

04

Access Control and Security in Kubernetes and GKE

Earlier in this course, you learned how to run workloads within a GKE cluster.

Now you'll focus on securing your cluster and applications by establishing appropriate access control mechanisms. This is essential for ensuring that only authorized users can perform specific actions within your GKE environment, thereby protecting your valuable resources.

Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity
- 04 Kubernetes Control Plane security
- 05 Pod security



Google Cloud

In this section titled, "Access Control and Security in Kubernetes and Google Kubernetes Engine, you'll:

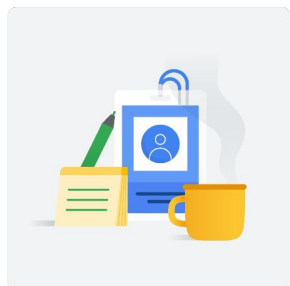
- Explore Kubernetes authentication and authorization.
- Examine Kubernetes role-based access controls and how it works with IAM to secure GKE clusters.
- Configure Workload Identity to access Google Cloud services from within GKE.
- Secure GKE with Kubernetes's Pod Security Standards and Pod Security Admission.
- And implement Role-Based Access Control with GKE.

Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity
- 04 Kubernetes Control Plane security
- 05 Pod security

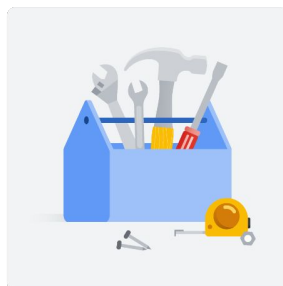


Kubernetes user access falls into two categories



Typically defined through a Cloud Identity domain.

Individual user accounts



Workload Identity bridges the gap between Kubernetes service accounts and IAM service accounts.

It provides access to other Google Cloud services.

Kubernetes service accounts

Within Kubernetes, user access falls into two categories: individual user accounts and Kubernetes service accounts. Whereas normal user accounts represent human users, Kubernetes service accounts manage credentials for applications running within the cluster.

In GKE, *user accounts* are typically defined through a Cloud Identity domain, which provides centralized management and fine-grained control, as compared to using individual consumer Google accounts. But then there is also Identity and Access Management, or **IAM**, which is a core Google Cloud service that lets you link permissions to *user accounts* and *groups*. We'll explore this in more detail soon.

To simplify managing *service accounts* in GKE clusters, Google Cloud offers **Workload Identity**, a feature that bridges the gap between Kubernetes service accounts and IAM service accounts in Google Cloud, which enables access to *other* Google Cloud services for applications that run within your cluster.

Workload Identity securely authenticates pods



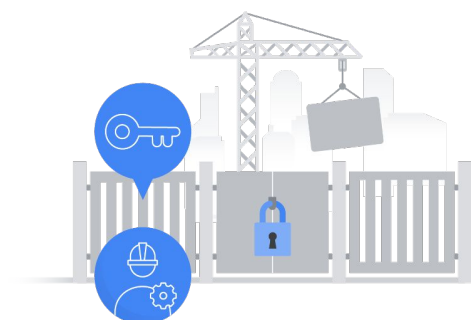
- Securely authenticate pods by keeping service account credentials outside the pods themselves.
- Pods automatically receive their service account credentials from the GKE metadata server.
- The GKE metadata server is a subset of the Compute Engine metadata server endpoints.
- Workload Identity simplifies pod authentication by eliminating the need to manage credentials within the pods.

Workload Identity offers a secure way to authenticate pods by keeping service account credentials outside the pods themselves. This prevents attackers from accessing these credentials even if a pod is compromised. Once a pod starts, it automatically receives its service account credentials from the GKE metadata server, allowing it to authenticate itself to other Google Cloud services securely.

The GKE metadata server is a subset of the Compute Engine metadata server endpoints required for Kubernetes workloads.

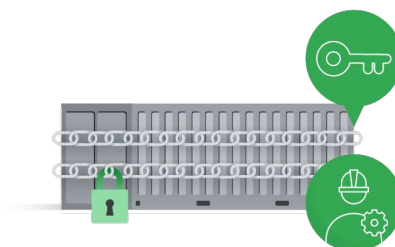
Furthermore, Workload Identity simplifies Pod authentication by eliminating the need to manage credentials within the Pods. This enhances both security and convenience for Kubernetes deployments and makes it a best practice for any deployment requiring access to other Google Cloud services.

Account authorization with IAM and RBAC



IAM

Define precise permissions
at the project or cluster level.



RBAC

Define granular permissions
at the cluster and namespace levels.

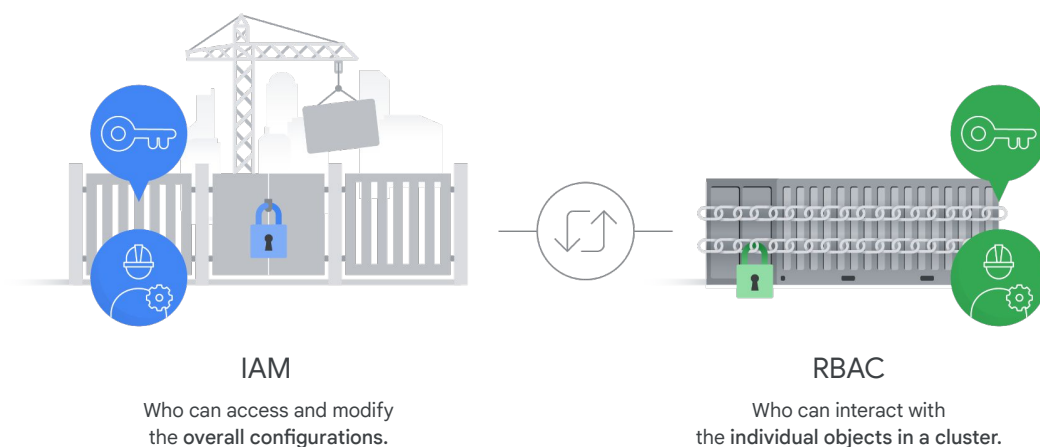
Google Cloud

Now once an account is *authenticated*, there are two main ways to *authorize* which actions it can perform: with **IAM** and **Kubernetes role-based access control, or RBAC**.

Think of IAM as the gatekeeper for Google Cloud resources beyond your Kubernetes clusters. It lets you define precise permissions for individual users or groups, which allows them to perform specific actions at the project or cluster level.

Kubernetes RBAC controls access within your cluster, offering fine-tuned roles with granular permissions for resources at the *cluster* and *namespace* levels.

