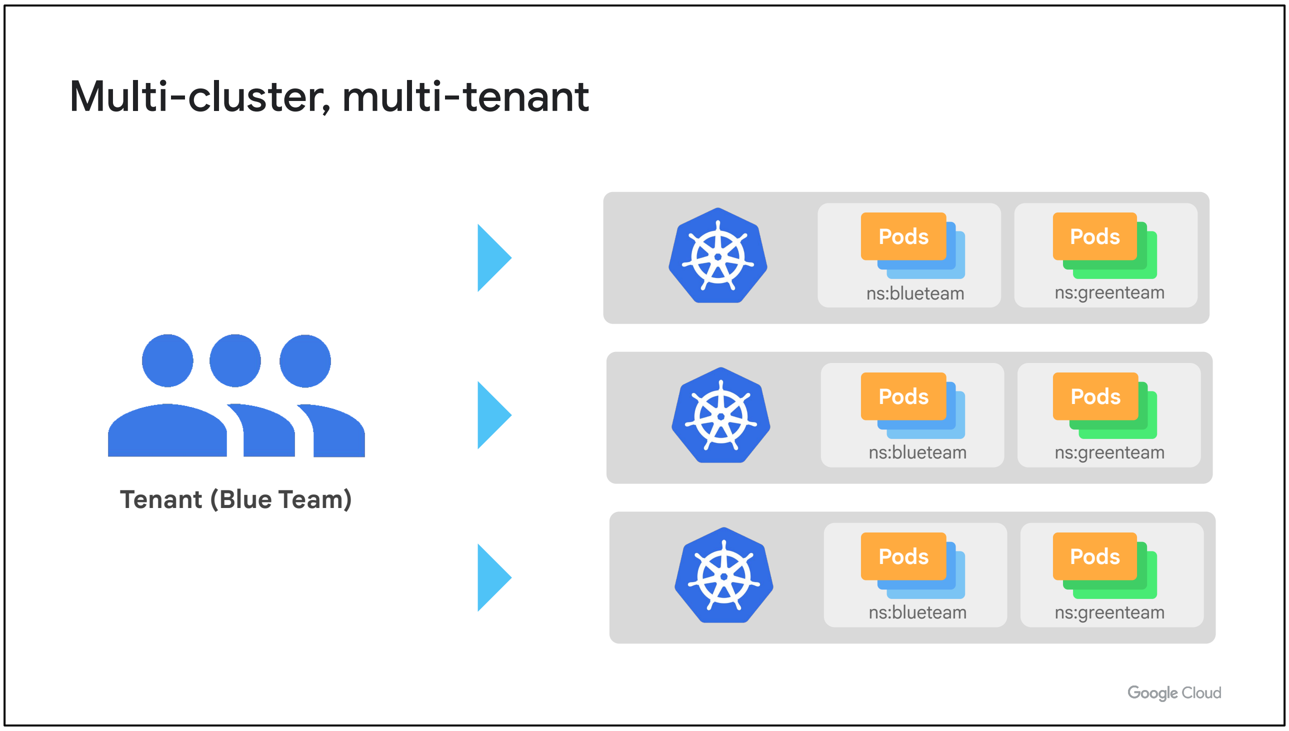
When you’re starting out, it’s simple to give each dev team their own cluster.



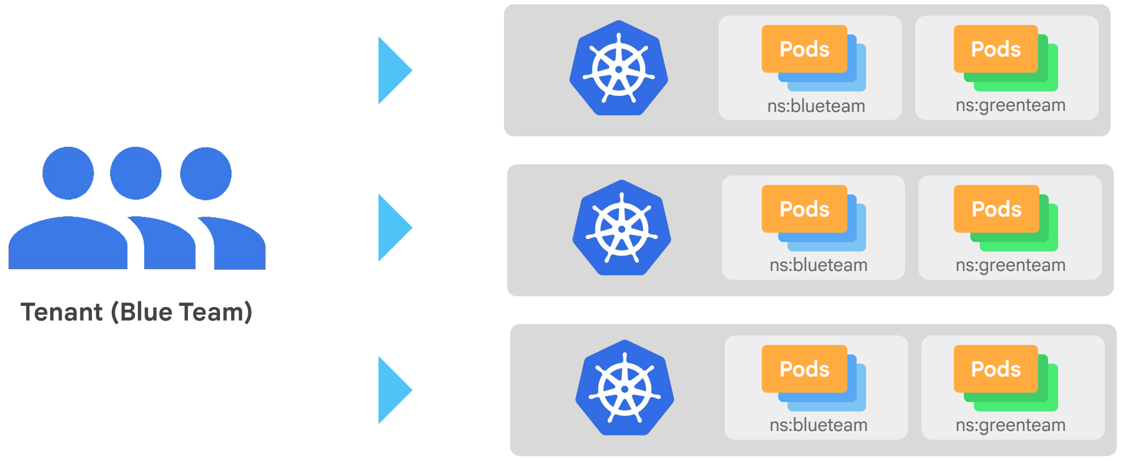
One tenant, many clusters

That team may quickly outgrow their cluster, which means you now need to provision the same environment across multiple clusters.

This happens when you need redundancy, a staging environment, multiple regions, etc.

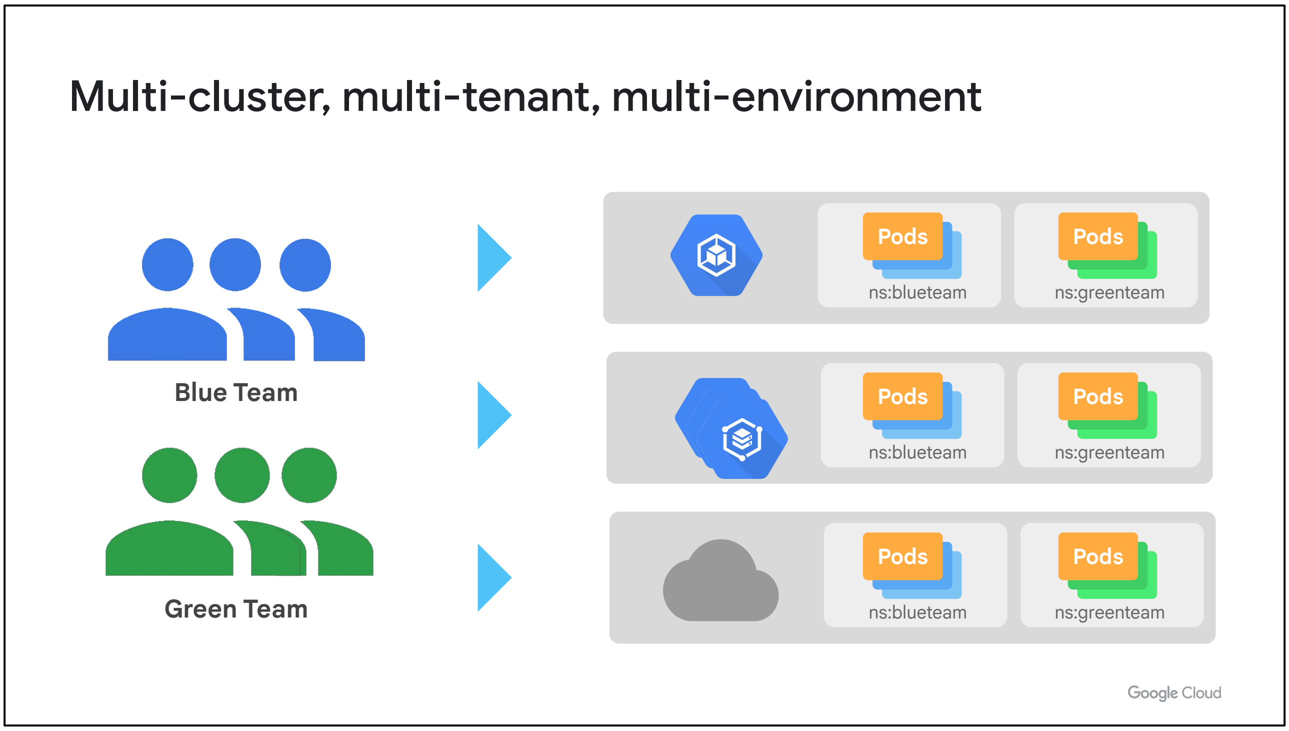


Multi-cluster, multi-tenant



Google Cloud

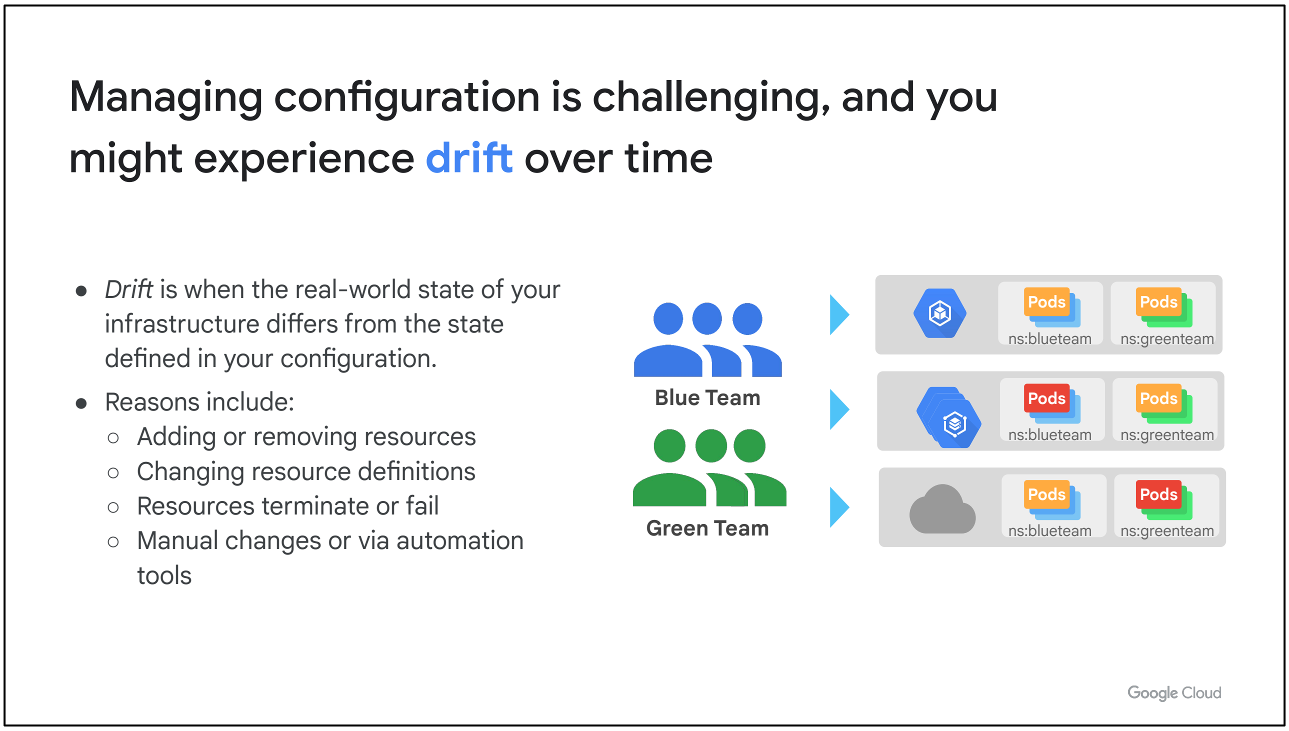
X tenants times Y namespace times Z clusters means exponential growth in the number of policy applications that your team needs to manage.



Multi-cluster, multi-tenant, multi-environment

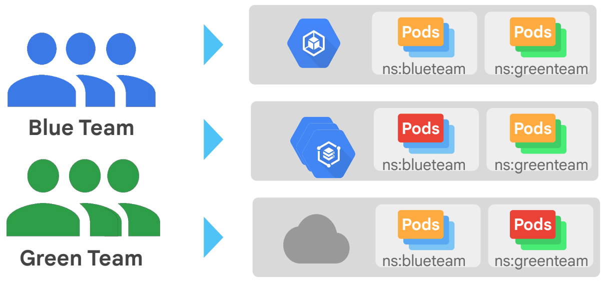
Blue Team
Green Team
Pods
ns:blueteam
Pods
ns:greenteam
Pods
ns:blueteam
Pods
ns:greenteam
Pods
ns:blueteam
Pods
ns:greenteam
Google Cloud

On top of all that, you need to maintain consistency and compliance across geographically distributed environments.



Managing configuration is challenging, and you might experience drift over time

- *Drift* is when the real-world state of your infrastructure differs from the state defined in your configuration.
- Reasons include:
  - Adding or removing resources
  - Changing resource definitions
  - Resources terminate or fail
  - Manual changes or via automation tools

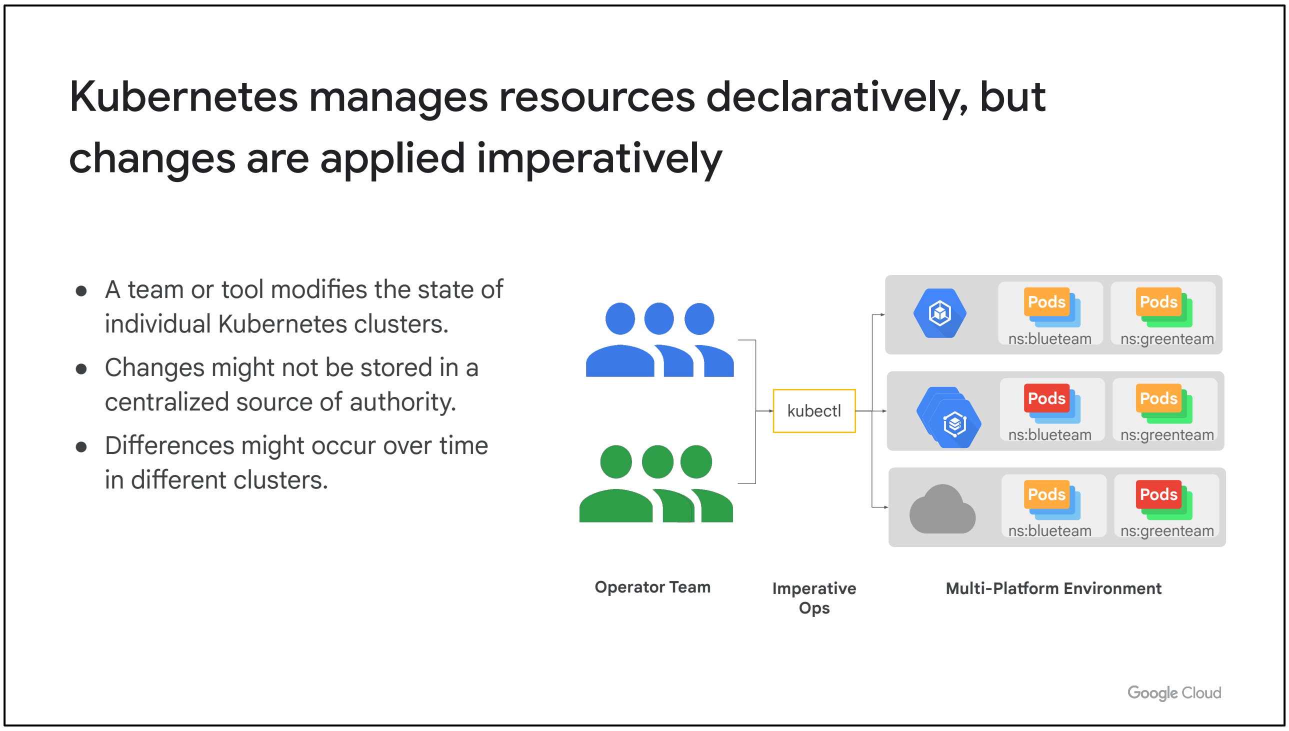


Google Cloud

Managing configuration is challenging, and over time, you might experience drift.

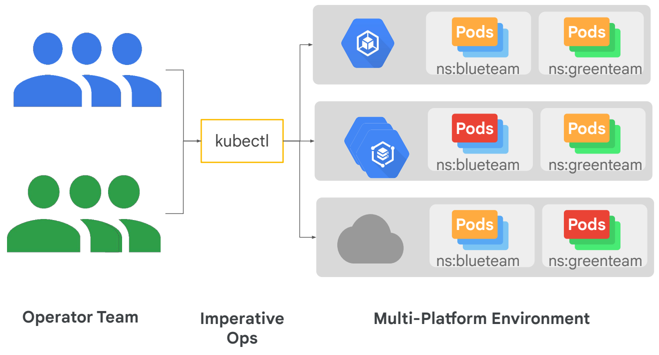
Drift is the term used for when the real-world state of your infrastructure differs from the state defined in your configuration.

There are different reasons why drift might occur, for instance: adding or removing resources, changing resource definitions, resources terminating or failing, or other infrastructure changes either manual or via automation tools.



Kubernetes manages resources declaratively, but changes are applied imperatively

- A team or tool modifies the state of individual Kubernetes clusters.
- Changes might not be stored in a centralized source of authority.
- Differences might occur over time in different clusters.



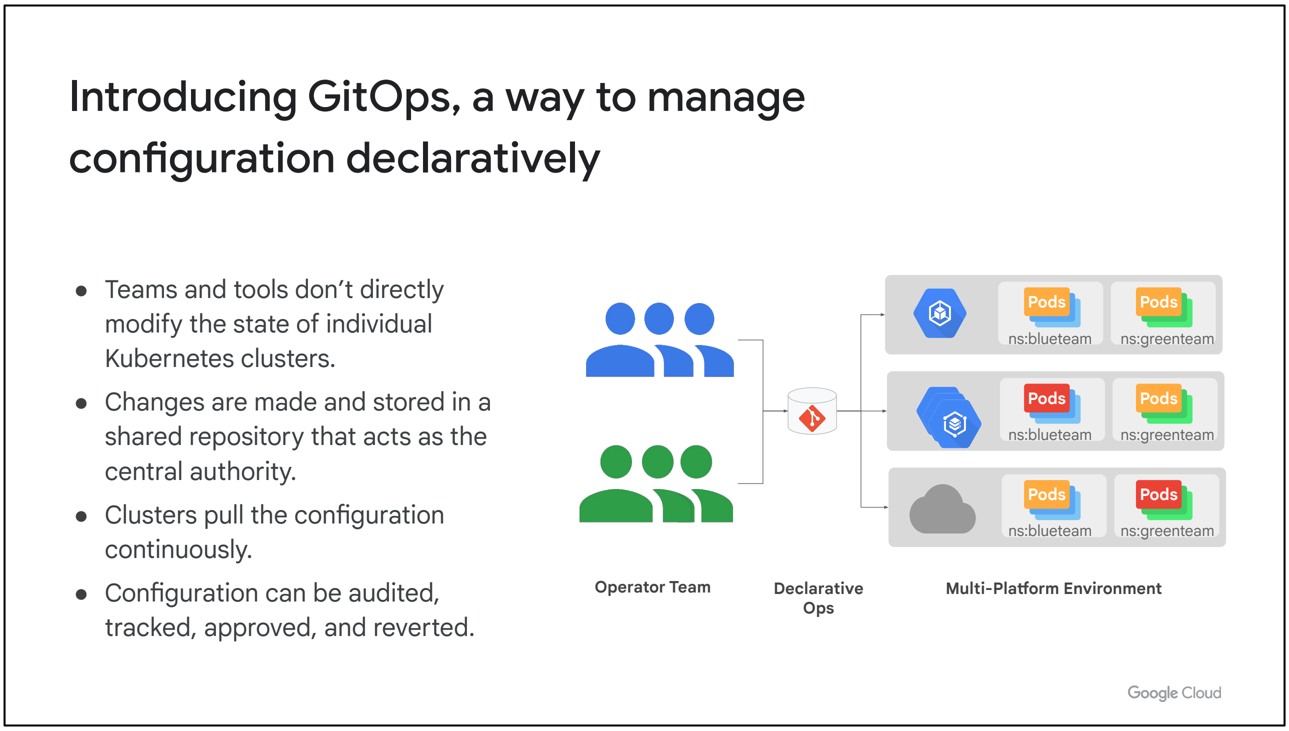
Google Cloud

Kubernetes manages resources declaratively, but changes are applied imperatively.

Imagine that the blue team modifies the state of an individual Kubernetes cluster, but they don’t store the changes in a centralized repository.

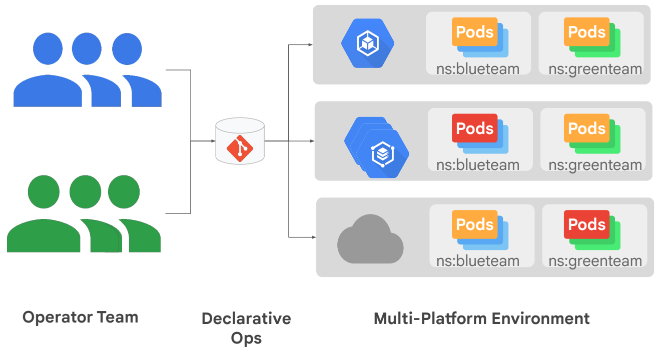
Further changes in the future by the green teams might override the blue team’s changes.

After a while, the state in which the cluster is expected to be differs from the actual cluster state, and it’s hard to point out what changes need to happen for the cluster to reach a desired state.