Comprehensive access control



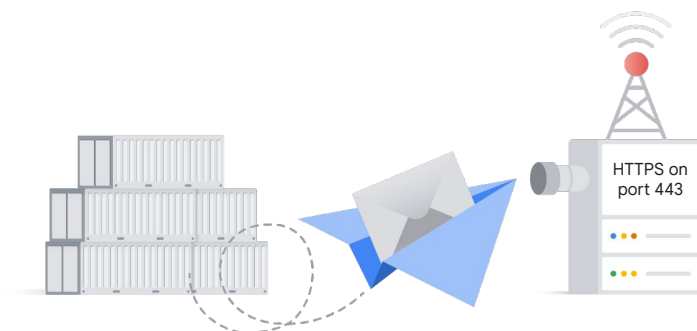
Google Cloud

To ensure comprehensive access control within your GKE environment, it's essential to have a synchronized approach involving both IAM and Kubernetes RBAC.

IAM acts as the outermost layer of security and defines who can access and modify the overall configurations of your GKE clusters, whereas Kubernetes RBAC governs who can interact with the individual Kubernetes objects residing within a cluster.

It's also crucial to adhere to the principle of least privilege, a cornerstone of security best practices. This means granting each user or group only the minimum set of permissions necessary to perform their specific tasks, minimizing potential risks.

Authenticating requests outside a cluster



GKE manages end-user authentication for you.
It is best practice to authenticate using OpenID connect tokens.

Google Cloud

Now your workloads might get requests from outside the cluster, and each request must be authenticated before it's acted upon. In Kubernetes, the API server listens for remote requests by using HTTPS on port 443.

GKE manages end-user authentication for you. It will authenticate users to Google Cloud, set up the Kubernetes configuration, get an OAuth access token for the cluster, and keep the access token up-to-date.

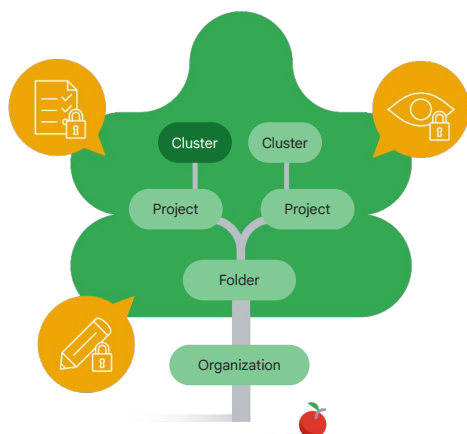
In Google Cloud it is best practice to authenticate using OpenID connect tokens.

Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity
- 04 Kubernetes Control Plane security
- 05 Pod security



Enhanced security with Kubernetes RBAC + IAM



IAM manages access at the GKE and cluster levels.

RBAC provides control over individual Kubernetes resources inside the cluster.

Kubernetes Role-Based Access Control, or RBAC, is a built-in security mechanism that allows for granular control over user permissions within a Kubernetes cluster.

Within GKE, RBAC works alongside IAM to strengthen security. While IAM manages access at the GKE and cluster levels, RBAC provides control over individual Kubernetes resources inside the cluster.

What Kubernetes RBAC controls



Subjects

Users, groups, or service accounts authorized to interact with the Kubernetes API server.



Verbs

Define the specific actions allowed for each resource, such as creating, modifying, or viewing.



Resources

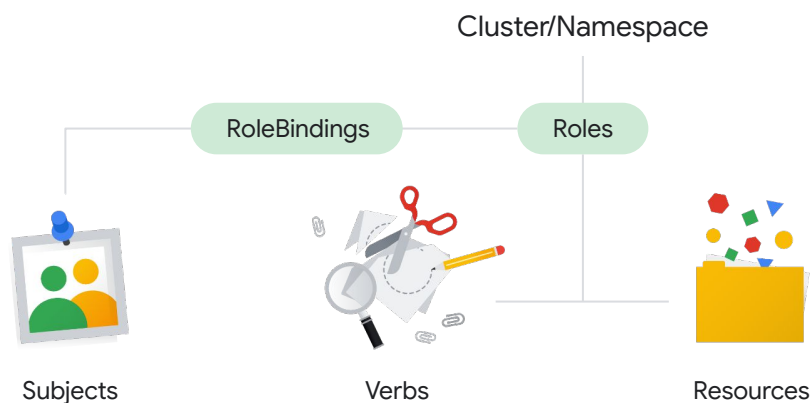
Encompass the Kubernetes objects the subjects can access.

Kubernetes RBAC can control who can do what with Kubernetes objects.

- **Subjects**, the who, are users, groups, or service accounts authorized to interact with the Kubernetes API server.
- **Verbs**, the what, define the specific actions allowed for each resource, such as creating, modifying, or viewing.
- And **resources** encompass the Kubernetes objects the subjects can access, like pods, deployments, or persistent volumes.

Think of it as defining who gets the keys, what doors they can unlock, and what actions they can perform inside.

What Kubernetes RBAC Controls



By combining these elements, two types of RBAC API objects can be created, **roles** and **RoleBindings**. Both applied at the cluster or namespace level, roles connect API *resources* and *verbs* and RoleBindings connect roles to *subjects*.

To use RBAC to grant users access to specific Kubernetes namespaces, you need to create roles and RoleBindings. First, you need to create the roles with proper permissions in each namespace. Then you need to create RoleBindings to bind subjects to the appropriate roles.

Defining namespace RBAC roles in manifests

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
  namespace: default # Specify the namespace where the Role applies
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

So, how does this work in practice? Let's explore how a manifest defines an RBAC role at the namespace level.

A manifest includes fields of the `apiVersion`, `kind`, `metadata`, and `rules`, which are permissions granted by the role.

Using the `kind` field, `Role` can be used to define a namespace-level role. Note that you can only define a single namespace for a role.

Under the `rules` field, you need to define the permissions granted by the role. This includes `apiGroups`, which are API groups of resources. Leave this field empty for the core API group.

The `resources` field specifies what resources the role can access (here, "Pods"), and the `verbs` field specifies the permitted actions on those resources.

RBAC ClusterRoles are defined at the cluster level

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin # Example name for a ClusterRole
rules:
- apiGroups: ["", "extensions", "apps"] # Multiple API groups
  resources: ["*"] # Apply to all resources within those groups
  verbs: ["*"] # Allow all actions on those resources
```

Whereas RBAC Roles are defined at the namespace level, RBAC ClusterRoles are defined at the cluster level.

In practice, this means defining the `kind` field of a manifest file with `ClusterRole` and leaving the `metadata` field blank, indicating a cluster-wide scope.

What RBAC rule sets control

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  resourceName: ["demo-pod"]
  verbs: ["patch", "update"]
```

- Used to limit the scope to:
 - Specific instances of a resource.
 - Verbs specified as patch and update.

```
rules:
- apiGroups: [""]
  resources: ["pods", "logs"]
  verbs: ["get", "list", "watch"]
```

- Used to specify access to Pod logs.
- Assigned as a unit, all or none.

```
rules:
- nonResourceURLs: ["metrics/", "/metrics/*"]
  verbs: ["get", "post"]
```

- Used to specify get and post actions for non-resource endpoints.
- Unique to ClusterRoles.

Google Cloud

Let's explore some rule sets that specify the resources and verbs covered by a Role or ClusterRole.

A resourceName is used to limit the scope to specific instances of a resource and the verbs specified as patch and update.

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  resourceName: ["demo-pod"]
  verbs: ["patch", "update"]
```

A subresource 'logs' can be added to the resources list to specify access to Pod logs also. The combination of verbs, get, list and watch, are the standard read-only verbs. You should typically assign them as a unit, all or none.

```
rules:
- apiGroups: [""]
  resources: ["pods", "logs"]
  verbs: ["get", "list", "watch"]
```

And nonResourceURLs can be used to specify get and post actions for non-resource endpoints. This form of rule is unique to ClusterRoles.

```
rules:
- nonResourceURLs: ["metrics/", "/metrics/*"]
  verbs: ["get", "post"]
```


Attaching RoleBindings and ClusterRoleBindings

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: default
  name: demo-rolebinding
subjects:
- kind: User
  name: "joe@example.com"
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: demo-role
  apiGroup: rbac.authorization.k8s.io
```

After ClusterRoles are defined, RoleBindings and ClusterRoleBindings can be attached.

To attach a RoleBinding, start by defining the namespace where it applies, then carefully specify the users, groups, or service accounts that will receive the permissions, paying attention to case sensitivity in their names.

With RBAC in GKE, you can control access for various user types: individual Google accounts, service accounts tied to Google Cloud projects, and identities associated with your workloads through Workload Identity. All these entities are identified by their email addresses.

To connect roles to users or groups, use the `roleRef` field within a RoleBinding. Specify both the role type—Role or ClusterRole—and its name. Remember, even ClusterRoles bound within a RoleBinding only grant permissions within that specific namespace, not the entire cluster.

kubectl api-resources command

kubectl api-resources

```
--namespaced=true - namespaced resources  
--namespaced=false - cluster-wide resources
```

- Cluster-level resources: ClusterRoles
- Namespaced resources: Roles
- Multiple namespaces: ClusterRoles

To identify both namespaced and non-namespaced resources, use the `kubectl api-resources` command, and specify `--namespaced=true` for namespaced resources and `--namespaced=false` for cluster-wide resources.

Before creating Roles or ClusterRoles, you can verify the resource scope—namespace or cluster-wide—using `kubectl api-resources` to list "NameSpaced" resources.

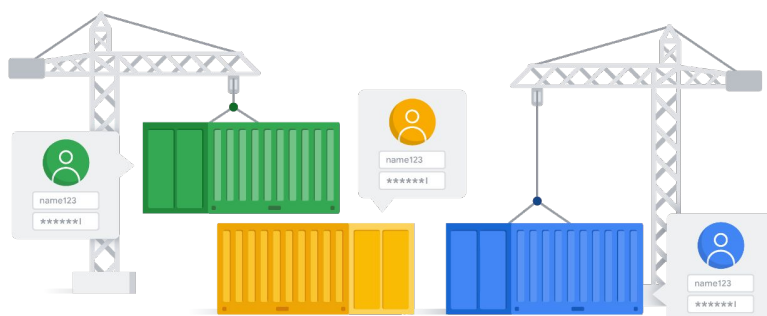
Typically, you should manage cluster-level resources with ClusterRoles and NameSpaced resources with Roles. For permissions spanning multiple namespaces, you'll likely need to use ClusterRoles.

Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity**
- 04 Kubernetes Control Plane security
- 05 Pod security

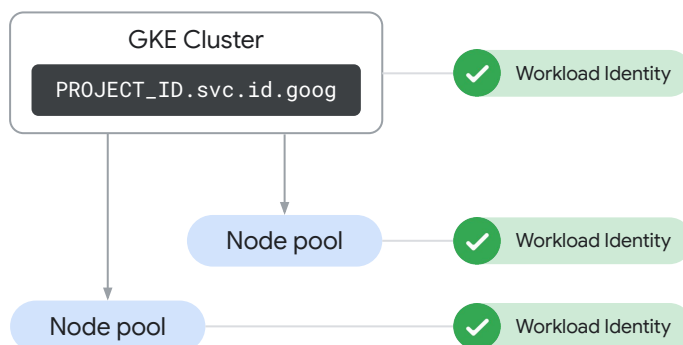


GKE Workload Identity simplifies how containerized applications access Google Cloud services



GKE Workload Identity simplifies how containerized applications access Google Cloud services. Instead of managing service account keys in a GKE environment, containers can directly authenticate by using their dedicated credentials. This enhances security and streamlines access control for deployments.

How do you use the Workload Identity feature?



So, how do you use the Workload Identity feature?

When you enable Workload Identity on the first cluster in a Project, GKE automatically create the pool by using the format "PROJECT_ID.svc.id.goog".

Any new node pools that you create have Workload Identity enabled by default. Containers deployed to those node pools are then able to use its service account credentials to authenticate to Google Cloud services, either through the Google Cloud Command Line Interface or by using the Application Default Credentials library.

To use Workload Identity on individual node pools, it must first be activated at the cluster level. It's useful to know that Autopilot clusters enable Workload Identity by default.

Commands to enable Workload Identity on clusters

gcloud container clusters create

Used to enable Workload Identity on a new standard cluster

```
gcloud container clusters create MyTestCluster
--region=us-central1-a
--workload-pool=myProjectID.svc.id.goog
```

gcloud container clusters update

Used to enable Workload Identity on an existing standard cluster

```
gcloud container clusters update MyTestCluster
--region=us-central1-a
--workload-pool=myProjectID.svc.id.goog
```

A couple of commands are available to enable Workload Identity on clusters.