Introducing GitOps, a way to manage configuration declaratively

- Teams and tools don’t directly modify the state of individual Kubernetes clusters.
- Changes are made and stored in a shared repository that acts as the central authority.
- Clusters pull the configuration continuously.
- Configuration can be audited, tracked, approved, and reverted.



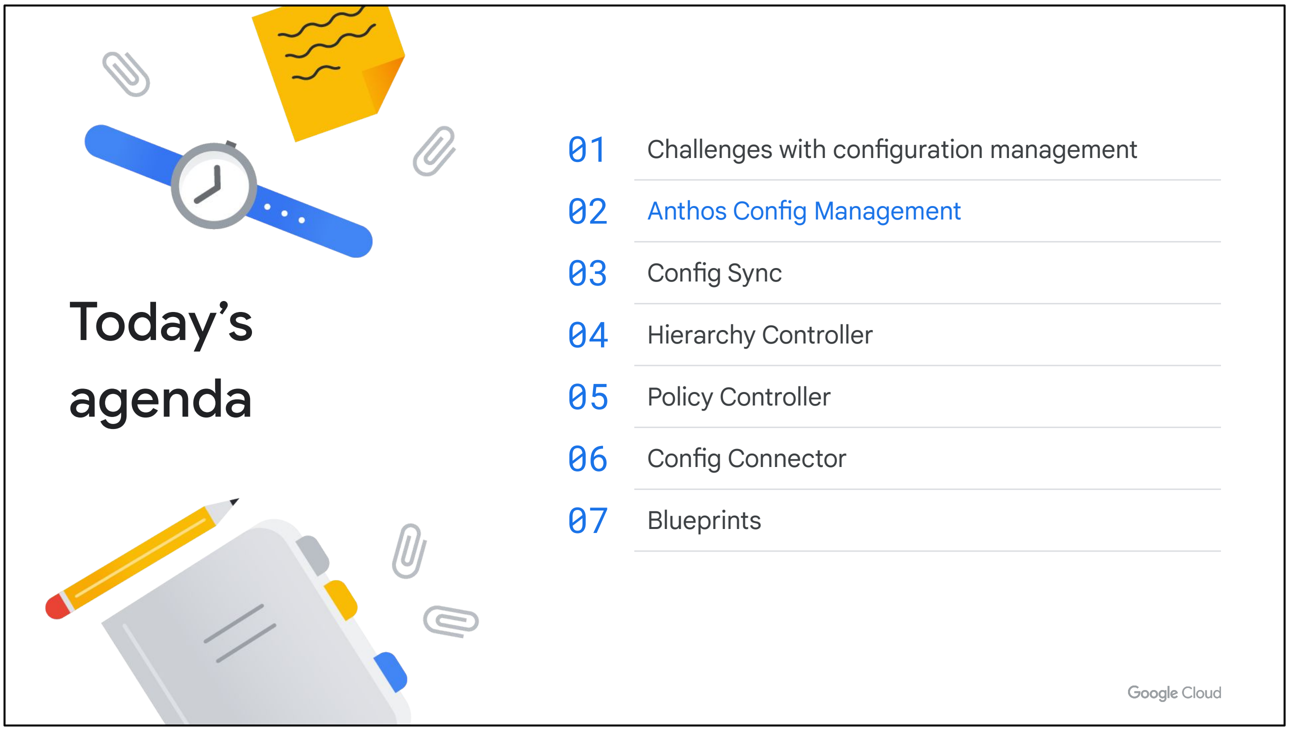
Google Cloud

To solve this problem, we are introducing GitOps, a way to manage configuration declaratively.

The idea is that instead of having teams or tools modifying directly the state of individual Kubernetes clusters, changes are made and stored in a shared repository that acts as the central point of truth.

Clusters pull the configuration continuously.

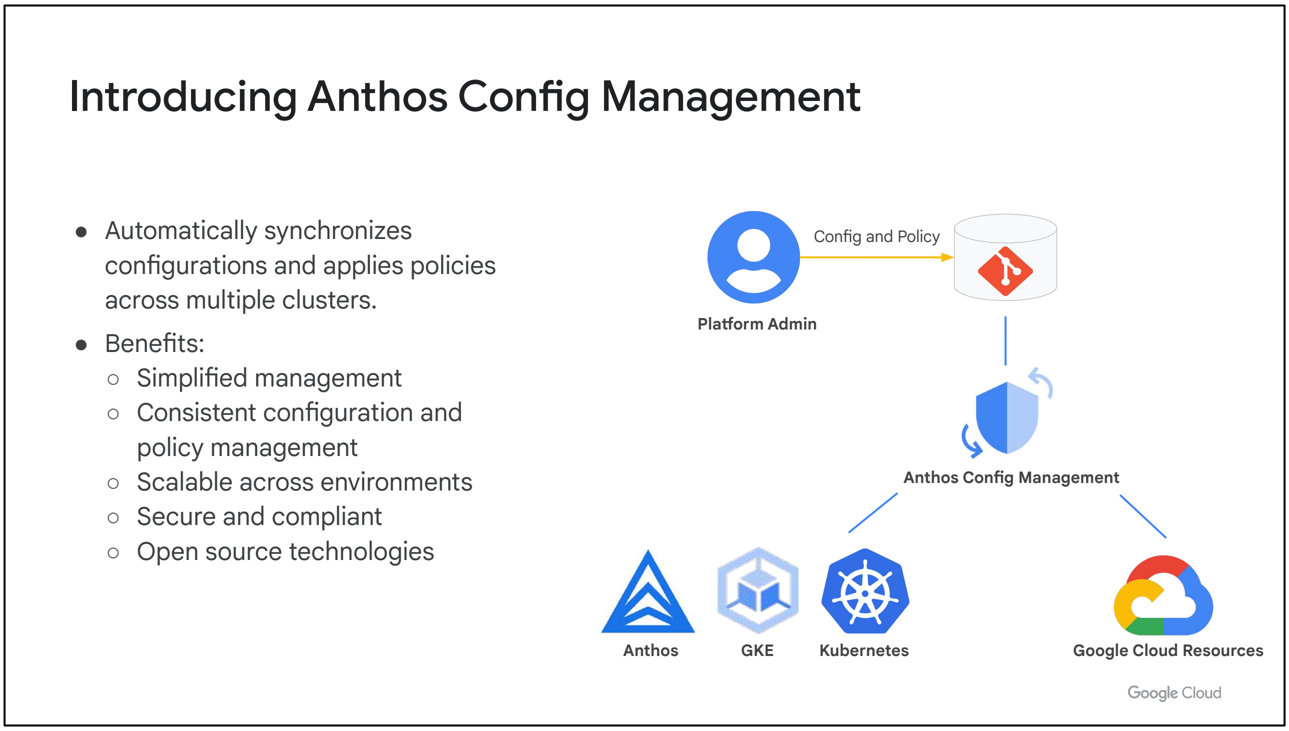
The benefit is that configuration can be audited, tracked, approved, and reverted, so that there is a higher visibility and control on what happens on a cluster.



Today's agenda

Google Cloud

Google offers Anthos Config Management, a tool that implements the GitOps methodology.



Introducing Anthos Config Management

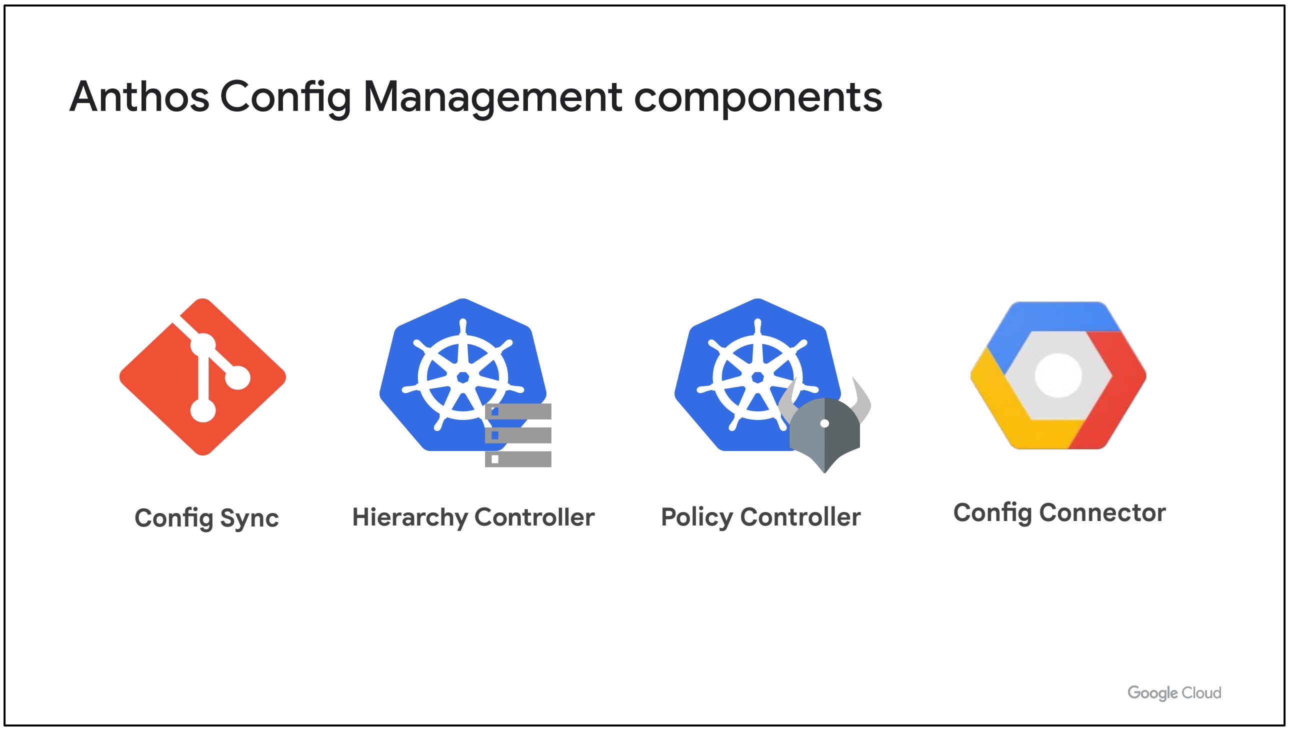
- Automatically synchronizes configurations and applies policies across multiple clusters.
- Benefits:
    - Simplified management
    - Consistent configuration and policy management
    - Scalable across environments
    - Secure and compliant
    - Open source technologies

Platform administrators can now push configurations and policies to a centralized repository and Anthos Config Management automatically synchronizes them across clusters and cloud resources.

Platform administrators obtain the following benefits:

- **Simplify management**: You can handle configurations and policies on multi-cluster environments without having to build your own toolchain from scratch.
- **Consistent configurations and policy management**: Anthos Config Management provides an auditable and version-controlled system that manages the configuration of your organization's clusters.
- **Scalable across environments**: Anthos Config Management centralizes the configuration and governance across environments, creating a scalable, automated, and reliable method for managing complex modern systems in production.
- **Secure and compliant**: With Anthos Config Management, platform administrators can reduce security risks. You can define a fully-customized set of policies and ensure that the policies are consistently applied across environments. Anthos Config Management also continuously monitors environments to ensure their desired configuration is in place and no violations

- of governance controls are present.
- **Open-source technologies**: Anthos Config Management is based on Kubernetes and cloud-native, open-source tools and projects, including Open Policy Agent Gatekeeper, which we cover in a couple of slides.



Anthos Config Management components

Config Sync

Hierarchy Controller

Policy Controller

Config Connector

Google Cloud

Anthos Config Management is composed of four main technologies that make it possible:

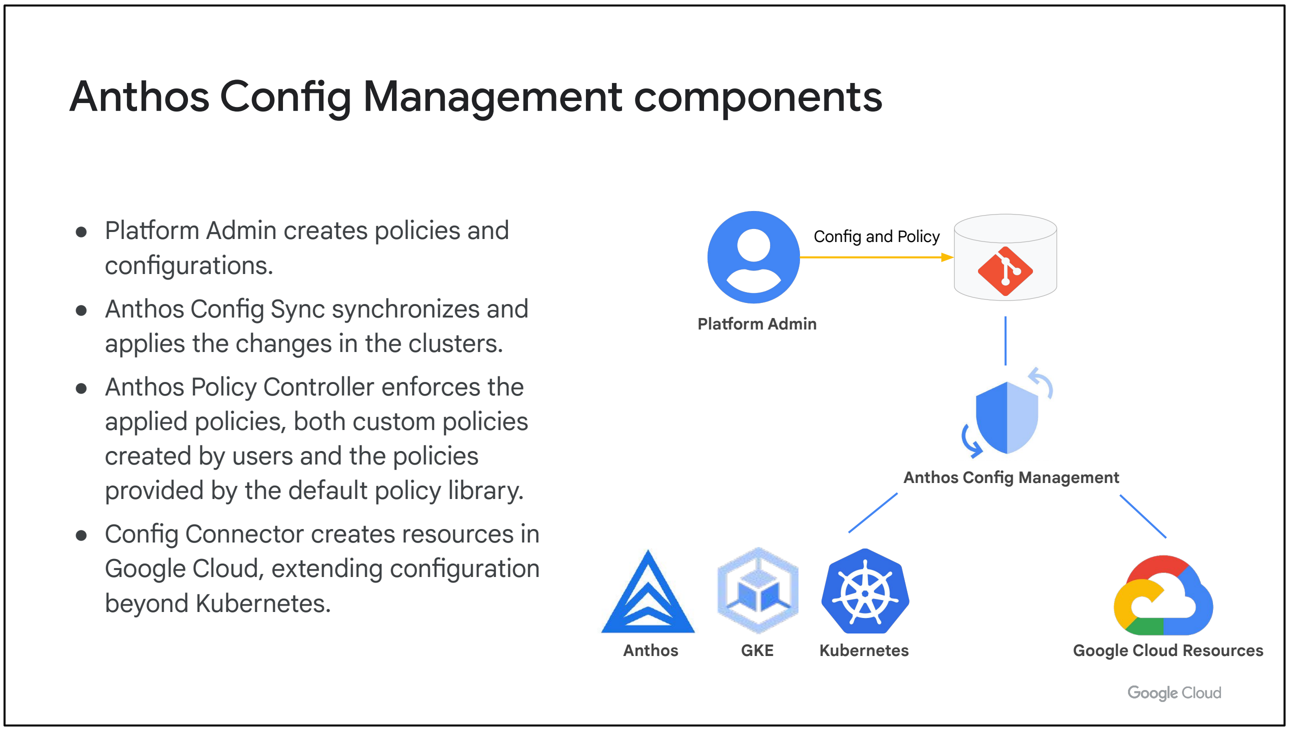
- Config Sync continuously reconciles your clusters to a central set of configurations that are stored in one or more Git repositories. This GitOps methodology lets you apply configuration consistently across clusters and environments with an auditable, transactional, and version-controlled deployment process.

- Hierarchy Controller is a CRD that extends Kubernetes namespaces so that you can better organize your configurations and policies in a hierarchical way that mimics the structure of your organization, similar to how you would organize resources in different projects and folders inside a Google Cloud Organization.

- Policy Controller enables the enforcement of fully programmable policies that represent constraints on the desired state. These policies act as "guardrails" and prevent configurations from violating security and compliance controls. You can use these policies to actively block non-compliant API requests, or simply to audit the configuration of your clusters and report violations. Policy Controller is built from the Open Policy Agent Gatekeeper project and comes with a full library of pre-built policies for common security and compliance controls. In addition, by following best practices for policy management, you can also enforce guardrails when editing configs or as a pre-submit check for Config Sync.

- Config Controller is a hosted service to provision and orchestrate Anthos and

- Google Cloud resources. This component offers an API endpoint that can provision, actuate, and orchestrate Google Cloud resources as part of Anthos Config Management.You can also run it in your cluster as Config Controller.

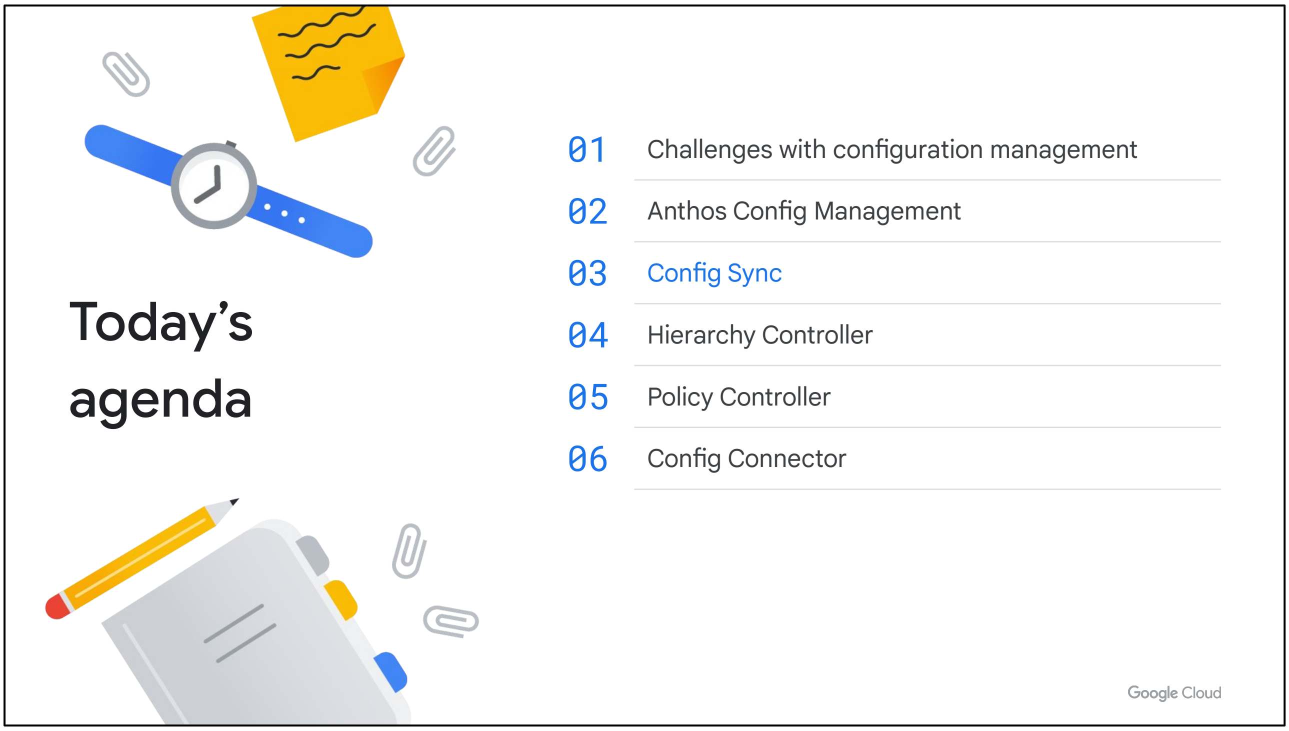


Anthos Config Management components

- Platform Admin creates policies and configurations.
- Anthos Config Sync synchronizes and applies the changes in the clusters.
- Anthos Policy Controller enforces the applied policies, both custom policies created by users and the policies provided by the default policy library.
- Config Connector creates resources in Google Cloud, extending configuration beyond Kubernetes.

The process works as follows:

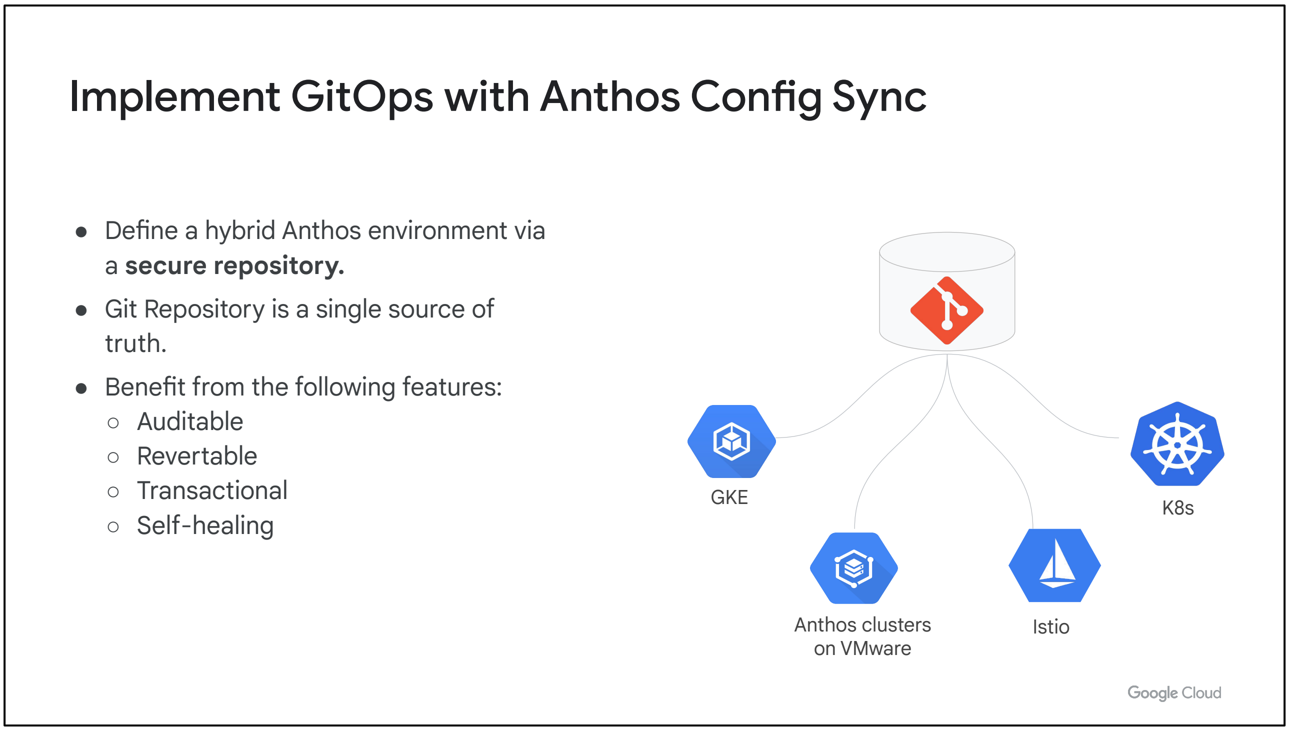
- Platform Admin creates policies and configurations and push them to a Git repository that can be hosted in the cloud or on-premises.
- Anthos Config Sync synchronizes and applies the changes in the Anthos clusters.
- Anthos Policy Controller enforces the applied policies, both custom policies created by users as well as the policies provided by the default policy library.
- Config Connector creates resources in Google Cloud, extending configuration beyond K8s.