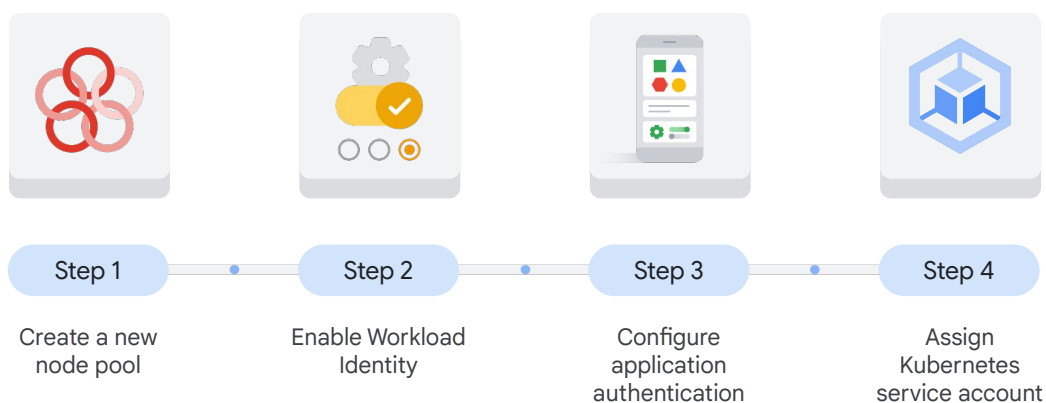
The first is the **gcloud container clusters create** command, which can be used to enable Workload Identity on a new standard cluster.

```
gcloud container clusters create MyTestCluster
--region=us-central1-a    --workload-pool=myProjectID.svc.id.goog
```

And second, there is the **gcloud container clusters update** command, which can be used to enable Workload Identity on an existing standard cluster.

```
gcloud container clusters update MyTestCluster
--region=us-central1-a    --workload-pool=myProjectID.svc.id.goog
```

Create a new node pool before migrating applications



Google Cloud

Before you migrate any applications to use Workload Identity, it's a best practice to create a new node pool.

After Workload Identity is enabled, you need to configure your applications to authenticate to Google Cloud using Workload Identity.

You need to assign a Kubernetes service account to the application, and then configure that Kubernetes service account to act as an IAM service account.

Configuring an application to use Workload Identity

01 Create a new namespace for the service account.

```
kubectl create namespace NAMESPACE
```

02 Create a new Kubernetes ServiceAccount for your application.

```
kubectl create serviceaccount KSA_NAME --namespace NAMESPACE
```

03 Ensure the IAM service account has the required roles.

```
gcloud projects add-iam-policy-binding myProjectID \
--role="ROLE_NAME"
--member=principal://iam.googleapis.com/projects/PROJECT_NUMBER/locations/global/
workloadIdentityPools/PROJECT_ID.svc.id.goog/subject/ns/NAMESPACE/sa/KSA_NAME \
--condition=None
```

Let's examine the steps involved in configuring an application to use Workload Identity.

1. The first step is to create a new namespace to use for the Kubernetes service account. You can use the `kubectl create namespace` command.
2. Next you'll need to create a new Kubernetes service account for your application to use, as opposed to using an existing Kubernetes service account in any namespace, including the default service account. To accomplish this, use the `kubectl create serviceaccount` command.
3. From there, create an IAM allow policy that references the Kubernetes ServiceAccount. To do so, use the `add-iam-policy-binding gcloud` command. You can use any IAM service account in any project in your organization. Be sure to grant permission to the specific Google Cloud resources that your application needs to access.

Configuring an application to use Workload Identity

04 Grant role to ServiceAccount.

```
gcloud storage buckets add-iam-policy-binding
gs://BUCKET --role=roles/storage.objectViewer \
--member=principal://iam.googleapis.com/project
s/PROJECT_NUMBER/locations/global/workloadIdent
ityPools/PROJECT_ID.svc.id.goog/subject/ns/NAME
SPACE/sa/KSA_NAME \
--condition=None
```

05 Add Workload Identity in pod manifest.

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: NAMESPACE
spec:
  serviceAccountName: KSA_NAME
  containers:
    - name: test-pod
      [...]
```

06 Apply this to your cluster with a kubectl apply command.