Today's agenda

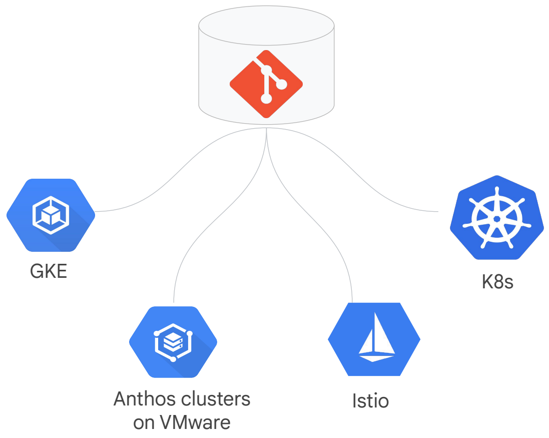
Google Cloud

Let's take a closer look at Config Sync.



Implement GitOps with Anthos Config Sync

- Define a hybrid Anthos environment via a **secure repository.**
- Git Repository is a single source of truth.
- Benefit from the following features:
  - Auditable
  - Revertable
  - Transactional
  - Self-healing

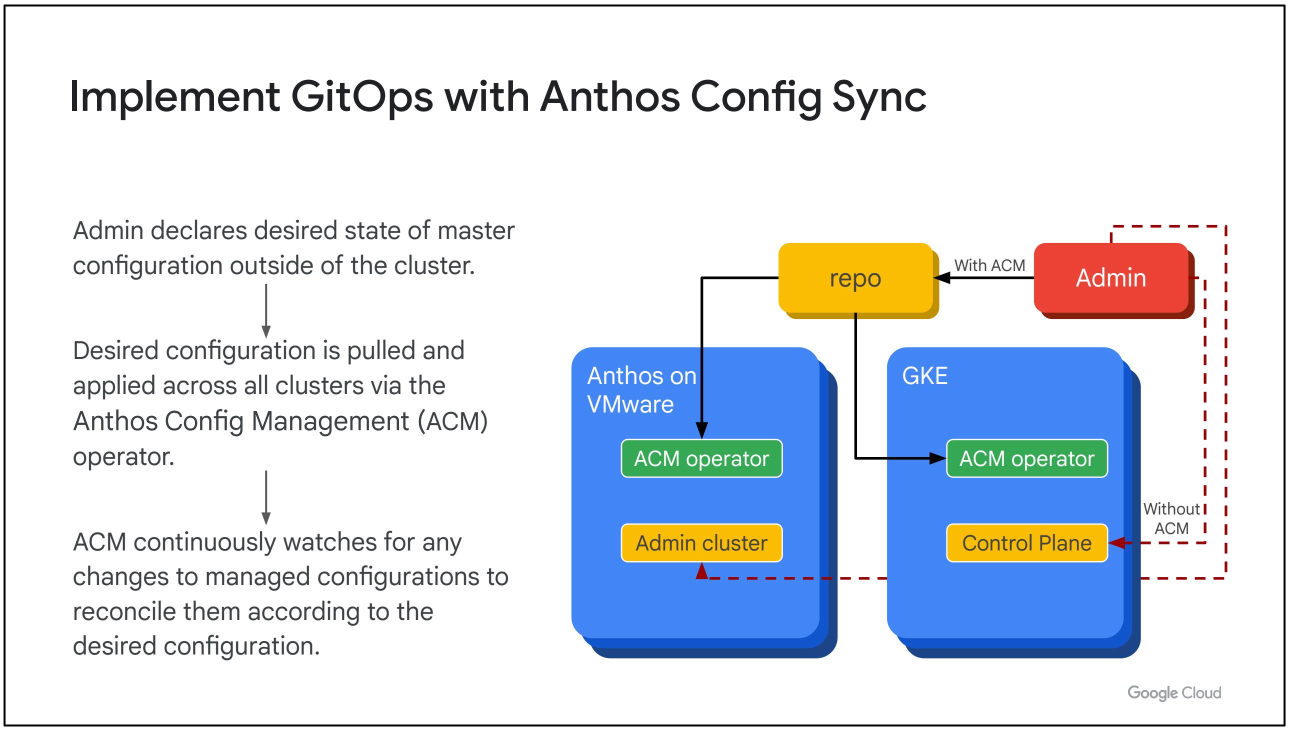


Google Cloud

Anthos Config Sync let's cluster operators and platform administrators deploy consistent configurations and policies, directly from a git repo, providing Configuration as Code.

The main benefits using Config Sync include:

- Auditability – Every change is in Git and tied to a specific commit.
- Revertability – You can roll back to the last good state, because you have a record of it.
- Transactional – Changes can be grouped and applied at once.
- Self-Healing – Config Sync guards from drift and ensures the declaration in the repo is always what you see on the cluster.

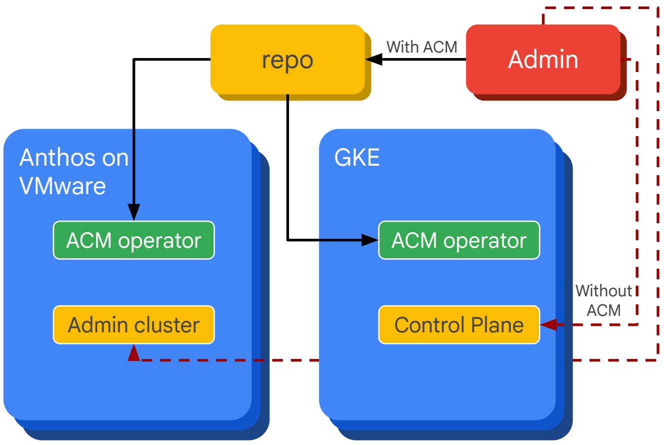


Implement GitOps with Anthos Config Sync

Admin declares desired state of master configuration outside of the cluster.

Desired configuration is pulled and applied across all clusters via the Anthos Config Management (ACM) operator.

ACM continuously watches for any changes to managed configurations to reconcile them according to the desired configuration.



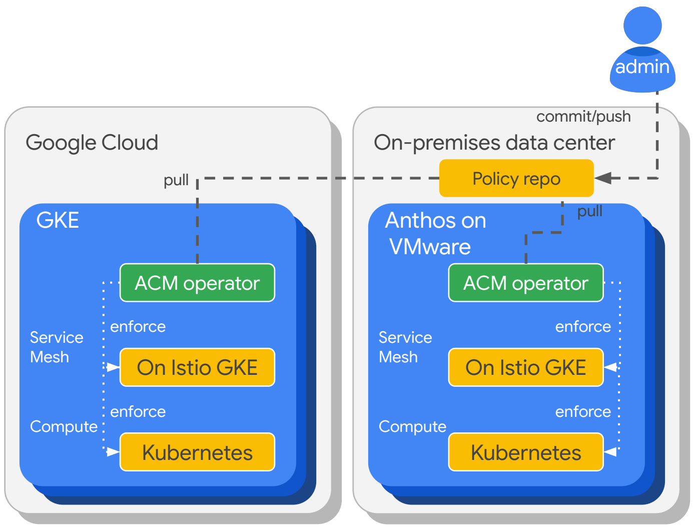
Instead of administrators running kubectl directly towards a cluster, they push their changes to Git.

Anthos clusters run an Anthos Config Management, or ACM, operator, which continuously pulls the repositories for changes.

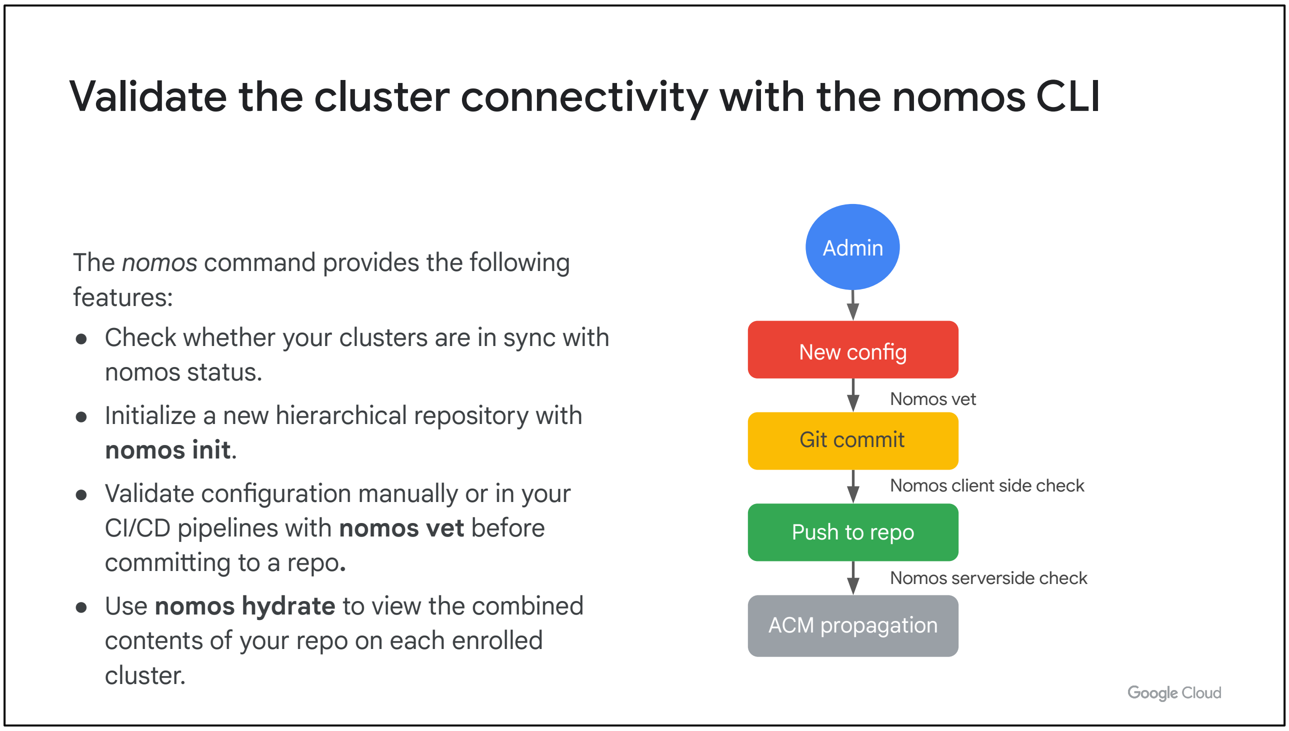
If there are changes, the operator applies them to reconcile the current state in the cluster with the desired state specified in the Git repository.

Implement GitOps with Anthos Config Sync

- The ACM operator is installed in its own namespace called *config-management-system*.
- The operator enforces the configuration in the cluster in a declarative way and reverts malicious or accidental operations.



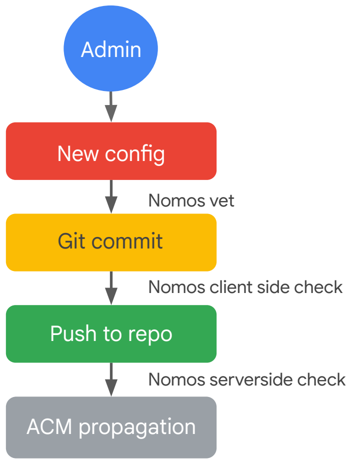
The ACM operator is implanted in your cluster in its own namespace called config-management-system. It pulls information from the Git repository which can be hosted either on-premises or in the cloud, and it forms a single source of truth. The operator enforces the configuration in the cluster in a declarative way, reverting malicious or accidental operations. This approach promotes enforced consistency across environments.



Validate the cluster connectivity with the nomos CLI

The *nomos* command provides the following features:

- Check whether your clusters are in sync with nomos status.
- Initialize a new hierarchical repository with **nomos init**.
- Validate configuration manually or in your CI/CD pipelines with **nomos vet** before committing to a repo**.**
- Use **nomos hydrate** to view the combined contents of your repo on each enrolled cluster.



Google Cloud

To validate the cluster connectivity and status, as well as the syntax of your configuration, Anthos Config Management provides a CLI tool called **nomos**.

Following the tradition of using Greek words in the Kubernetes world, "Nomos" is Greek for "law".

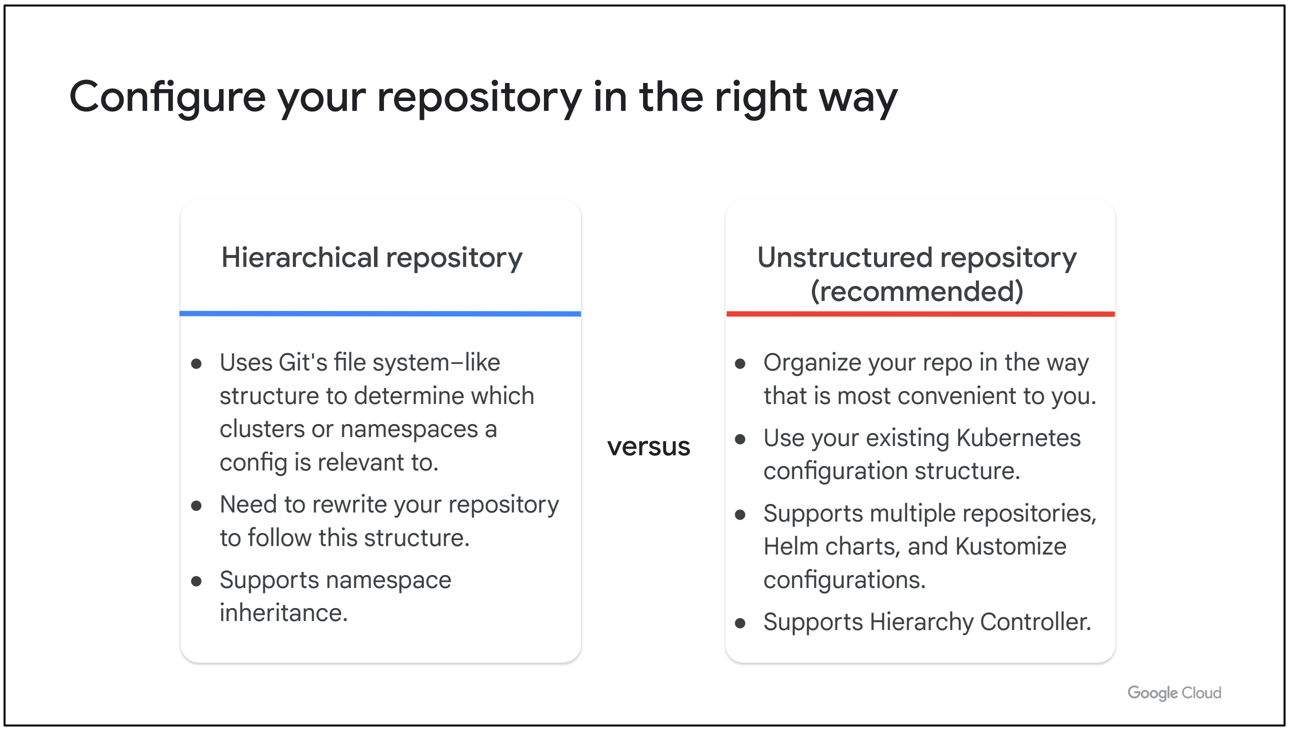
Note, that before using nomos, the Config Sync operator must be installed in your cluster by applying a Kubernetes CRD.

Once the operator is running, you can view whether your clusters are in sync by running nomos status. Specifically, you can see whether your clusters are accessible and the latest configuration has been applied.

To perform checks on a local repository, which has been structured hierarchically, you must initialize it with **nomos init.** If the repository is unstructured, you don't need to initialize it. We see different repository structures later.

Before you commit a config to the repo, use the **nomos vet** command to check the syntax and validity of the configs in your repo. Config Sync also applies those checks preventing you from running incorrect configurations.

Finally, you can use the **nomos hydrate** command to view the combined contents of your repo on each enrolled cluster. This can also be used to convert a hierarchical repo to one or more unstructured repos.



Configure your repository in the right way

Hierarchical repository

- Uses Git's file system–like structure to determine which clusters or namespaces a config is relevant to.
- Need to rewrite your repository to follow this structure.
- Supports namespace inheritance.

versus

Unstructured repository (recommended)

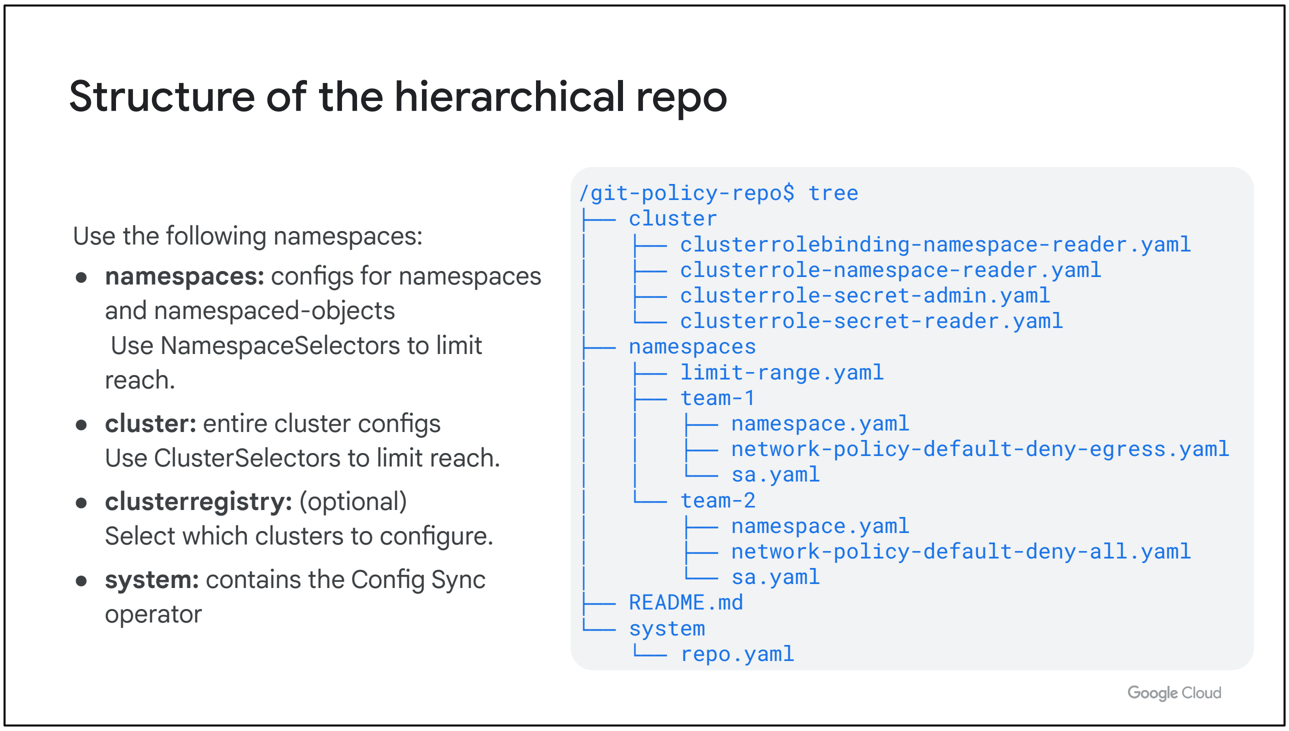
- Organize your repo in the way that is most convenient to you.
- Use your existing Kubernetes configuration structure.
- Supports multiple repositories, Helm charts, and Kustomize configurations.
- Supports Hierarchy Controller.

Google Cloud

Let’s take a look at the two options for structuring your repository.

Hierarchical repositories expect a structure defined by Anthos Config Management. This file system structure determines the clusters or namespaces a config is relevant to. If you adopt ACM on an existing cluster, you will need to rewrite your repository to follow this hierarchical repository structure. It’s called hierarchical because it supports namespace inheritance.

Unstructured repositories is a newer, more flexible solution that allows you to organize your repository in the way that is most convenient to you. You can use your existing Kubernetes configuration structure, as well as using multiple repositories, Helm charts, and Kustomize configurations. You can install the Hierarchy Controller to enable namespace inheritance.