```
kubectl apply -f WorkloadID.yaml
```

4. The next step is to grant the required roles to the Service Account. For this example the role, *Storage Object Viewer* is granted.

5. And finally add the name of the Service Account and identify which specific nodes the Pod should be scheduled to run on, to the specification YAML file.

6. This can be applied to your cluster by using a `kubectl apply` command.

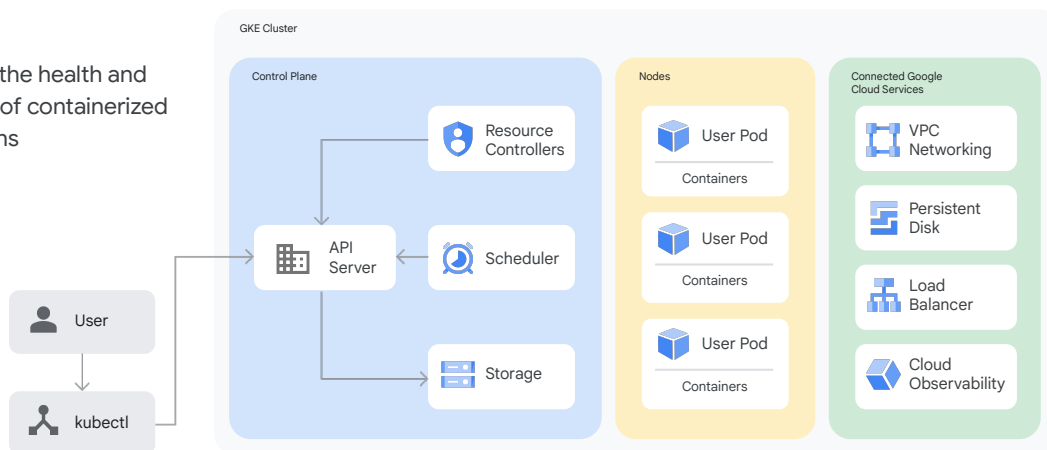
Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity
- 04 Kubernetes Control Plane security**
- 05 Pod security



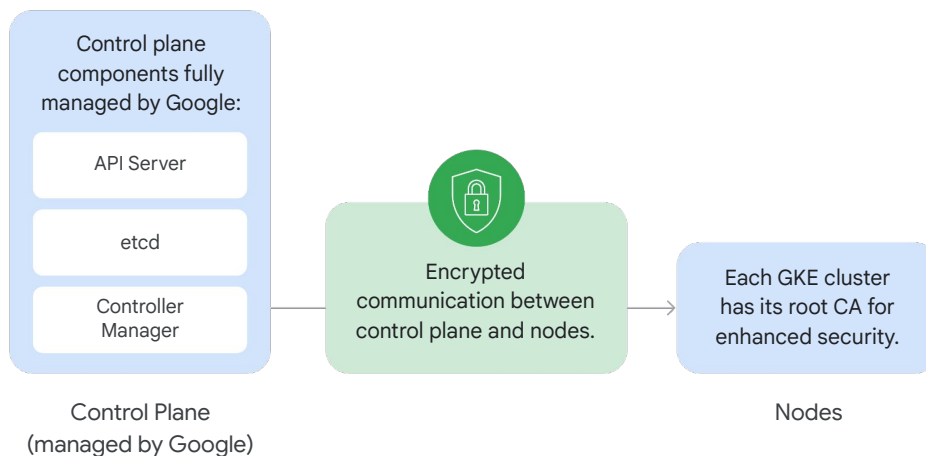
The Kubernetes control plane is the brain of a cluster

Maintains the health and operation of containerized applications



The Kubernetes control plane, which is the brain of a cluster, is critical for maintaining the health and operation of containerized applications. Securing it is paramount, because any compromise can grant attackers access and control over an entire cluster.

Google manages all control plane components

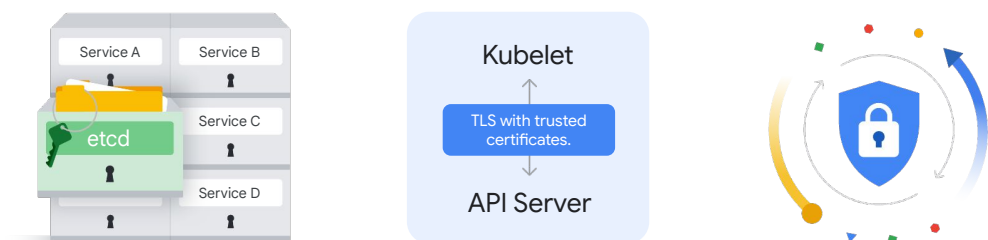


In GKE, Google manages all of the control plane components. This includes the API server (kube-apiserver), etcd (distributed key-value store), and the controller manager.

Each GKE cluster has a dedicated root certificate authority (CA) for security, and Google internally manages the root keys for these CAs to ensure their integrity.

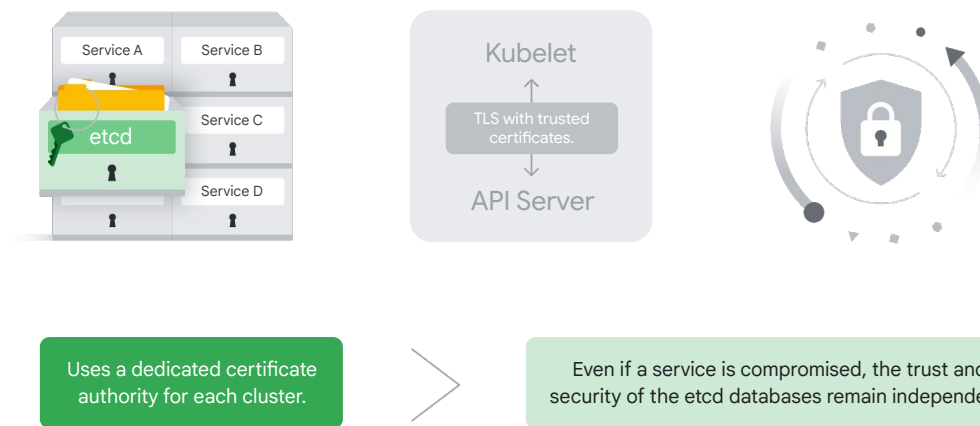
Secure communications between the control plane and nodes in a cluster relies on the shared root of trust provided by the certificates issued by this CA.

Ways to secure the GKE control plane



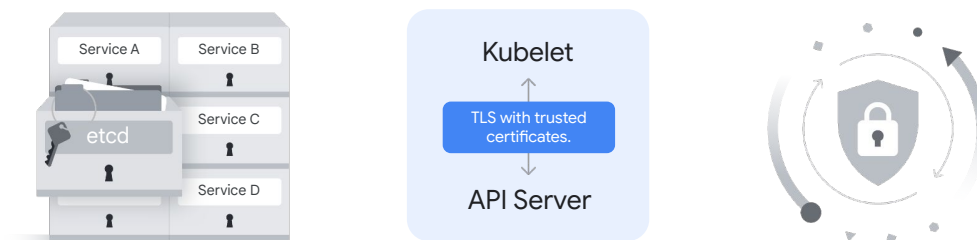
Let's explore how GKE implements security features on the control plane.

GKE isolates etcd databases from other cluster service



The first is that **GKE isolates etcd databases** from other cluster services by using a dedicated certificate authority for each cluster. This means that even if a service is compromised, the trust and security of the etcd databases remain independent, which prevents potential breaches from spreading.

Secure communication: Kubelets and the API server



Maintain secure communication with the API server through trusted certificates issued by the cluster's root CA.



Safeguards the confidentiality and integrity of interactions between Kubelets and the API server.

Then there are **kubelets**, which are the primary Kubernetes agents located on nodes. Kubelets maintain secure communication with the API server through trusted certificates issued by the cluster's root CA. This mechanism safeguards the confidentiality and integrity of interactions between Kubelets and the API server.

How kubelets obtain certificates

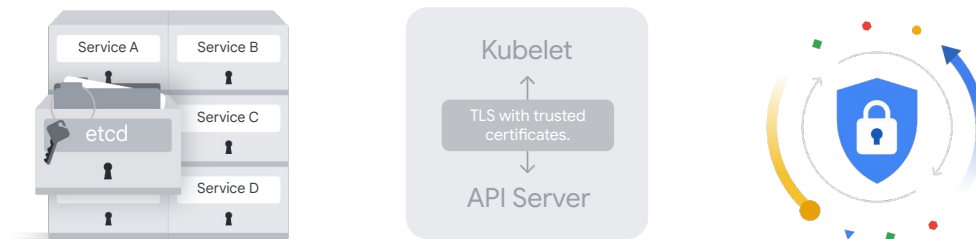


So, how does this work?

When a new node is created, it's provisioned with a shared secret. A "secret" refers to a special type of Kubernetes object used to securely store sensitive information, such as credentials or encryption keys.

This secret enables the kubelet on each node to initiate certificate signing requests to the cluster's root CA. Nodes then acquire client certificates upon joining the cluster and obtain fresh certificates when renewal or rotation is required.

Credential rotation means regularly rotating credentials for GKE control plane component



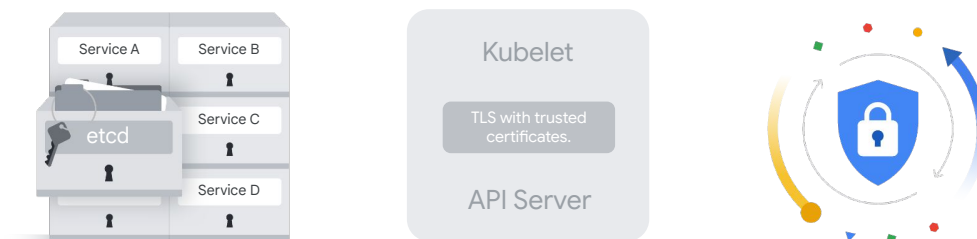
Frequency considerations:

- Balance security needs with operational overhead.
- Align rotation frequency with your organization's security policies.
- Consider sensitivity of data and assets managed by the cluster.

An important control plane security feature is **credential rotation**, which is the practice of regularly rotating credentials for GKE control plane components and nodes to minimize the window of opportunity for attackers who might exploit compromised credentials.

How frequent you rotate credentials will depend on your organization's security policies. For example, if your cluster manages high-value assets, you might want to rotate credentials more often, but not so often that it causes disruption to your cluster.

The credential rotation process in GKE



1. Create a new IP address for the cluster control plane.
2. Issue new certificate credentials to the control plane with the new IP address.
3. Update the nodes to use the new IP address and credentials.

4. Update all API clients outside the cluster to use the new credentials.
5. Remove the old IP address and old credentials.

The credential rotation process in GKE involves several steps.

1. The first step is to create a new IP address for the cluster control plane.
2. The second step is to issue new certificate credentials to the control plane with the new IP address.
3. The third step is to update the nodes to use the new IP address and credentials.
4. The fourth step is to update all API clients outside the cluster to use the new credentials.
5. And the final step to complete the rotation is to remove the old IP address and old credentials.

It's important to note that to prevent your cluster from entering an unrecoverable state if your current credentials expire, GKE will automatically start a credential rotation within 30 days of your current CA expiry date.

Commands to help complete a credential rotation

start-credential-rotation
gcloud command