Structure of the hierarchical repo

Use the following namespaces:

- **namespaces:** configs for namespaces and namespaced-objects
  Use NamespaceSelectors to limit reach.
- **cluster:** entire cluster configs
  Use ClusterSelectors to limit reach.
- **clusterregistry:** (optional)
  Select which clusters to configure.
- **system:** contains the Config Sync operator

```
/git-policy-repo$ tree
├── cluster
│   ├── clusterrolebinding-namespace-reader.yaml
│   ├── clusterrole-namespace-reader.yaml
│   ├── clusterrole-secret-admin.yaml
│   └── clusterrole-secret-reader.yaml
├── namespaces
│   ├── limit-range.yaml
│   ├── team-1
│   │   ├── namespace.yaml
│   │   ├── network-policy-default-deny-egress.yaml
│   │   └── sa.yaml
│   └── team-2
│       ├── namespace.yaml
│       ├── network-policy-default-deny-all.yaml
│       └── sa.yaml
├── README.md
└── system
    └── repo.yaml
```

Google Cloud

The following directory structure demonstrates how to use a Config Sync hierarchical repo to configure a Kubernetes cluster shared by two different teams, team-1 and team-2.

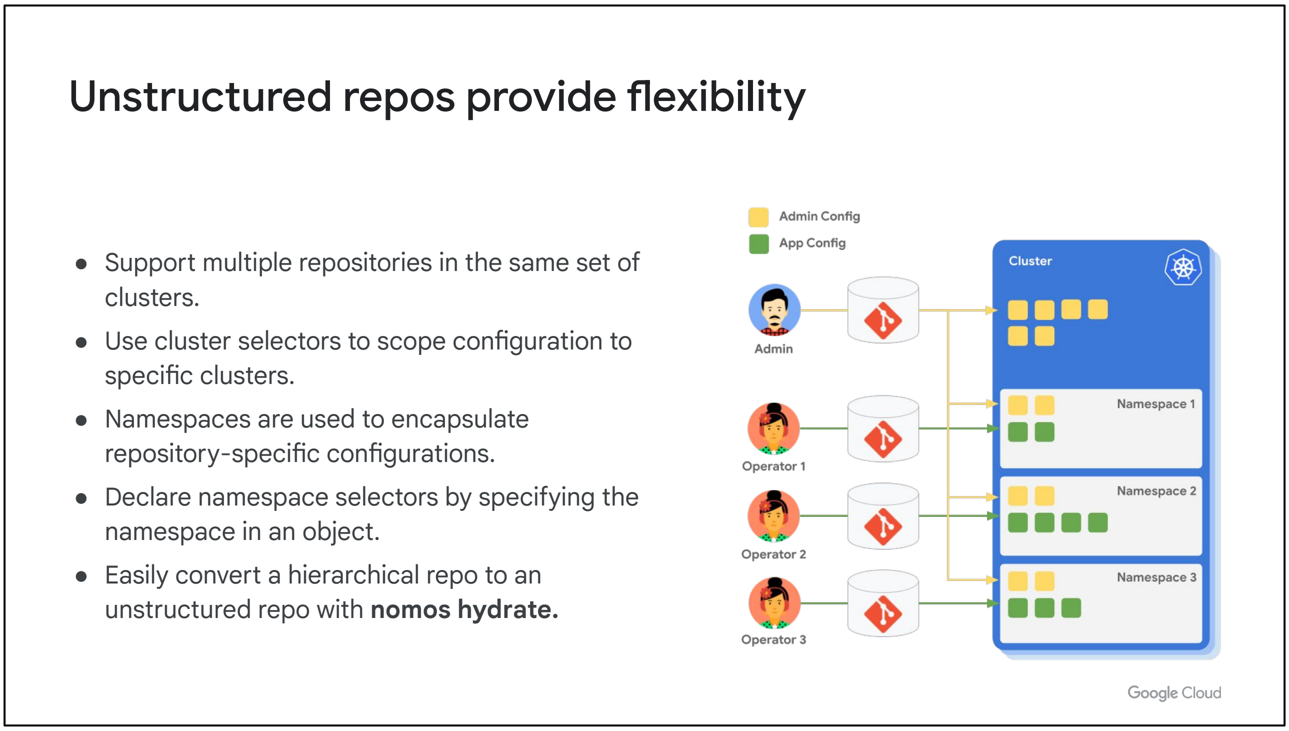
- Each team has its own Kubernetes namespace, Kubernetes service account, resource quotas, network policies, rolebindings.
- The cluster admin sets up a policy in namespaces/limit-range.yaml to constrain resource allocations (to pods or containers) in both namespaces.
- The cluster admin also sets up ClusterRoles and ClusterRoleBinidngs.

A valid Config Sync hierarchical repo must include three subdirectories: cluster/, namespaces/, and system/.

The cluster/ directory contains configs that apply to entire clusters (such as ClusterRole, ClusterRoleBinding), rather than to namespaces.

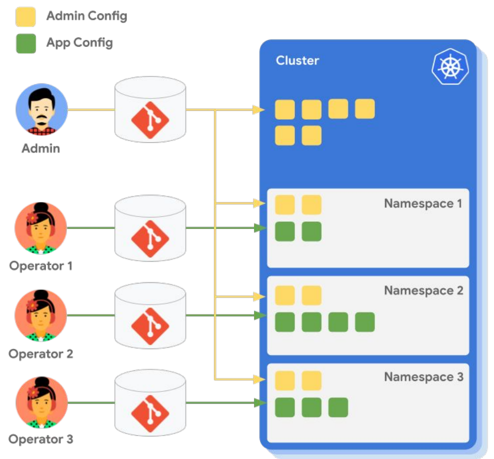
The namespaces/ directory contains configs for the namespace objects and the namespace-scoped objects. Each subdirectory under namespaces/ includes the configs for a namespace object and all the namespace-scoped objects under the namespace. The name of a subdirectory should be the same as the name of the namespace object. Namespace-scoped objects which need to be created in each namespace, can be put directly under namespaces/ (for example, namespaces/limit-range.yaml).

The system/ directory contains configs for Config Management Operator.



Unstructured repos provide flexibility

- Support multiple repositories in the same set of clusters.
- Use cluster selectors to scope configuration to specific clusters.
- Namespaces are used to encapsulate repository-specific configurations.
- Declare namespace selectors by specifying the namespace in an object.
- Easily convert a hierarchical repo to an unstructured repo with **nomos hydrate.**



Google Cloud

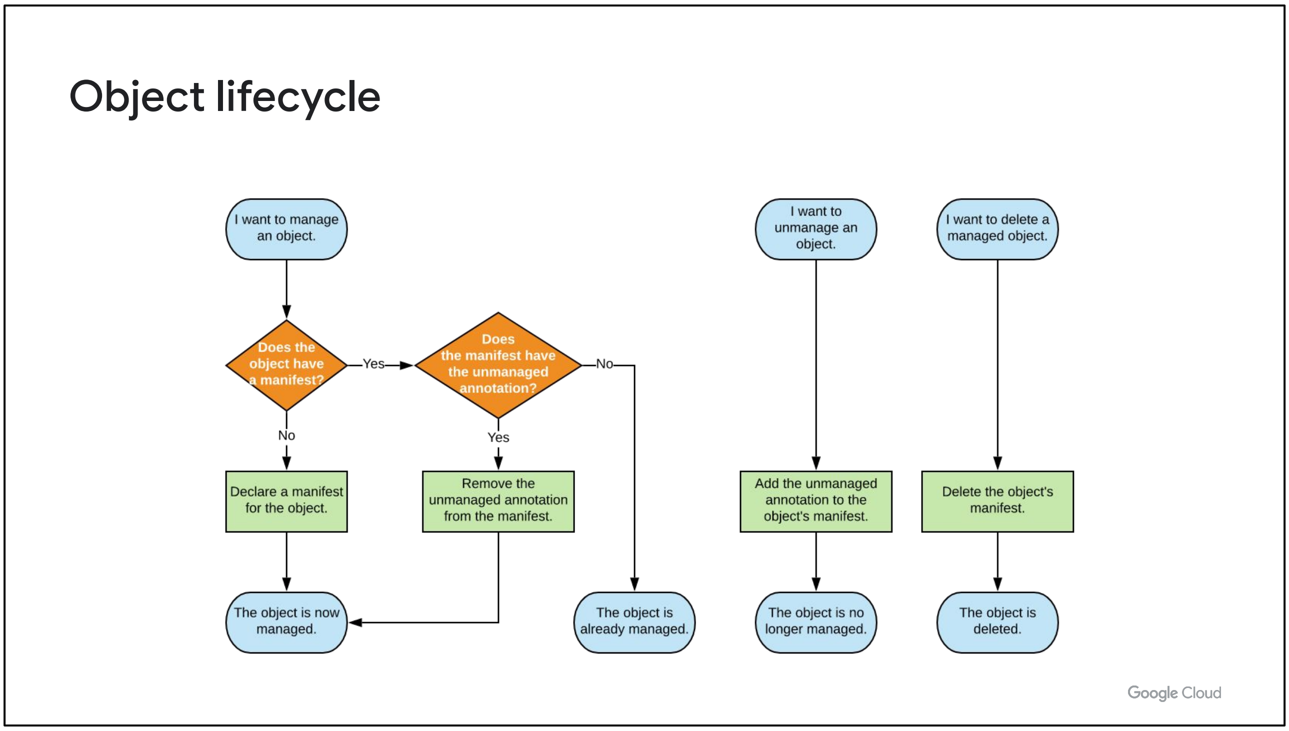
Alternatively, you can use Config Sync with unstructured repos for maximum flexibility.

Unstructured repos support multiple repositories in the same set of clusters, so that different teams can have their configuration in their own repositories.

Operators can then use cluster selectors to scope configuration to specific clusters and namespaces are used to encapsulate repository-specific configurations.

You can easily declare namespace selectors by specifying the namespace in an object.

If you started using a hierarchical repo, you can easily convert to an unstructured repo with **nomos hydrate**.



Object lifecycle

I want to manage an object.
Does the object have a manifest?
Yes
No
Does the manifest have the unmanaged annotation?
No
Yes
Declare a manifest for the object.
Remove the unmanaged annotation from the manifest.
The object is now managed.
The object is already managed.
I want to unmanage an object.
Add the unmanaged annotation to the object's manifest.
The object is no longer managed.
I want to delete a managed object.
Delete the object's manifest.
The object is deleted.
Google Cloud

In order to manage an object with Config Sync in a particular repo, the following steps are followed.

First, does the object have a config in the repo?

- **No**: Create a config for the object. Anthos Config Management sets the annotation configmanagement.gke.io/managed=enabled and begins managing the object.
- **Yes**: Does the config set the annotation configmanagement.gke.io/managed=disabled?
  - **No**: The object is already managed.
  - **Yes**: Change the config to set the annotation configmanagement.gke.io/managed=disabled.

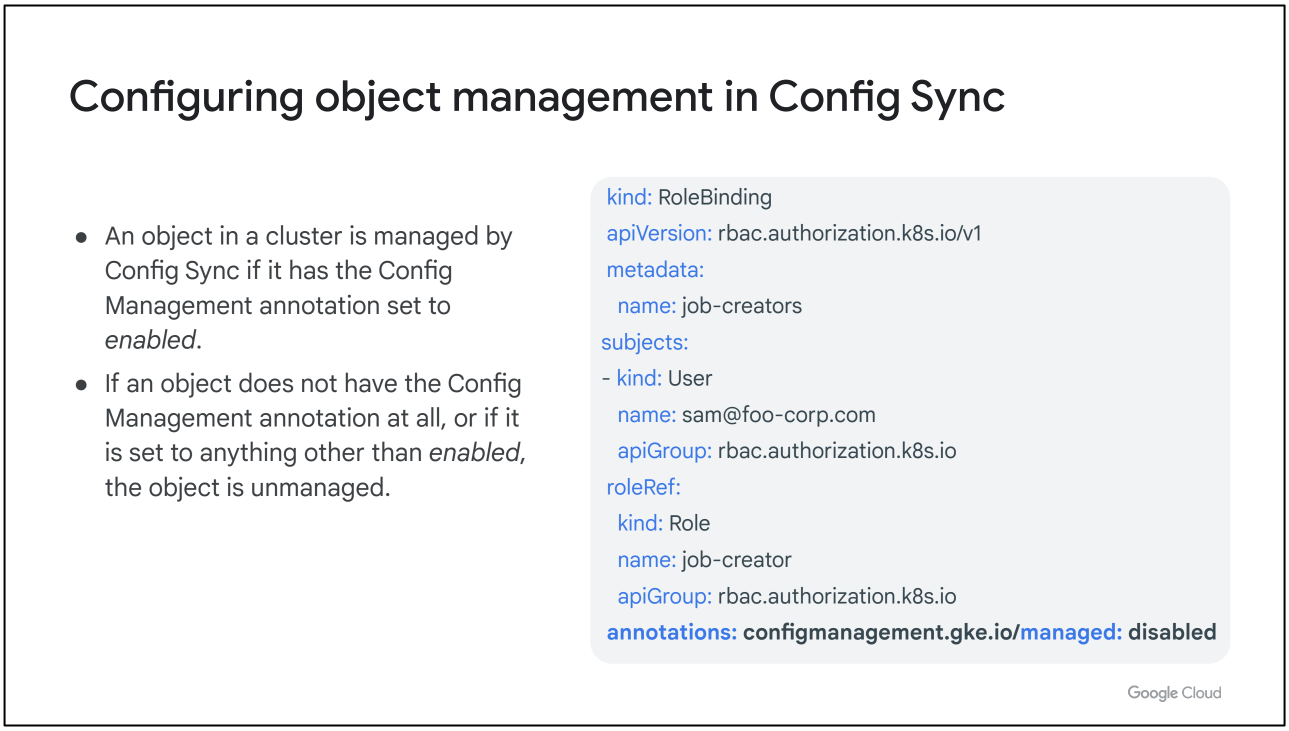
Second, If you want to stop managing an object but not delete it:

- Edit the config for the object in the repo and set the annotation configmanagement.gke.io/managed=disabled. When the config change is detected, Anthos Config Management stops managing the object.

Third, If you want to stop managing an object and delete it:

- Delete the object's config from the repo. When you delete a config for an object that was previously managed, the object is deleted from all clusters or namespaces the config applies to. Anthos Config Management uses a

- declarative model to apply configuration changes to your clusters by reading your desired configuration from files in your repo. If you attempt to apply the annotation manually (either using the kubectl command or the Kubernetes API), the manual change is quickly and automatically overridden by the contents of your repo.



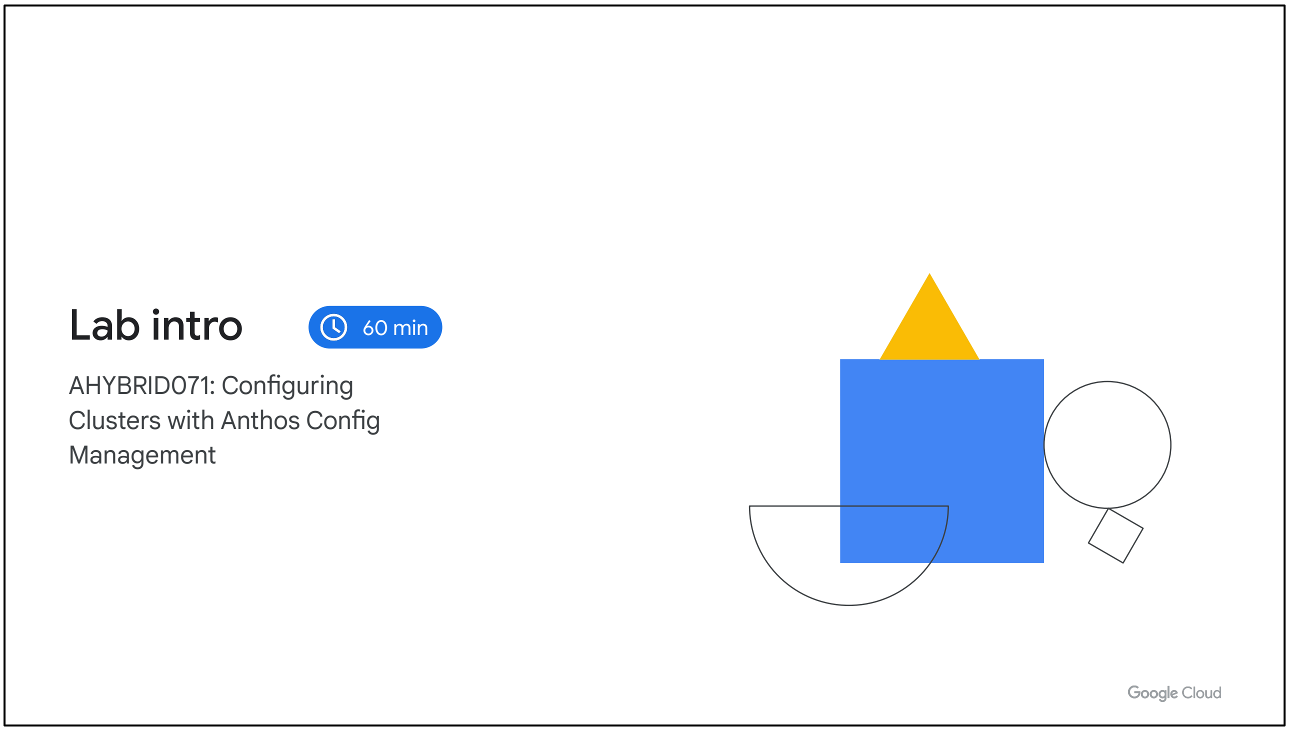
Configuring object management in Config Sync

- An object in a cluster is managed by Config Sync if it has the Config Management annotation set to *enabled*.
- If an object does not have the Config Management annotation at all, or if it is set to anything other than *enabled*, the object is unmanaged.

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: job-creators
subjects:
- kind: User
  name: sam@foo-corp.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: job-creator
  apiGroup: rbac.authorization.k8s.io
annotations: configmanagement.gke.io/managed: disabled
```

Google Cloud

When Anthos Config Management manages a cluster object, it watches the object and the set of configs in the repo that affect the object, and ensures that they are in sync.
That is defined by the annotation configmanagement.gke.io/managed:. In this case, tracking for this object is disabled, so it won't be synced. To manage it, change the annotation value to enabled.



Lab intro

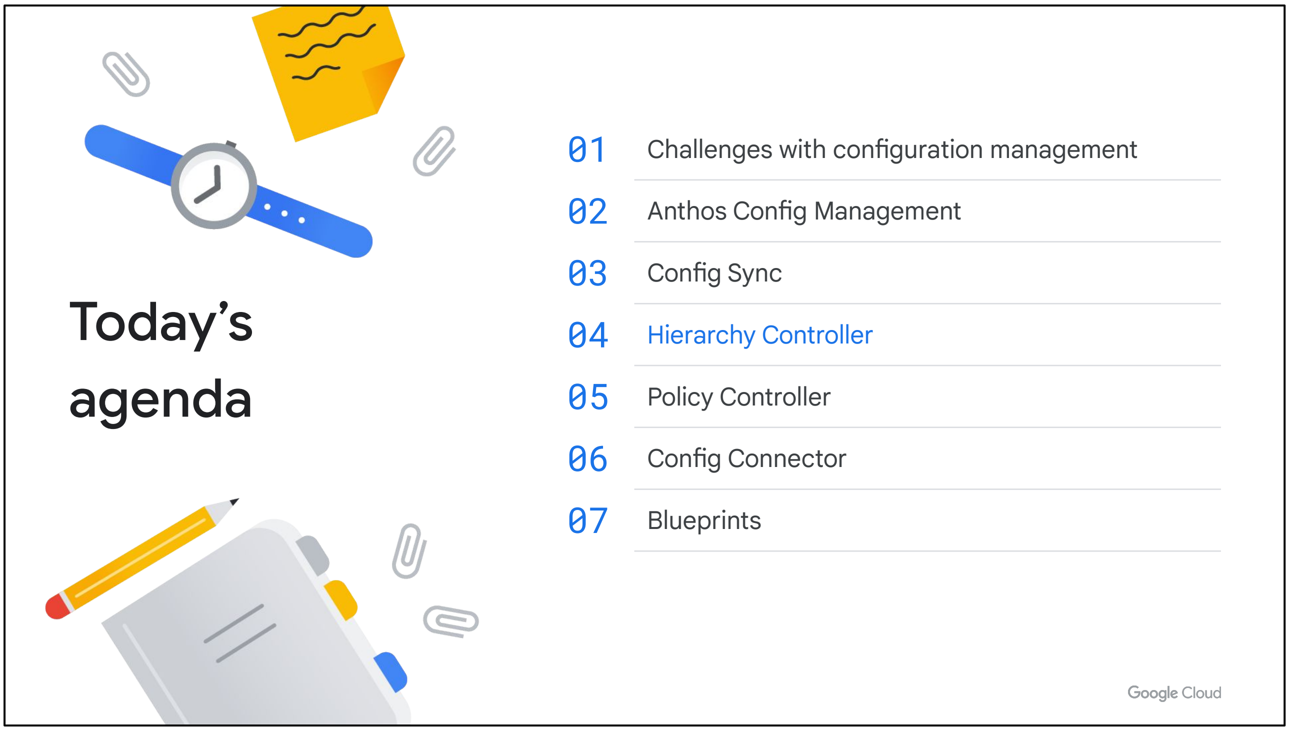
60 min

AHYBRID071: Configuring Clusters with Anthos Config Management

Google Cloud

Objectives:

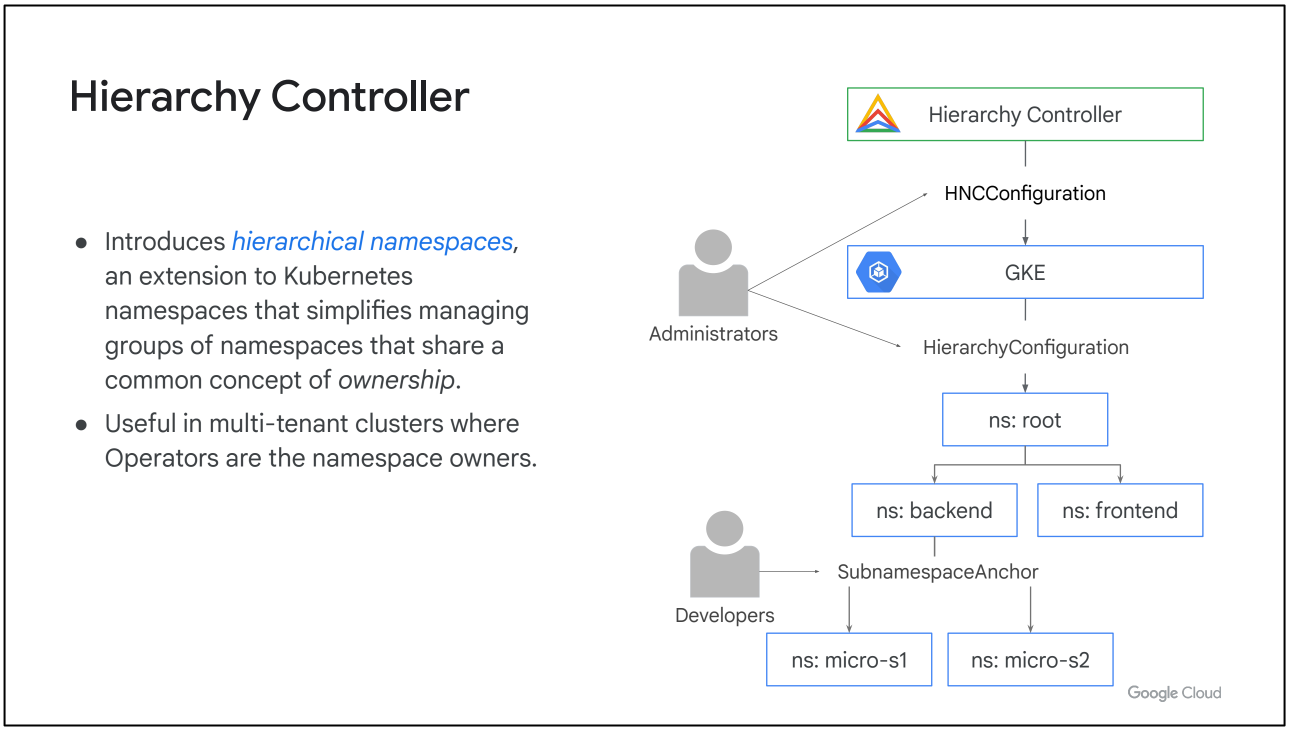
- Install the Config Management Operator and the nomos command-line tool.
- Set up your config repo in Cloud Source Repositories.
- Connect your GKE clusters to the config repo.
- Examine the configs in your clusters and repo.
- Filter application of configs by namespace.
- Review automated drift management.
- Update a config in the repo.



Today's agenda

Google Cloud

Let's discuss Hierarchy Controller next.



Hierarchy Controller

- Introduces *hierarchical namespaces*, an extension to Kubernetes namespaces that simplifies managing groups of namespaces that share a common concept of *ownership*.
- Useful in multi-tenant clusters where Operators are the namespace owners.

Hierarchy Controller introduces the concept of *hierarchical namespaces*, which are extensions to Kubernetes namespaces that make it easy to manage groups of namespaces that share a common concept of *ownership*. They are especially useful in clusters that are shared by multiple *teams*, but the owners do not need to be people. For example, you might want to make an operator an owner of a set of namespaces.

To support multi-tenant use cases, hierarchical namespaces have two key behaviors in addition to regular Kubernetes namespaces:

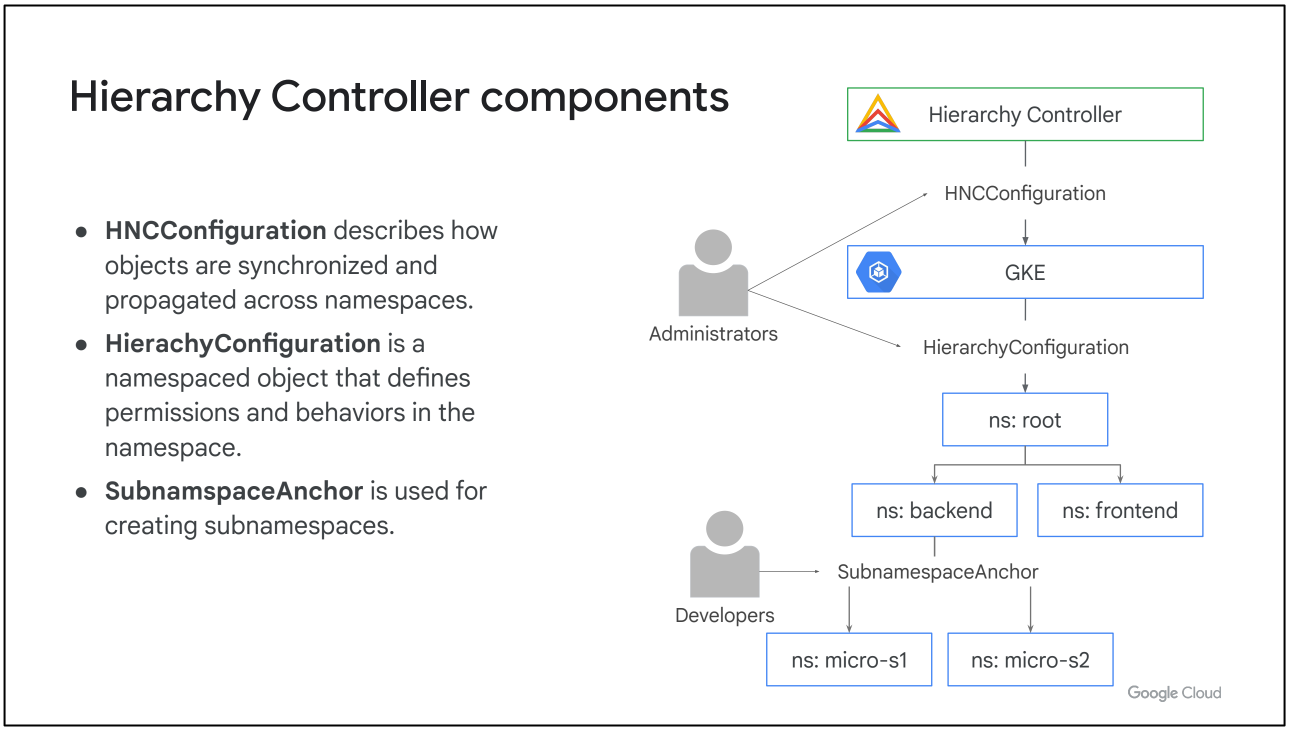
- **Delegated namespace creation.** Hierarchy Controller introduces the concept of a *subnamespace*, which can be created by users under an existing namespace even if that user does not have cluster-level namespace privileges.

- For example, you may define a namespace for a team in Anthos Config Management in a Git repository, but then delegate subnamespace creation to the team. This lets them create their own subnamespaces underneath the team-level namespace that you defined for them, without making any modifications in Git.

- **Policy enforcement.** Just as Anthos Config Management can propagate policy objects from abstract namespace directories to Kubernetes namespaces, Hierarchy Controller can further propagate these objects to subnamespaces. For example, doing this ensures that the policies that are applied to a team's *root* namespace are also applied to their subnamespaces.

- However, there are some differences between how Hierarchy Controller and Anthos Config Management propagate policies. Anthos Config Management copies *all* objects in abstract namespace directories to their descendant Kubernetes namespaces. By contrast, Hierarchy Controller copies *only* RBAC roles and role bindings by default, although it can be configured to propagate any other type of Kubernetes object.

Hierarchy Controller is integrated into Anthos Config Management version 1.4.1 and later. Hierarchy Controller is based on the Hierarchical Namespace Controller, or HNC, an open-source project.



Hierarchy Controller components

- **HNCConfiguration** describes how objects are synchronized and propagated across namespaces.
- **HierachyConfiguration** is a namespaced object that defines permissions and behaviors in the namespace.
- **SubnamspaceAnchor** is used for creating subnamespaces.

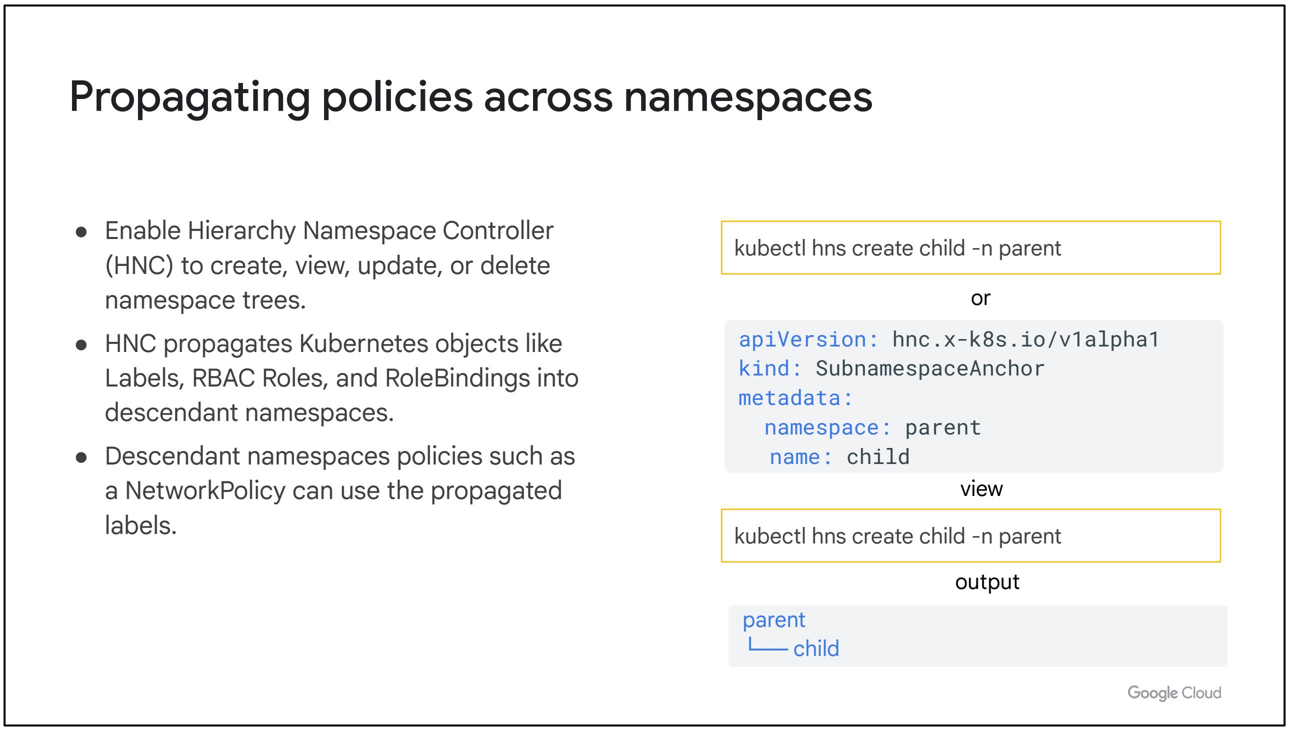
The Hierarchy Controller has several components:

- **HNCConfiguration** is a single non-namespaced config object that defines the synchronization and propagation behaviour of the entire cluster. You can define whether namespaces should propagate policies to descendants, remove all propagated policies, or ignore new changes to stop propagation. Ideally, only cluster admins are responsible for configuring the HNCConfiguration since it may contain information about any namespace in the cluster.

- **HierarchyConfiguration** is a namespaced object, which defines permissions and behaviours in the namespace. Only admins for the cluster or for as specific namespace should have permissions to modify them.

- **SubnamespaceAnchor** is used to create subnamespaces. Access to *create* or *read* these objects should be granted quite freely to users to have permission to use other objects in a given namespace, since this allows them to use hierarchical namespaces to organize their objects.

It is important to note that just because a user *created* a subnamespace, that does not make them an *administrator* of that subnamespace. That requires someone to explicitly grant them the update permission for the HierarchyConfiguration object in that namespace. As a result, an unprivileged user who creates a subnamespace generally can't delete it as well, since this would require them to set the allowCascadingDeletion property of the child namespace.



Propagating policies across namespaces

- Enable Hierarchy Namespace Controller (HNC) to create, view, update, or delete namespace trees.
- HNC propagates Kubernetes objects like Labels, RBAC Roles, and RoleBindings into descendant namespaces.
- Descendant namespaces policies such as a NetworkPolicy can use the propagated labels.

```
kubectl hns create child -n parent
```

or

```
apiVersion: hnc.x-k8s.io/v1alpha1
kind: SubnamespaceAnchor
metadata:
  namespace: parent
   name: child
```

view