```
gcloud container clusters update [NAME] \  
  --region=[REGION_NAME] \  
  --start-credential-rotation
```

complete-credential-rotation
gcloud command

```
gcloud container clusters update [NAME] \  
  --region=[REGION_NAME] \  
  --complete-credential-rotation
```

Now let's explore a couple of the commands that can help complete these steps.

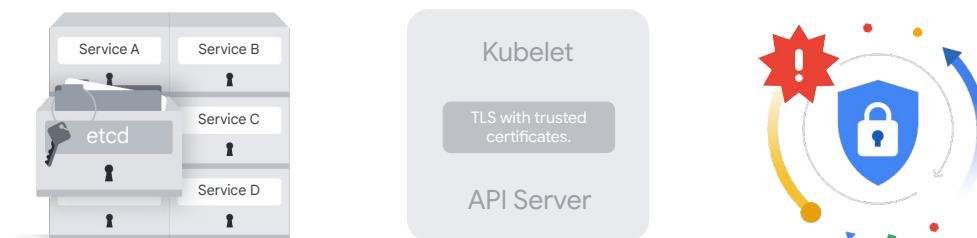
To initiate a credential rotation, use the **start-credential-rotation** gcloud command, which will create new credentials and issue them to the control plane.

```
gcloud container clusters update [NAME] \  
  --region=[REGION_NAME] \  
  --start-credential-rotation
```

To complete a rotation, use the **complete-credential-rotation** gcloud command. This command configures the control plane to serve only with the new credentials.