```
kubectl hns create child -n parent
```

output

```
parent
└── child
```

Google Cloud

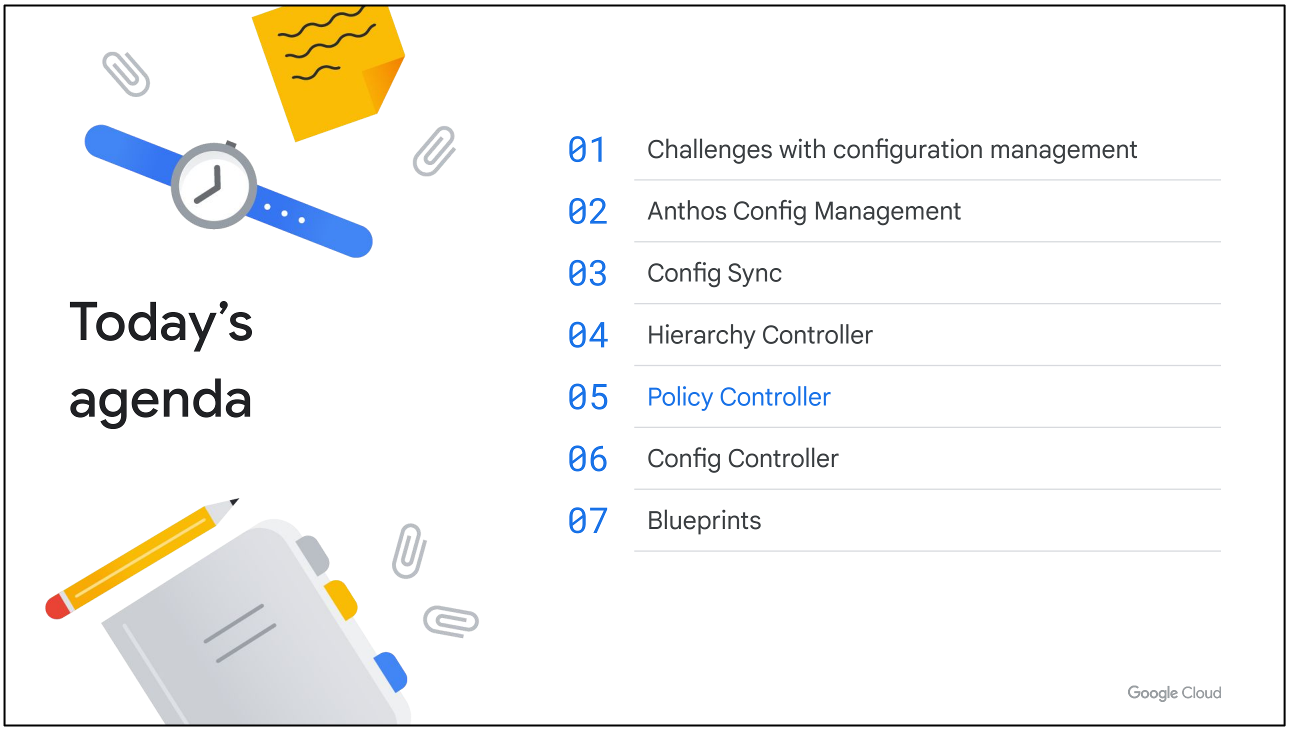
Once the Hierarchy Namespace Controller, or HNC, has been enabled, you can create, view, update, or delete namespace trees by interacting with the hns CRDs with kubectl.

By default, HNC propagates Kubernetes objects such as Labels, RBAC Role, and RoleBinding. If you create objects of these kinds in a parent namespace, it will automatically be copied into any descendant namespaces as well. However, you cannot modify these propagated copies directly and HNC's admission controllers will attempt to stop you from editing them.

Similarly, if you try to create an object in a parent ancestor with the same name as an object in one of its descendants, HNC will stop you from doing so, because this would result in the objects in the descendants being silently overwritten. HNC will also prevent you from changing the parent of a namespace if this would result in objects being overwritten.

However, if you bypass these admission controllers—for example, by updating objects while HNC is being upgraded—HNC *will* overwrite conflicting objects in descendant namespaces. This is to ensure that if you are able to successfully create a policy in an ancestor namespace, you can be confident that it will be uniformly applied to all descendant namespaces.

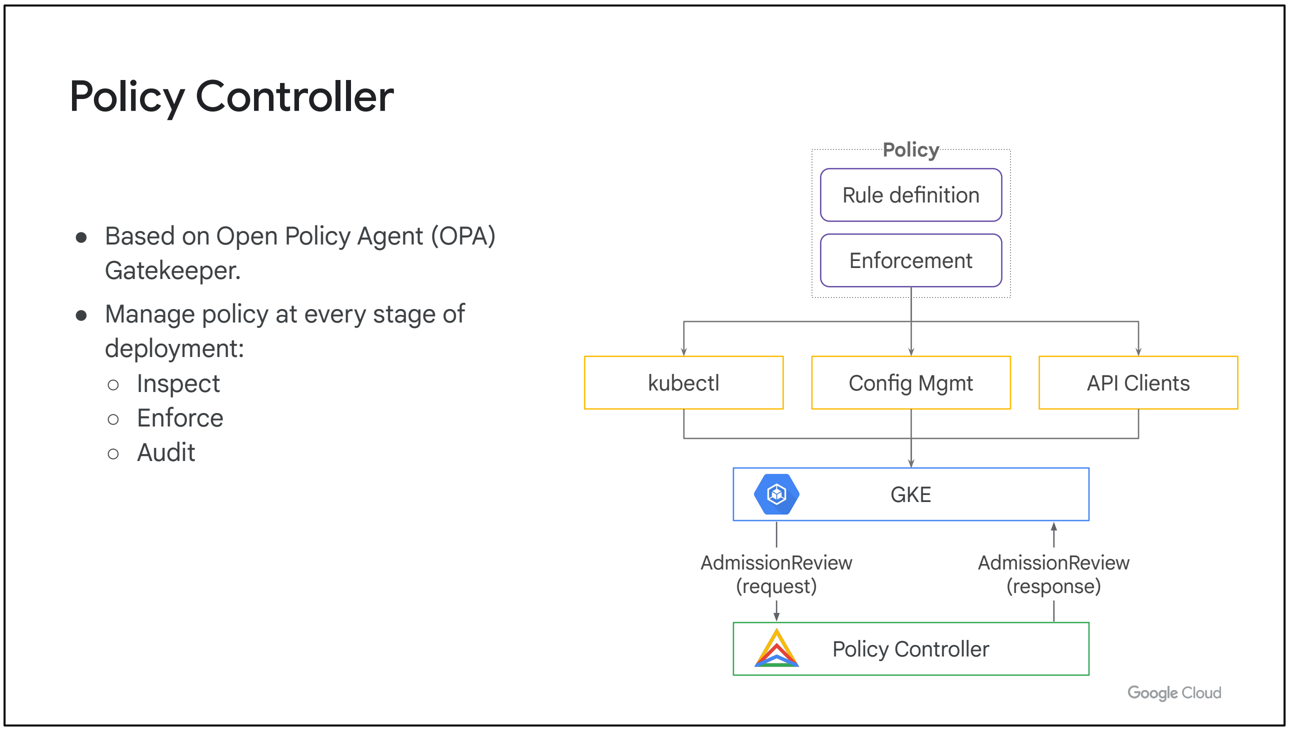
Occasionally, objects might fail to be propagated to descendant namespaces for a variety of reasons; e.g., HNC itself might not have sufficient RBAC permissions. To understand why an object is not being propagated to a namespace, use kubectl hns describe <ns>, where <ns> is either the source (ancestor) or destination (descendant) namespace.



Today's agenda

Google Cloud

Let's take a look at the Policy Controller.



Policy Controller

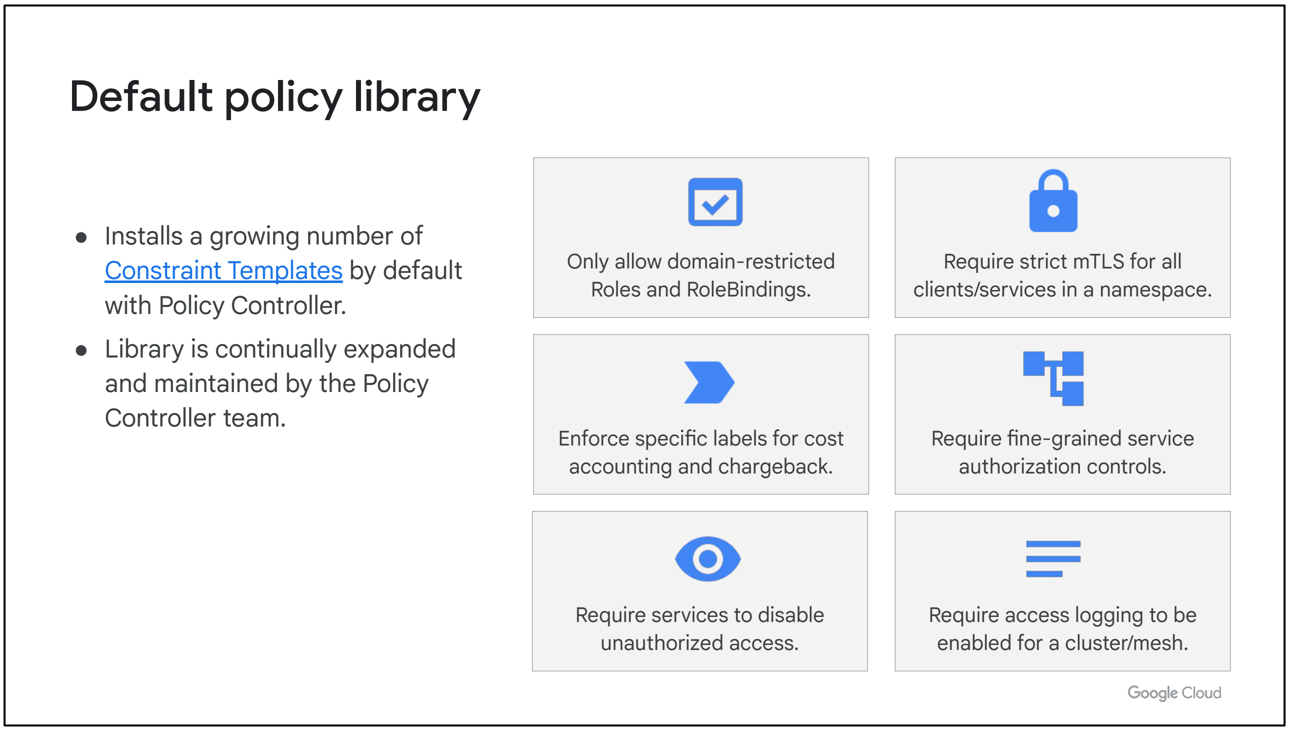
- Based on Open Policy Agent (OPA) Gatekeeper.
- Manage policy at every stage of deployment:
  - Inspect
  - Enforce
  - Audit

Policy
Rule definition
Enforcement
kubectl
Config Mgmt
API Clients
GKE
AdmissionReview (request)
AdmissionReview (response)
Policy Controller
Google Cloud

Policy Controller is a programmable policy enforcement point, based on open source projects (OPA and Gatekeeper).

Using built-in or custom policies, you can apply security and operational controls directly in the Kubernetes API, ensuring that no matter what the client is, they are subject to your corporate security and compliance controls.

- Policy Controller can run as a pre-deploy analyzer, inspecting every pull request to a config repo.
- It also runs as an admission controller, actively blocking any API request that is out-of-bounds.
- Finally, it can scan the full set of Kubernetes objects, reporting audit results on any existing configuration that violates your policies.



Default policy library

- Installs a growing number of Constraint Templates by default with Policy Controller.
- Library is continually expanded and maintained by the Policy Controller team.

Only allow domain-restricted Roles and RoleBindings.

Require strict mTLS for all clients/services in a namespace.

Enforce specific labels for cost accounting and chargeback.

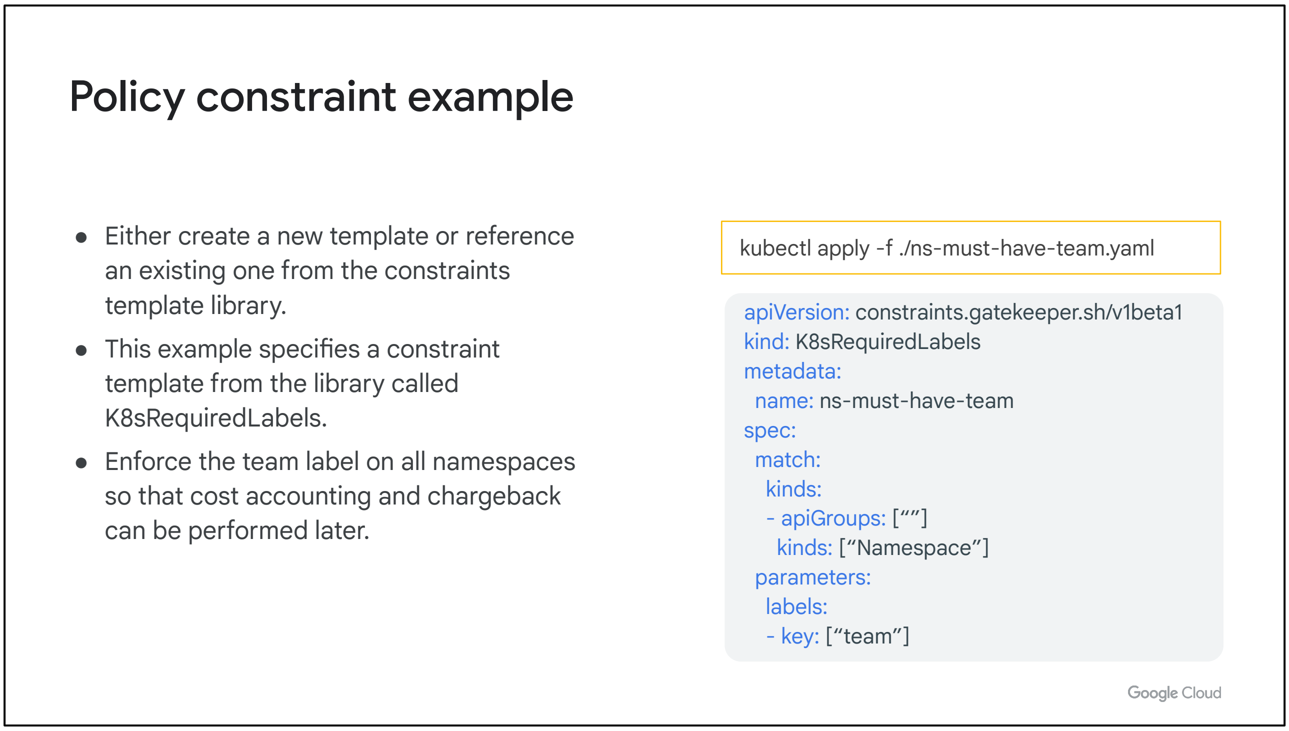
Require fine-grained service authorization controls.

Require services to disable unauthorized access.

Require access logging to be enabled for a cluster/mesh.

Google Cloud

The built-in library comes with a number of out-of-the-box policies for common use cases. Google is expanding this all the time as they find new ways of helping customers build guardrails for their Anthos environments.



Policy constraint example

- Either create a new template or reference an existing one from the constraints template library.
- This example specifies a constraint template from the library called K8sRequiredLabels.
- Enforce the team label on all namespaces so that cost accounting and chargeback can be performed later.

```
kubectl apply -f ./ns-must-have-team.yaml
```

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sRequiredLabels
metadata:
  name: ns-must-have-team
spec:
  match:
    kinds:
    - apiGroups: [“”]
      kinds: [“Namespace”]
  parameters:
    labels:
    - key: [“team”]
```

Google Cloud

Here’s an example of a policy constraint:

You define a constraint by using YAML, and you do not need to understand or write Rego, the language in which the yaml is interpreted. Instead, a constraint invokes a constraint template and provides it with parameters specific to the constraint.

If you are using a structured repo, we recommend that you create your constraints in the `cluster/` directory.

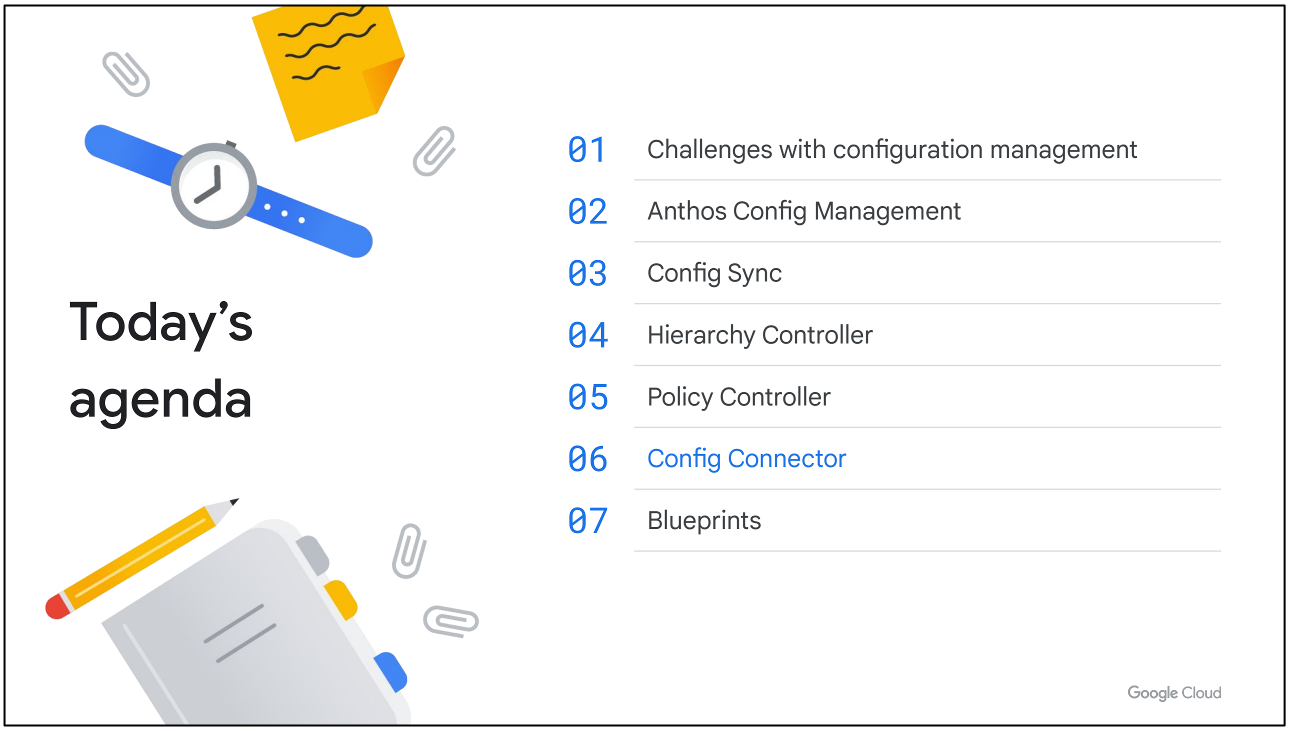
Constraints have the following fields:

- The lowercased `kind` matches the name of a constraint template.
- The `metadata.name` is the name of the constraint.
- The `match` field defines which objects the constraint applies to. All conditions specified must be matched before an object is in-scope for a constraint. `match` conditions are defined by the following sub-fields:
  - `kinds` are the kinds of resources the constraint applies to, determined by two fields: `apiGroups` is a list of Kubernetes API groups that will match and `kinds` is a list of kinds that will match. "*" matches everything. If at least one `apiGroup` and one `kind` entry match, the `kinds` condition is satisfied.
  - `namespaces` is a list of namespace names the object can belong to. The object must belong to at least one of these namespaces. Namespace resources are treated as if they belong to themselves.

    - `excludedNamespaces` is a list of namespaces that the object cannot belong to.
    - `labelSelector` is a Kubernetes label selector that the object must satisfy.
    - `namespaceSelector` is a label selector on the namespace the object belongs to. If the namespace does not satisfy the object, it will not match. Namespace resources are treated as if they belong to themselves.

- The `parameters` field defines the arguments for the constraint, based on what the constraint template expects.

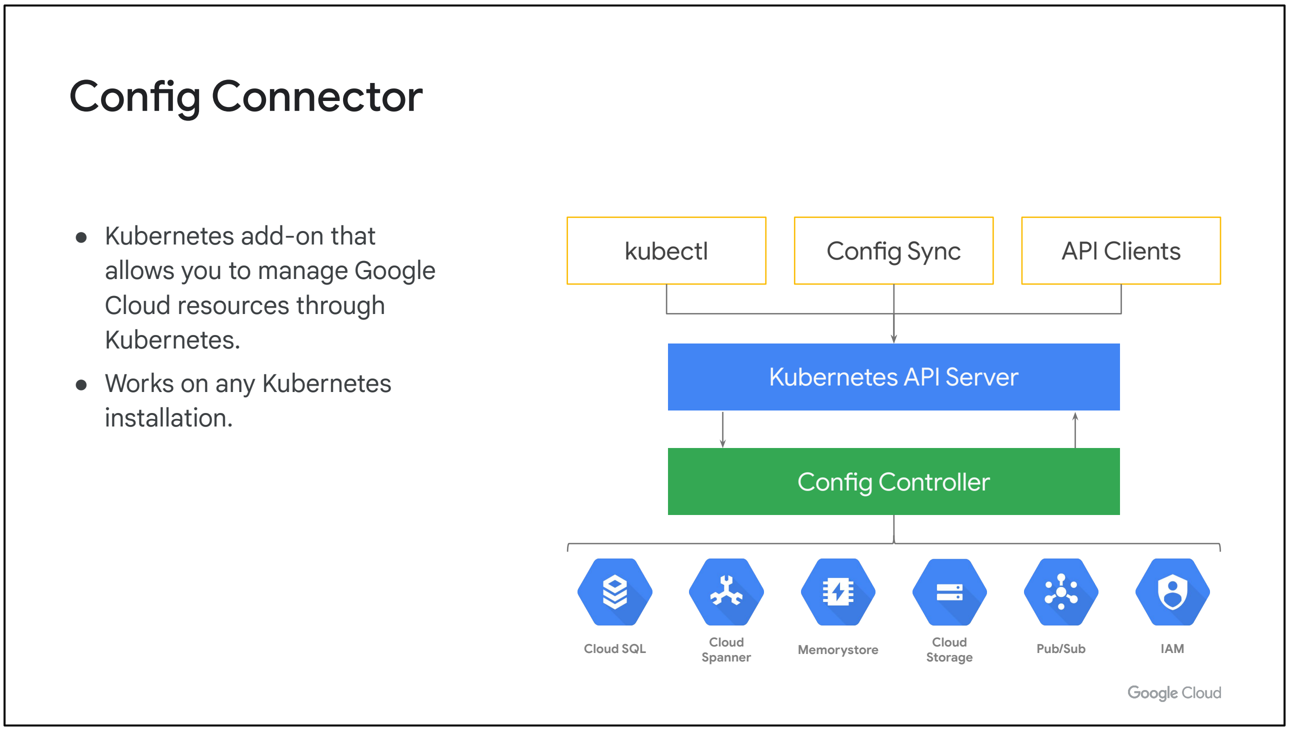
The following constraint, called `ns-must-have-team`, invokes a constraint template called `K8sRequiredLabels`, which is included in the constraint template library provided by Google. The constraint defines parameters that the constraint template uses to evaluate whether namespaces have the `team` label set to some value.



Today's agenda

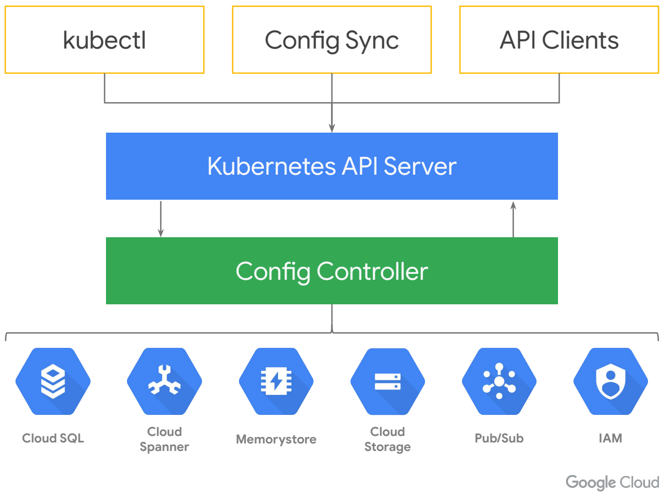
Google Cloud

Let's cover the last Anthos Config Management component, the Config Connector

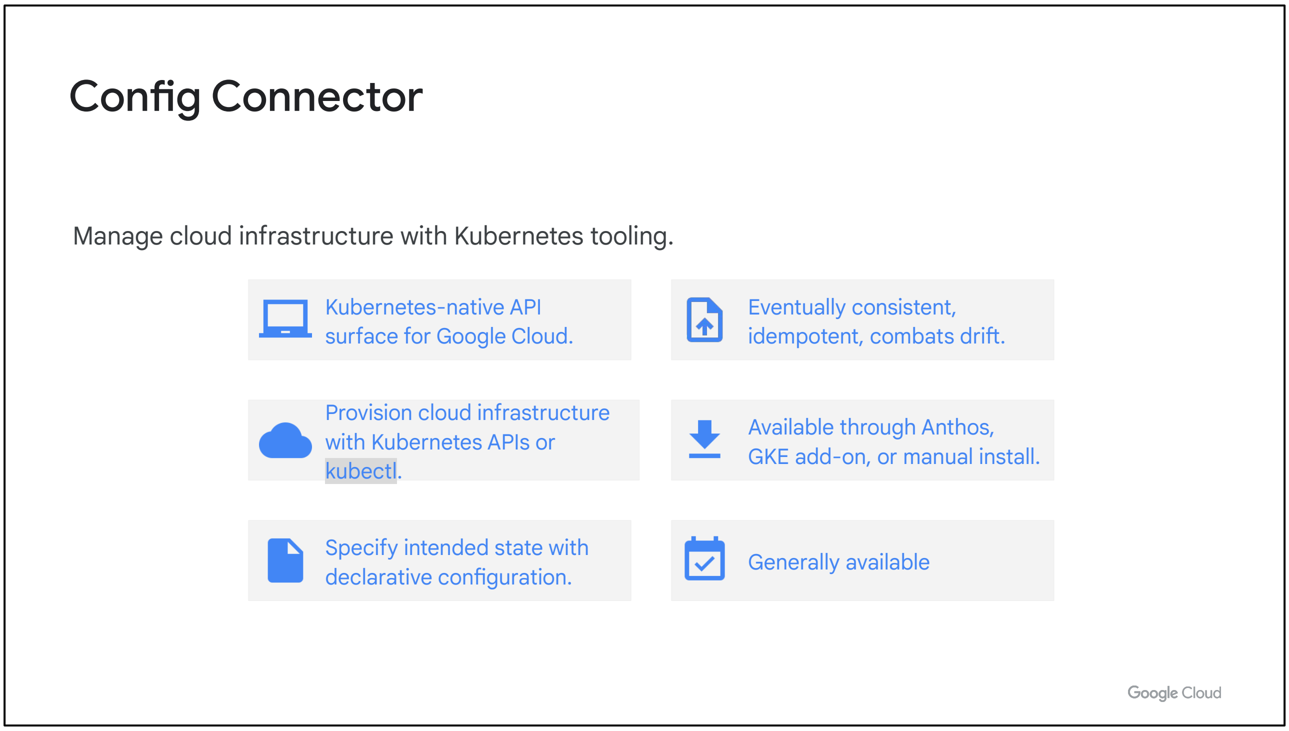


Config Connector

- Kubernetes add-on that allows you to manage Google Cloud resources through Kubernetes.
- Works on any Kubernetes installation.



Config Connector brings Google Cloud to Kubernetes, taking the services you use in cloud and wrapping them in a declarative Kubernetes API that is available to all your clients.



Config Connector

Manage cloud infrastructure with Kubernetes tooling.

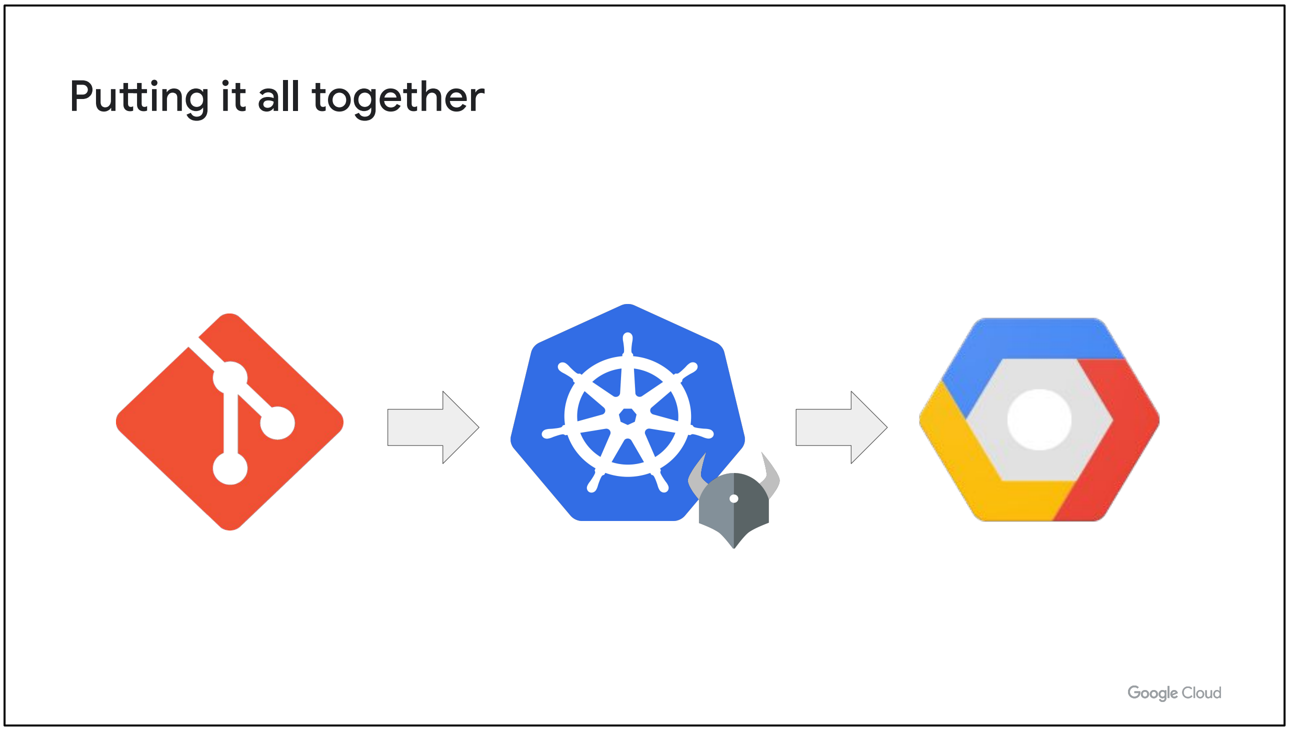
- Kubernetes-native API surface for Google Cloud.
- Eventually consistent, idempotent, combats drift.
- Provision cloud infrastructure with Kubernetes APIs or kubectl.
- Available through Anthos, GKE add-on, or manual install.
- Specify intended state with declarative configuration.
- Generally available

Google Cloud

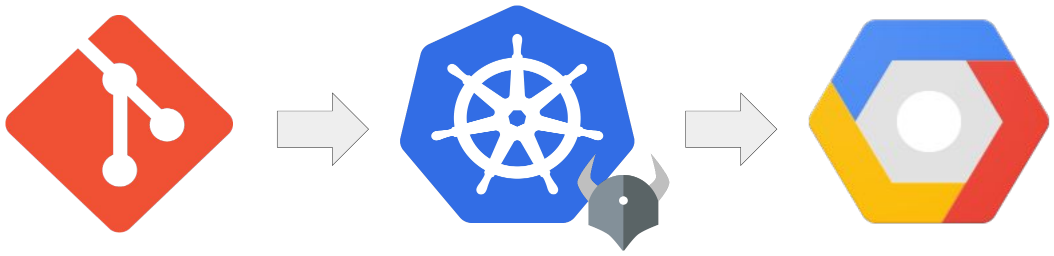
Treat Google Cloud services like K8s services, deploying your cloud-backed containers against a single API surface.

Benefit from the same GitOps mechanism we used with Config Sync to manage and combat drift in Kubernetes in a broad coverage of Google Cloud APIs.

Both Anthos and GKE customers can try it out today.



Putting it all together



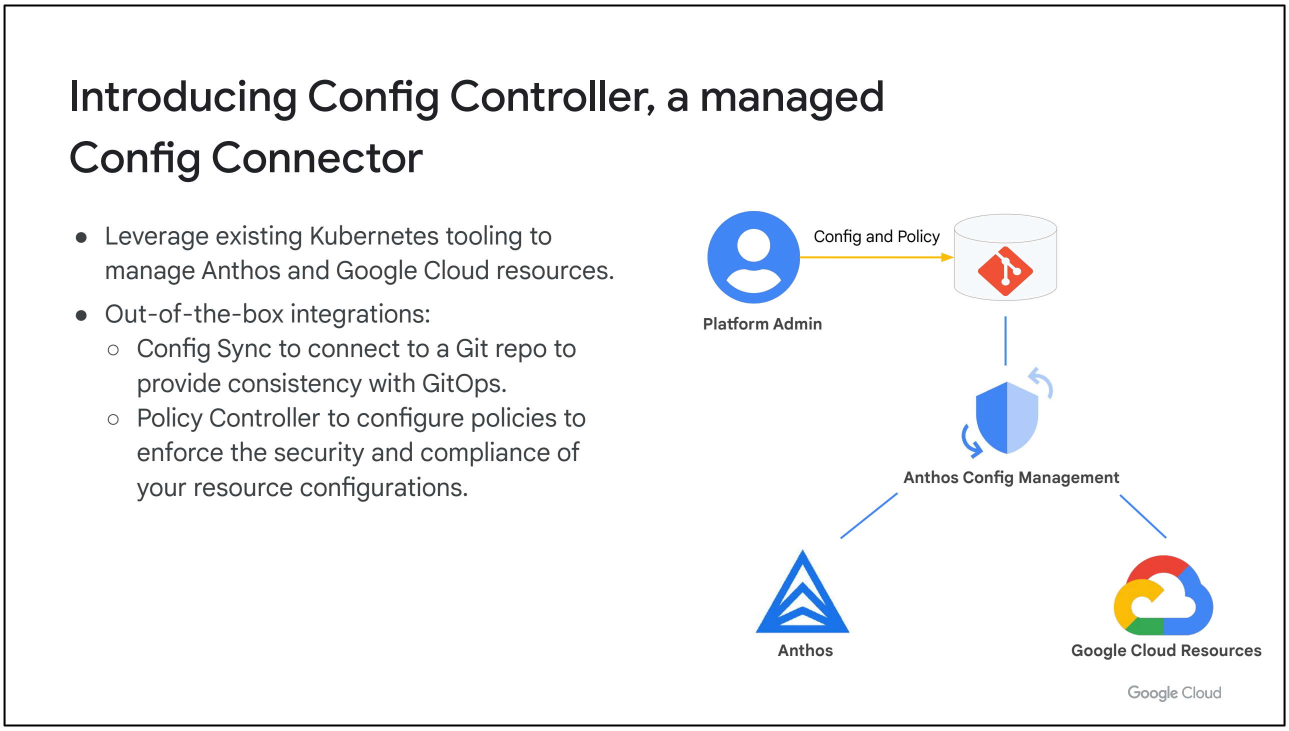
Google Cloud

Store your configuration in Git and get the power of a declarative, collaborative way of deploying to multi-cloud, hybrid environments.

Guard your clusters with dynamic and custom security and compliance rules that get out of the way of your developers without compromising on corporate controls.

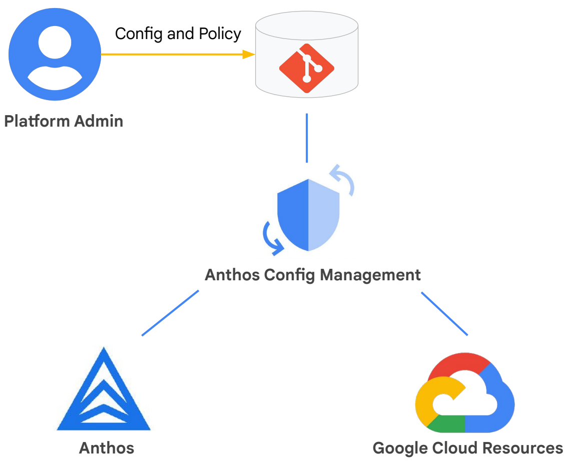
Apply Git-based deployment, active policy guardrails, and declarative APIs to Google Cloud, unifying your configuration in a common toolset.

All of this is available in a single package today. That’s Anthos Config Management.



Introducing Config Controller, a managed Config Connector

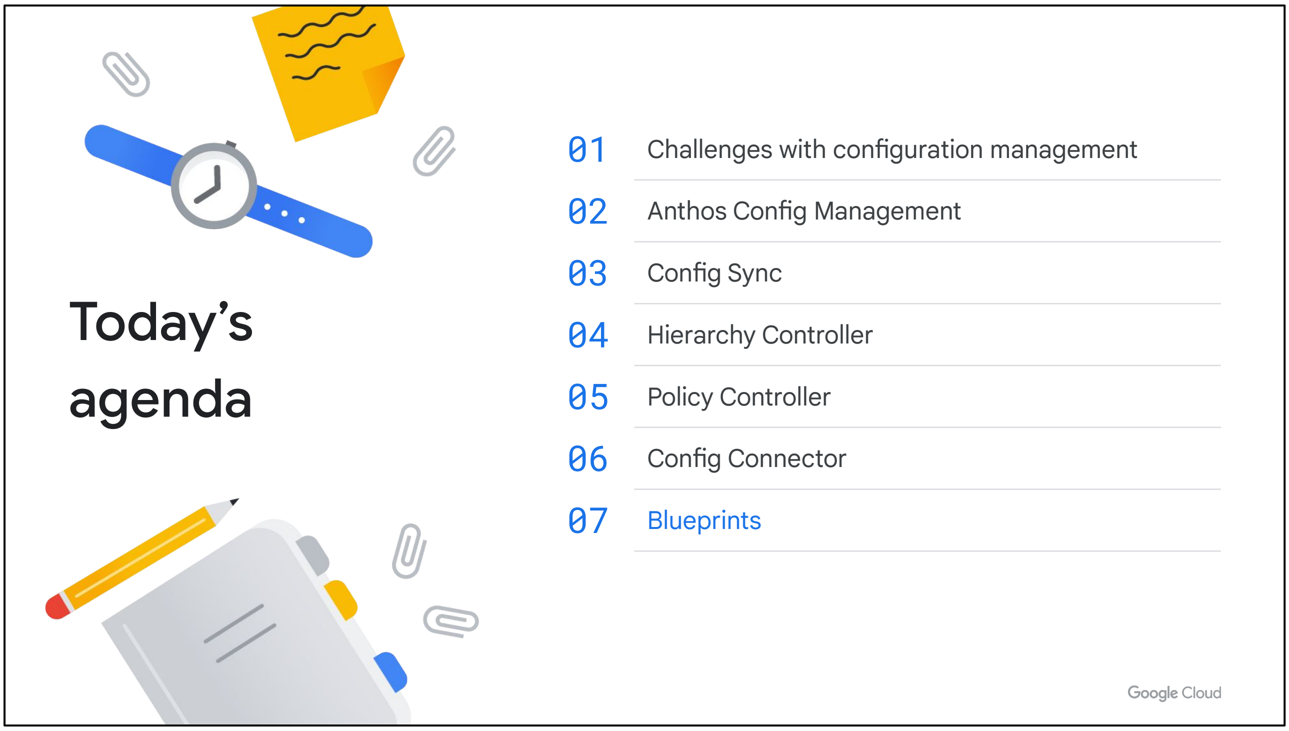
- Leverage existing Kubernetes tooling to manage Anthos and Google Cloud resources.
- Out-of-the-box integrations:
  - Config Sync to connect to a Git repo to provide consistency with GitOps.
  - Policy Controller to configure policies to enforce the security and compliance of your resource configurations.



Google Cloud

Config Controller is a Google-managed Config Connector that runs on Google infrastructure. Free up resources in your Anthos clusters and manage Google Cloud services with Config Controller outside of your cluster. Use the same CRDs you used in Config Connector and Google manages the controllers and the watch loop to make sure the current state matches your desired state.

Config Controller provides out-of-the-box integrations with Config Sync, so that you can continue to manage your configuration following the GitOps approach.
Also, it integrates with Policy Controller to configure policies to enforce the security and compliance of your cloud resource configurations.



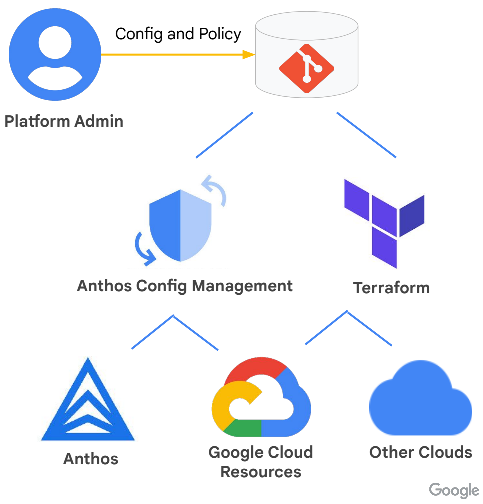
Today's agenda

Google Cloud

Using all Anthos Config Management can seem cumbersome the first time. That's why Google offers you Blueprints.

Blueprints enable reusability

- Package of deployable, reusable configuration and policy that implements and documents a specific opinionated solution following best practices.
- Codify knowledge and expertise for rapidly deploying new systems and environments with declarative configuration.
- Use Google-provided blueprints for both Kubernetes and Terraform.

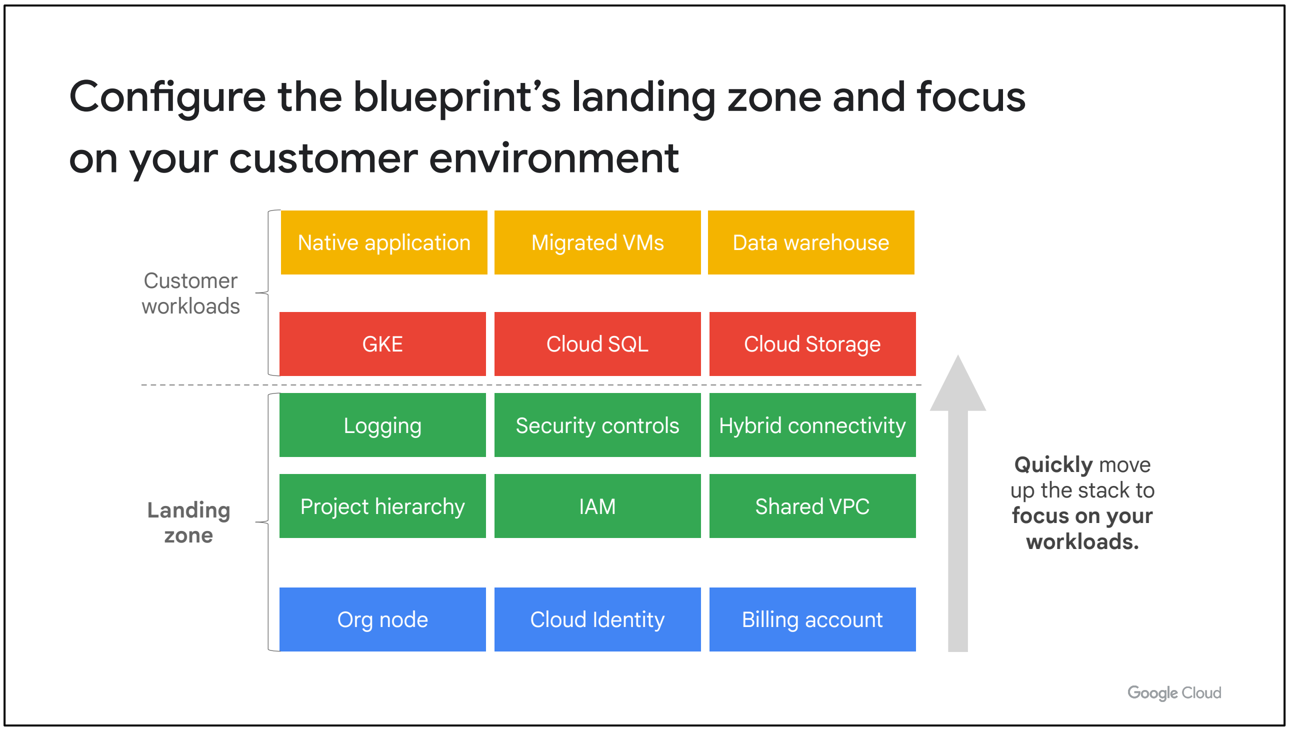


Google Cloud

A blueprint is a package of deployable, reusable configuration and policy that implements and documents a specific opinionated solution following best practices.

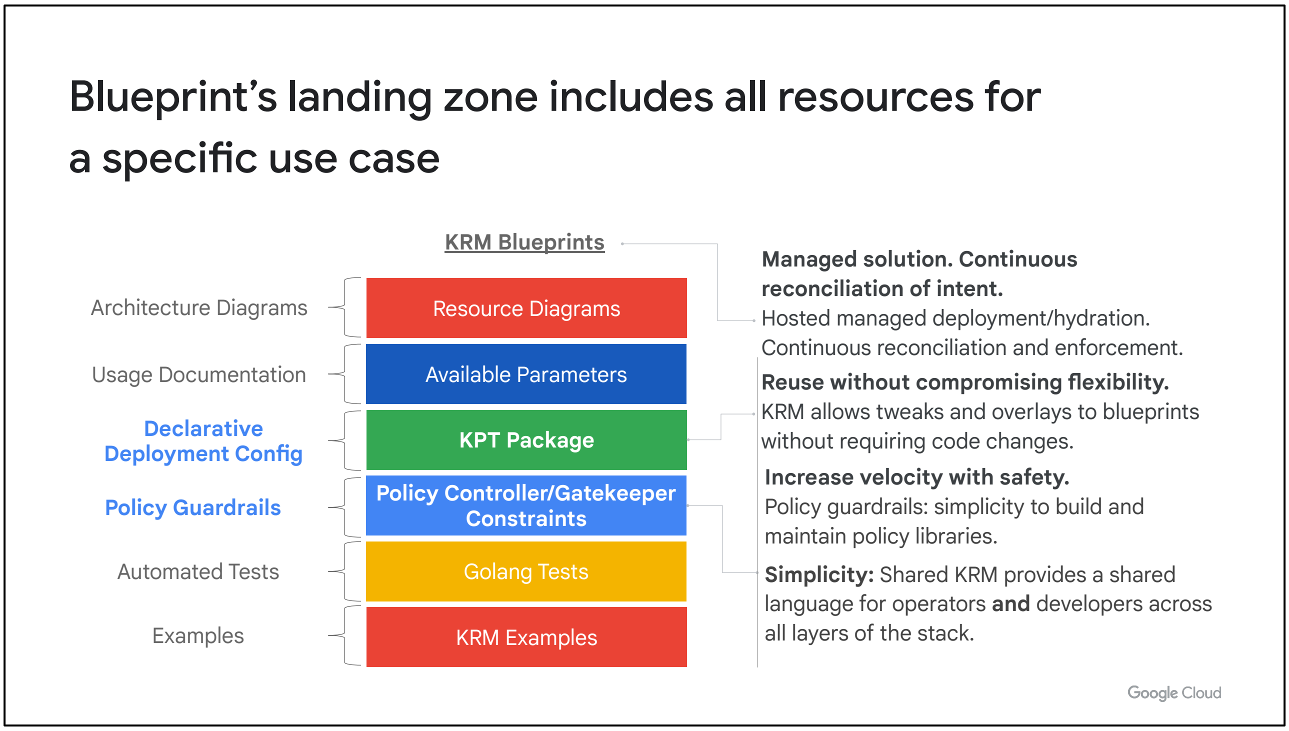
Blueprints enable reusability by enabling you to codify knowledge and expertise for rapidly deploying new systems and environments with declarative configuration.

Use Google-provided blueprints for both Kubernetes and Terraform, so that you can manage Kubernetes and Google Cloud resources and configuration, as well as extend to other Cloud providers with Terraform modules. These blueprints are released as versioned, shareable artifacts, which can be upgraded without breaking production environments.



Configure the blueprint's landing zone and focus on your customer environment

A landing zone is a setup of the overall structure of your organization, permissions, billing, logging, networking, etc. So that you can quickly configure the right structure using best practices and sensible, contextual defaults which reduce the need to tune every option on each resource. This makes onboarding faster and reduces cost.