```
gcloud container clusters update [NAME] \  
  --region=[REGION_NAME] \  
  --complete-credential-rotation
```

Credential rotations may disrupt the cluster API server



- Can cause a temporary pause in new Pod creation.
- Existing Pods continue to run without disruption.
- Schedule Pod deployments before or after credential rotation.

Please know that during credential rotation, the cluster API server may experience a brief interruption, temporarily pausing new Pod creation. Running Pods will remain unaffected, which ensures ongoing operations. However, to avoid deployment delays, consider scheduling Pod creation before or after the rotation process.

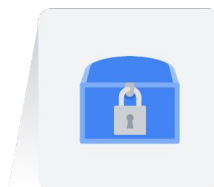
Secure GKE with a layered approach



Fine-tuning
permissions



Disabling
outdated access



Hiding sensitive
data

Whereas credential rotation helps secure your GKE cluster, it's important to make additional security considerations.

Consider *fine-tuning permissions* by granting the cluster's service account minimum privileges through IAM, *disabling outdated access* by turning off legacy metadata APIs for extra security, and *hiding sensitive data* by enabling metadata concealment to protect hidden credentials.

A layered approach to security is key for robust GKE cluster protection.

Access Control and Security in Kubernetes and GKE

- 01 Authentication and authorization
- 02 Kubernetes role-based access control
- 03 Workload Identity
- 04 Kubernetes Control Plane security
- 05 Pod security**



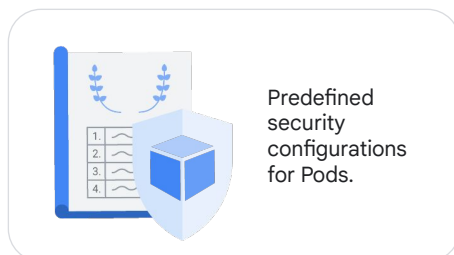
Secure Pods for a robust GKE environment



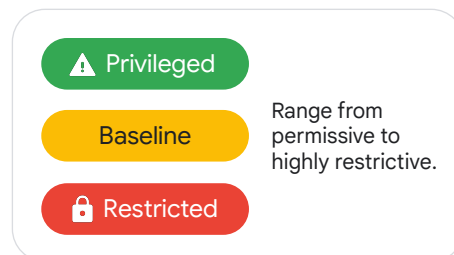
To ensure a robust and secure GKE environment, pod security is paramount.

Google Kubernetes Engine offers two primary mechanisms for achieving this. The first is **Pod Security Standards (PSS)** and the second is **Pod Security Admission (PSA)**.

Pod Security Standards (PSS)



Pod Security Standards (PSS)



Pre-defined PSS levels

Pod Security Standards are a set of predefined security configurations for Pods, which offers a flexible framework for securing Kubernetes clusters.

These predefined security policies range from permissive to highly restrictive, so you can match your security measures to your unique needs.

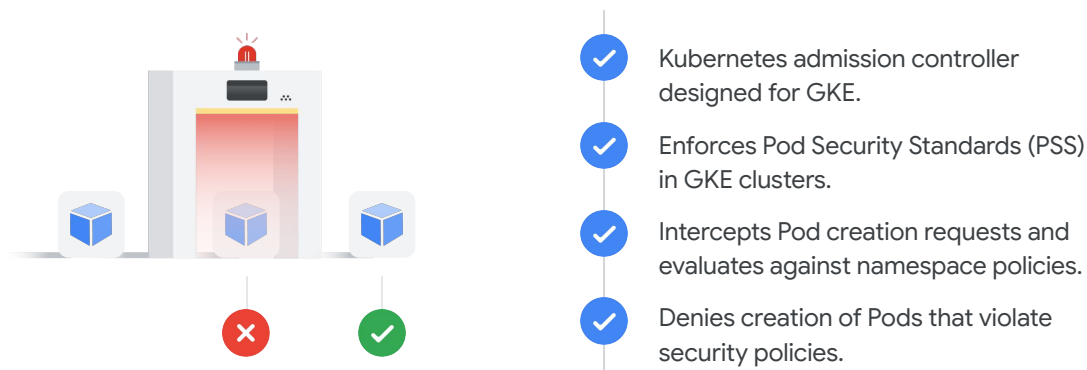
GKE supports three predefined Pod Security Standards levels.

The first level is *privileged*, which grants unrestricted access to the system. While powerful, it can also introduce significant security risks. It's primarily used for Pods that require elevated privileges for tasks like debugging or system administration.

The second level is *baseline*, which offers a balance between security and flexibility, and prevents known privilege escalations while allowing common pod operations. It's recommended for most production workloads.

And the third level is *restricted*, which enforces the most stringent security measures, severely restricting Pod behavior to minimize potential security risks. It's most suitable for Pods that handle sensitive data or run in highly sensitive environments.

Pod Security Admission (PSA)

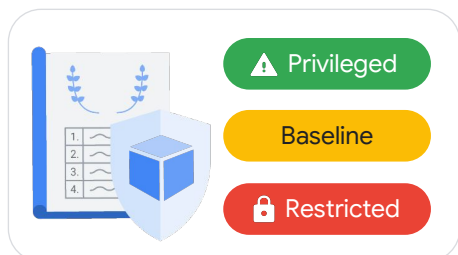


Pod Security Admission is a Kubernetes admission controller *specifically designed for GKE* that uses Pod Security Standards to simplify and automate pod security enforcement.

Pod Security Admission functions as a gatekeeper, intercepting Pod creation requests and evaluating the Pod's security context against the specified PSS policy for the namespace. If the Pod doesn't comply with the policy, the request is rejected, preventing the Pod from being created.

This ensures that only Pods that adhere to the defined security standards are allowed to run within the cluster.

The advantages of combining PSS and PSA



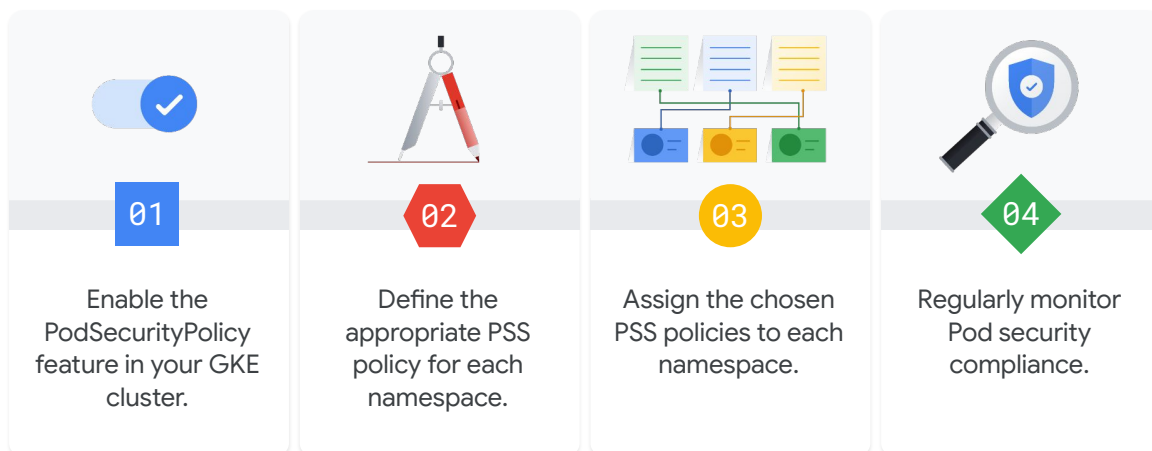
PSS + PSA

- ✓ PSS makes security easier by offering ready-to-use policies.
- ✓ PSA guarantees all Pods follow PSS policies.
- ✓ Stricter Pod security measures reduce attack surfaces.
- ✓ PSS policies can help organizations meet compliance requirements.

Combining Pod Security Standards and Pod Security Admission presents a list of advantages for GKE Pod security. Let's explore that list.

- PSS makes security easier by offering ready-to-use policies, saving you the time of writing your own security rules. This makes security management less complicated and less prone to mistakes.
- PSA enforces consistent security by guaranteeing that all Pods follow PSS policies, keeping your cluster secure and preventing breaches.
- By enforcing stricter Pod security measures, the attack surface is reduced, minimizing potential vulnerabilities and making it more difficult for attackers to exploit weaknesses.
- And finally, adherence to PSS policies can help organizations meet compliance requirements, which ensures that their Kubernetes environments align with industry standards and regulatory mandates.

Steps to implement PSS and PSA



Google Cloud

Now that you know the benefits of PSS and PSA in GKE, let's look at how you actually implement them.

The first step is to **enable the PodSecurityPolicy feature** in your GKE cluster. This feature provides the foundation for PSA to enforce PSS policies.

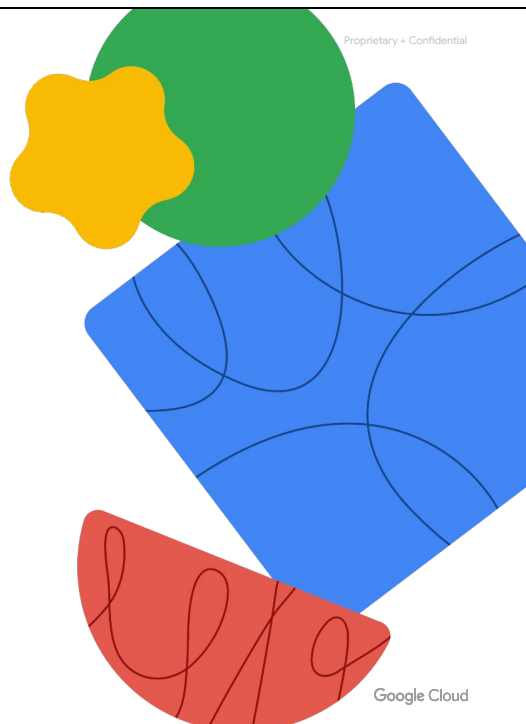
Next, **define the appropriate PSS policy for each namespace** based on its security requirements. This requires selecting the appropriate level of restrictiveness—privileged, baseline, or restricted—for each namespace.

And finally, **assign the chosen PSS policies** to each respective namespace.

After the PSS policies are applied, it's important to **regularly monitor Pod security compliance** and adapt policies as needed. This ensures that Pods remain compliant with the defined security standards, and it allows for policy adjustments to be made when needed.

Quiz questions

Let's pause for a quick check in.



Quiz | Question 1

Question

In a Kubernetes pod, what type of account do containerized processes use to communicate with the kube-apiserver running on the Kubernetes cluster control plane?

- A. A Google Cloud Service Account.
- B. A Kubernetes Service account.
- C. An IAM user account.
- D. A Kubernetes normal user account.

Quiz | Question 1

Answer

In a Kubernetes pod, what type of account do containerized processes use to communicate with the kube-apiserver running on the Kubernetes cluster control plane?

- A. A Google Cloud Service Account.
- B. A Kubernetes Service account.
- C. An IAM user account.
- D. A Kubernetes normal user account.



In a Kubernetes pod, what type of account do containerized processes use to communicate with the kube-apiserver running on the Kubernetes cluster control plane?

A. A Google Cloud Service Account.: This is the **incorrect answer** because Google Cloud service accounts are used at the project level for accessing Google Cloud APIs, not for pod-to-control-plane communication within a Kubernetes cluster.