Blueprint’s landing zone includes all resources for a specific use case

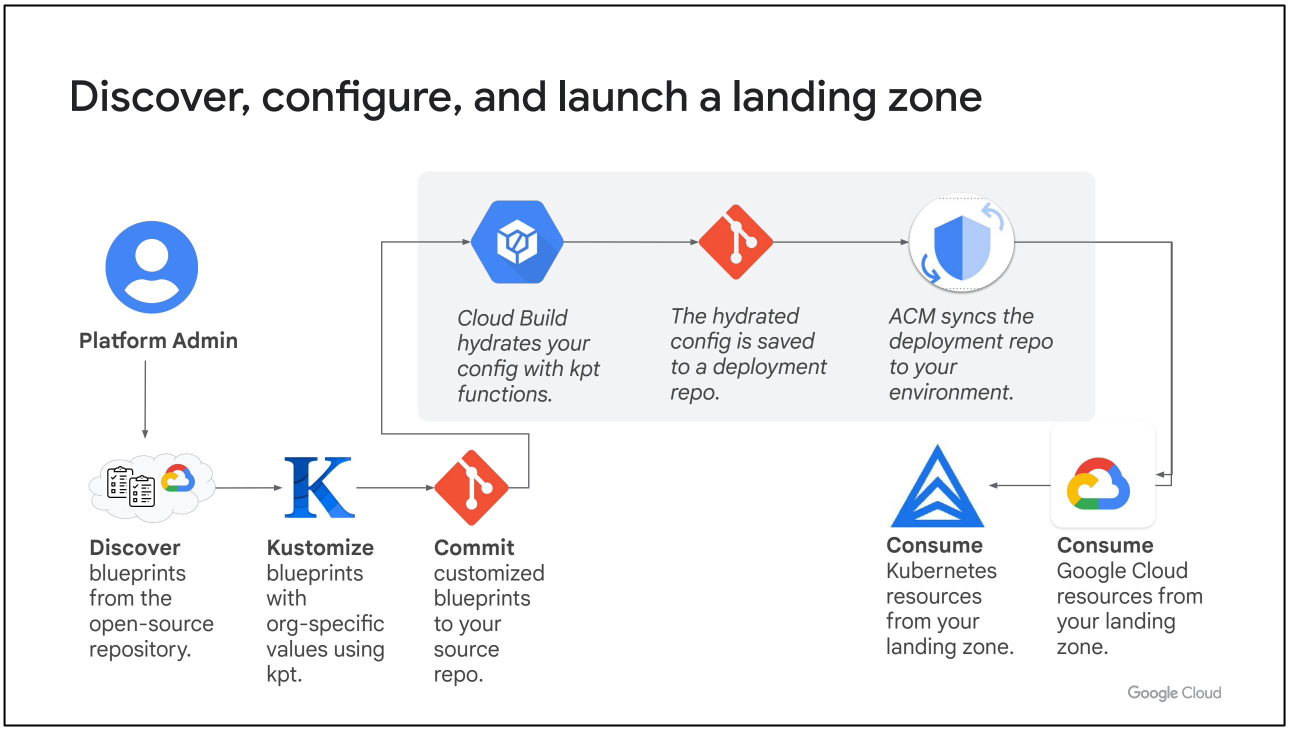
Managed solution. Continuous reconciliation of intent. Hosted managed deployment/hydration. Continuous reconciliation and enforcement.

Reuse without compromising flexibility. KRM allows tweaks and overlays to blueprints without requiring code changes.

Increase velocity with safety. Policy guardrails: simplicity to build and maintain policy libraries.

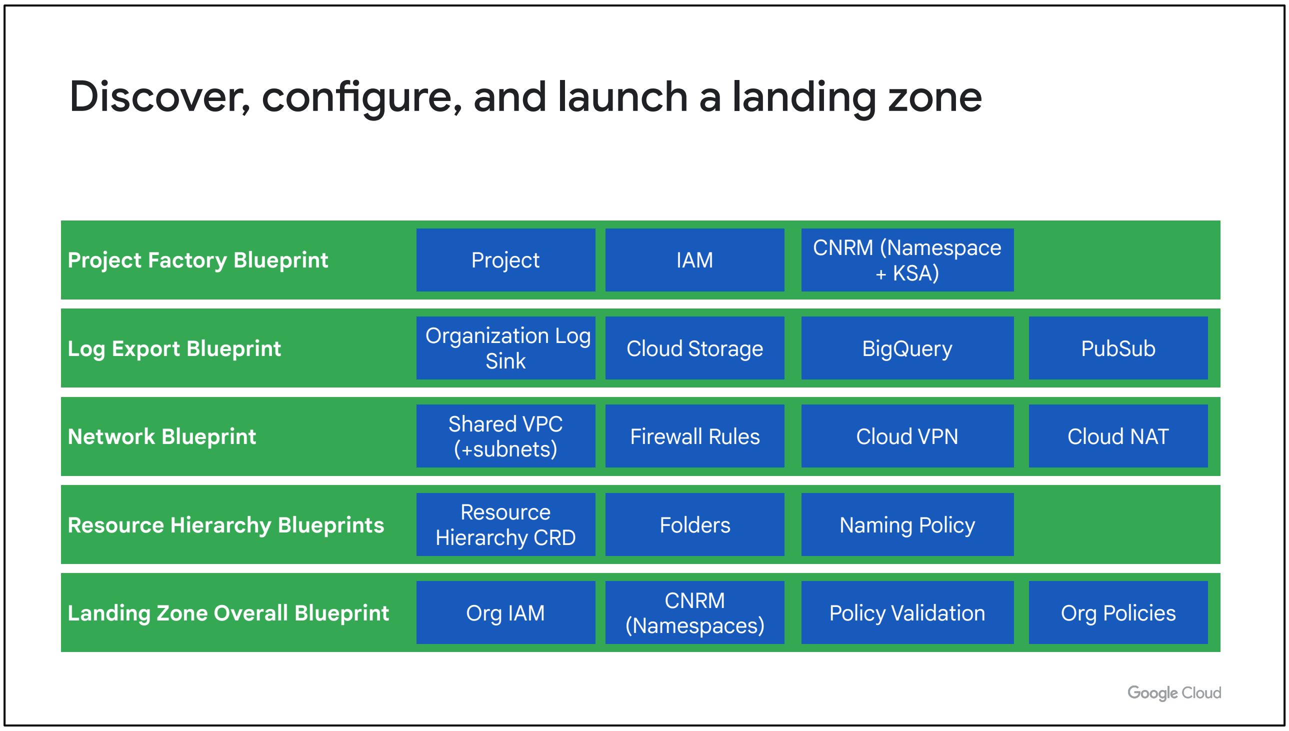
Simplicity: Shared KRM provides a shared language for operators **and** developers across all layers of the stack.

Blueprint’s landing zone includes all resources for a specific use case so that you can share a KRM model across all layers of the organization, including administrators, app operators, and developers. Why is this key? Because you can reuse assets by reusing of blueprints without complex coding and build and maintain policy libraries that can be validated early in the CI/CD pipeline.


Discover, configure, and launch a landing zone
Platform Admin
Cloud Build hydrates your config with kpt functions.
The hydrated config is saved to a deployment repo.
ACM syncs the deployment repo to your environment.
Discover blueprints from the open-source repository.
Kustomize blueprints with org-specific values using kpt.
Commit customized blueprints to your source repo.
Consume Kubernetes resources from your landing zone.
Consume Google Cloud resources from your landing zone.
Google Cloud

What does the process look like?

- An administrator goes through the blueprints in Google's open-source repository and selects a blueprint that fits their needs.
- Then they use kpt to fill the organization-specific values into the blueprint's landing zone and commits it to a git repository.
- As part of the GitOps blueprint, they create a Cloud Build trigger which monitors the source repository for changes, so that changes are validated using kpt functions.
- And the final "hydrated" configuration is saved to the repository and pushed to your cluster using Config Sync.
- Finally, all Kubernetes and Google Cloud changes are reconciled and resources are created.

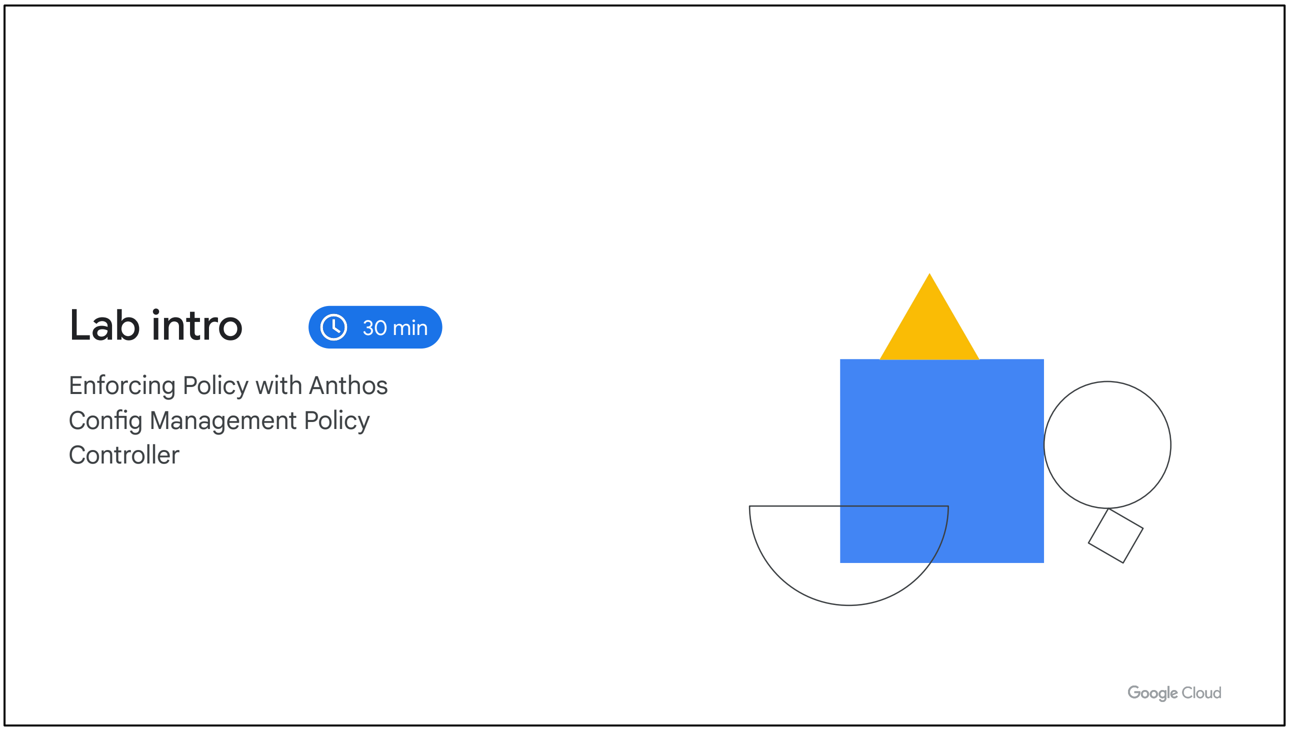


Discover, configure, and launch a landing zone

Project Factory Blueprint	Project	IAM	CNRM (Namespace + KSA)	
Log Export Blueprint	Organization Log Sink	Cloud Storage	BigQuery	PubSub
Network Blueprint	Shared VPC (+subnets)	Firewall Rules	Cloud VPN	Cloud NAT
Resource Hierarchy Blueprints	Resource Hierarchy CRD	Folders	Naming Policy	
Landing Zone Overall Blueprint	Org IAM	CNRM (Namespaces)	Policy Validation	Org Policies

Google Cloud

Here are some ready-to-deploy, out-of-the-box blueprints with all best practices from Google's library.



Lab intro

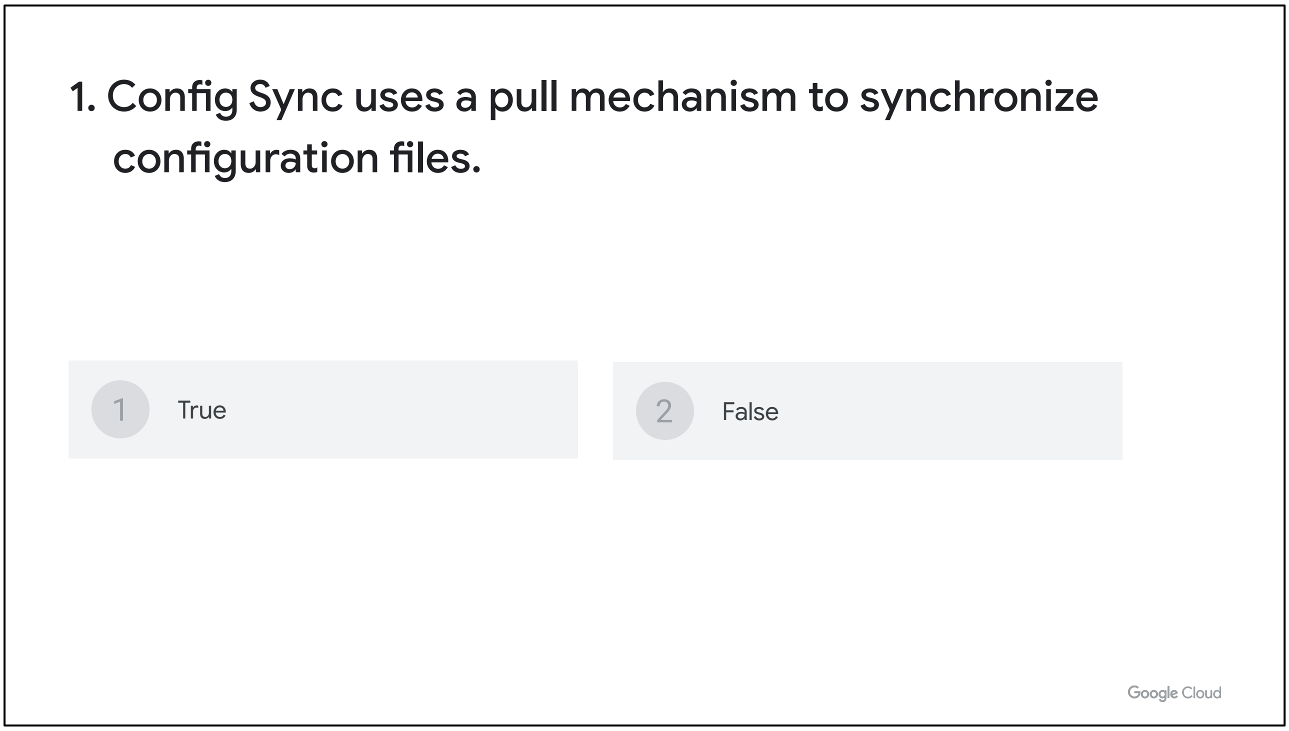
30 min

Enforcing Policy with Anthos Config Management Policy Controller

Google Cloud

Objectives:

- Review the installation of the Anthos Config Management and Policy Controller.
- Create and enforce constraints using the Template Library provided by Google.
- Audit constraint violation.
- Create your own Template Constraints to create the custom compliance policies that your company needs.

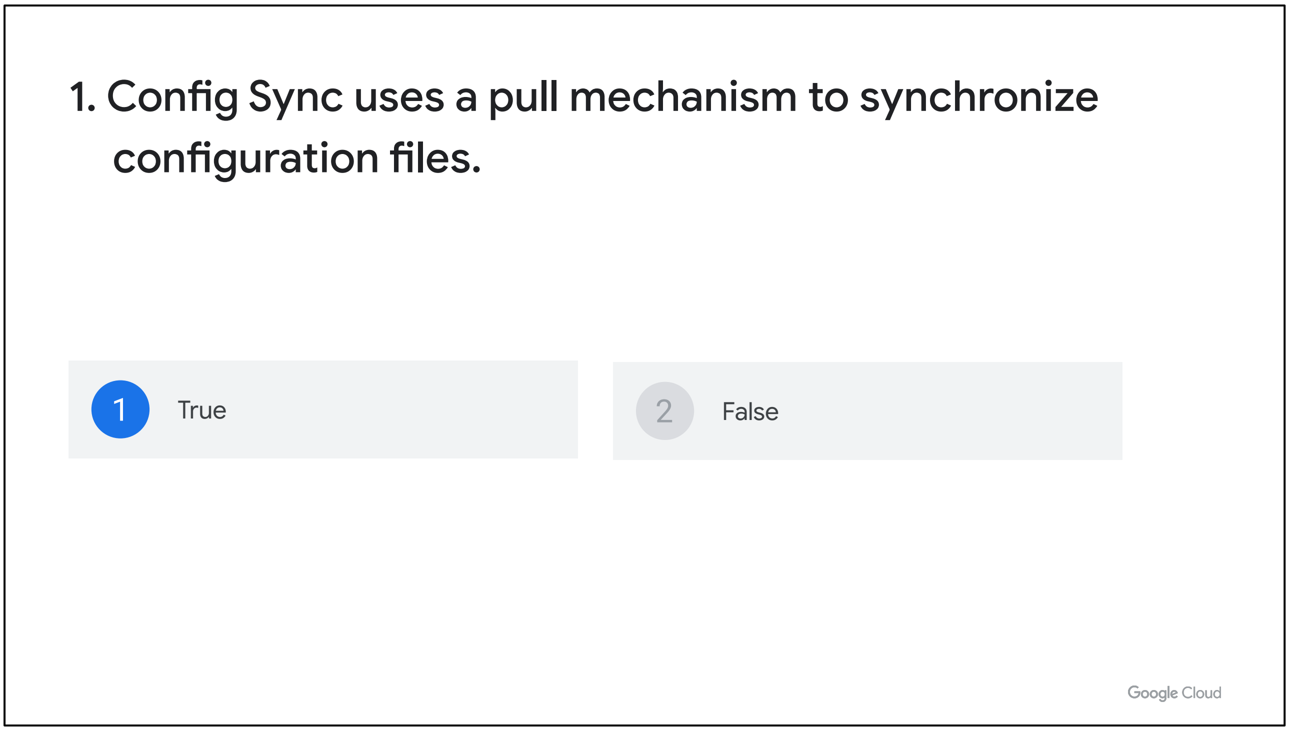


1. Config Sync uses a pull mechanism to synchronize configuration files.

1 True

2 False

Google Cloud



1. Config Sync uses a pull mechanism to synchronize configuration files.

[✓ 1 True]

2 False

Google Cloud

2. What CLI tool is used to create and manage Anthos clusters on bare metal?

1 gcloud

2 gsutil

3 bq

4 bmctl

Google Cloud

2. What CLI tool s used to create and manage Anthos clusters on bare metal?

1 gcloud

2 gsutil

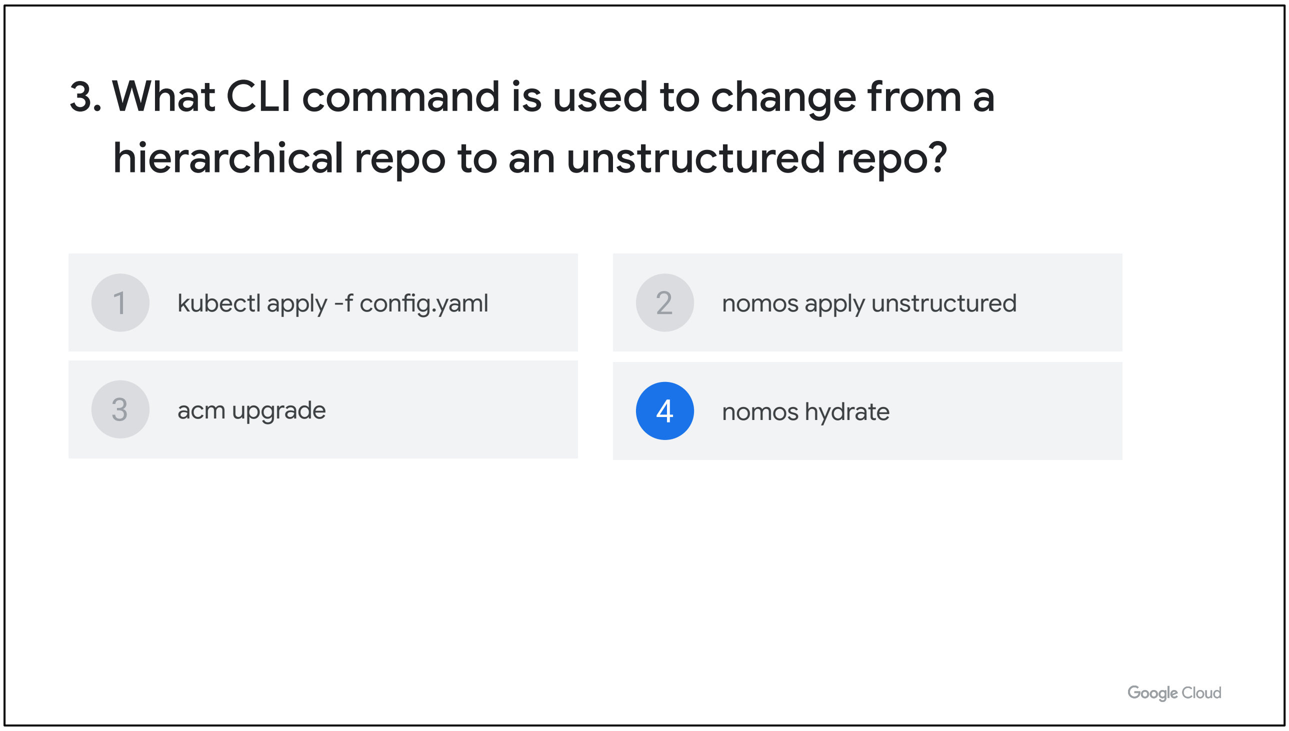
3 bq

[✓ 4] bmctl

Google Cloud

3. What CLI command is used to change from a hierarchical repo to an unstructured repo?

1. kubectl apply -f config.yaml
2. nomos apply unstructured
3. acm upgrade
4. nomos hydrate



3. What CLI command is used to change from a hierarchical repo to an unstructured repo?

1 kubectl apply -f config.yaml	2 nomos apply unstructured
3 acm upgrade	[✓ 4] nomos hydrate

Google Cloud