B. A Kubernetes Service account.: This is the **correct answer** because **Kubernetes Service Accounts** are specifically designed to provide an identity for processes running in a pod, allowing them to authenticate to the Kubernetes API server.

C. An IAM user account.: This is the **incorrect answer** because IAM user accounts are used for human users to interact with Google Cloud resources, not for communication within a Kubernetes pod.

D. A Kubernetes normal user account.: This is the **incorrect answer** because normal user accounts represent human users, not containerized processes.

Quiz | Question 2

Question

You need to implement account controls to grant junior admin users the ability to view details about production clusters and applications, and to fully manage and test lab resources inside your GKE cluster environments. Which account control mechanism is best suited for this granular control?

- A. Radius
- B. IAM
- C. OAuth2
- D. Kubernetes RBAC

Quiz | Question 2

Answer

You need to implement account controls to grant junior admin users the ability to view details about production clusters and applications, and to fully manage and test lab resources inside your GKE cluster environments. Which account control mechanism is best suited for this granular control?

- A. Radius
- B. IAM
- C. OAuth2
- D. Kubernetes RBAC



You need to implement account controls to grant junior admin users the ability to view details about production clusters and applications, and to fully manage and test lab resources inside your GKE cluster environments. Which account control mechanism is best suited for this granular control?

- A. Radius: This is the **incorrect answer** because **Radius** is a networking protocol for Authentication, Authorization, and Accounting. It's not designed for fine-grained authorization within a Kubernetes cluster..
- B. IAM: This is the **incorrect answer** because **IAM** is used for managing access at the cloud project or cluster level, not for fine-grained control of individual Kubernetes resources.
- C. OAuth2: This is the **incorrect answer** because **OAuth2** is an authorization framework, but it's not specifically designed for the granular access control needed within a Kubernetes cluster.
- D. **Kubernetes RBAC**: This is the **correct answer** because **Kubernetes RBAC** allows for fine-grained control of permissions within a cluster, making it possible to grant junior admins view-only access to production resources and full management rights over lab resources..

Quiz | Question 3

Question

Workload Identity is a feature that bridges the gap between what two Google Cloud services?

- A. Kubernetes service accounts and IAM service accounts
- B. Google Kubernetes Engine and IAM
- C. Google Compute Engine and IAM
- D. Kubernetes and IAM

Quiz | Question 3

Answer

Workload Identity is a feature that bridges the gap between what two Google Cloud services?

- A. Kubernetes service accounts and IAM service accounts
- B. Google Kubernetes Engine and IAM
- C. Google Compute Engine and IAM
- D. Kubernetes and IAM



Workload Identity is a feature that bridges the gap between what two Google Cloud services?

- A. Kubernetes service accounts and IAM service accounts:** This is the **correct answer** because Workload Identity is specifically designed to allow Kubernetes service accounts to access Google Cloud APIs.
- B. Google Kubernetes Engine and IAM:** This is the **incorrect answer** because Workload Identity specifically bridges Kubernetes service accounts and IAM service accounts, not the broader services of GKE and IAM.
- C. Google Compute Engine and IAM:** This is the **incorrect answer** because Workload Identity is not directly related to Google Compute Engine. It focuses on authentication within Kubernetes (GKE) using IAM service accounts.
- D. Kubernetes and IAM:** This is the **incorrect answer** because Workload Identity has a narrower scope than the general concept of "access control." .

Quiz | Question 4

Question



How can you use Pod securityContexts to control security? Choose all responses that are correct (2 correct responses).

- A. Limit access to some Linux capabilities, like granting certain privileges to a process, but not to all root user privileges.
- B. Configure audit logging to redirect all Pod logs to an external webhook backend for persistent event auditing.
- C. Limit access to Google Cloud services and resources to prevent Pod users from accessing Cloud Storage objects.
- D. Enable AppArmor, which uses security profiles to restrict individual program actions

Quiz | Question 4

Answer

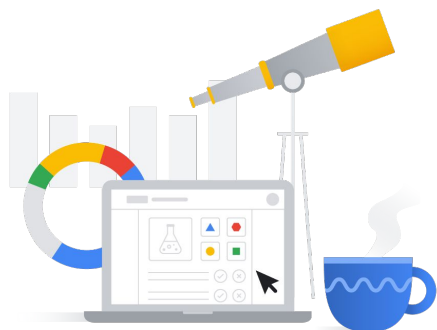
How can you use Pod securityContexts to control security? Choose all responses that are correct (2 correct responses).

- A. Limit access to some Linux capabilities, like granting certain privileges to a process, but not to all root user privileges. 
- B. Configure audit logging to redirect all Pod logs to an external webhook backend for persistent event auditing.
- C. Limit access to Google Cloud services and resources to prevent Pod users from accessing Cloud Storage objects.
- D. Enable AppArmor, which uses security profiles to restrict individual program actions. 

How can you use Pod securityContexts to control security? Choose all responses that are correct (2 correct responses).

- A. Limit access to some Linux capabilities, like granting certain privileges to a process, but not to all root user privileges.:** This is the **correct answer** because you can grant specific capabilities needed by a process without giving it full root access, enhancing security.
- B. Configure audit logging to redirect all Pod logs to an external webhook backend for persistent event auditing.:** This is the **incorrect answer** because audit logging is typically handled by Kubernetes's built-in audit logging feature or external tools, not directly through Pod securityContexts.
- C. Limit access to Google Cloud services and resources to prevent Pod users from accessing Cloud Storage objects.:** This is the **incorrect answer** because access to Google Cloud services is managed through IAM and Workload Identity, not through Pod securityContexts.
- D. Enable AppArmor, which uses security profiles to restrict individual program actions.:** This is the **correct answer** because you can use to enable AppArmor profiles for Pods. AppArmor provides an additional layer of security by defining what system resources individual applications can access.

Lab: Securing Google Kubernetes Engine with IAM and Pod Security Admission



01 Use IAM to control GKE access.

02 Create and use Pod security admission to control Pod creation.

03 Perform IP address and credential rotation.

It's time for some hands-on practice with GKE.

In the lab titled "Securing Google Kubernetes Engine with IAM and Pod Security Admission," you'll:

- Use IAM to control GKE access.
- Create and use Pod security admission to control Pod creation.
- And finally, you'll perform IP address and credential rotation to complete the lab.