Google Cloud

Security features of Connect for Anthos

Google Cloud

Introduction

This document explains the security measures built into Connect for Anthos customers who are interested in ensuring the security of their GKE On-Prem user clusters.

An effective hybrid and multi-cloud platform delivers central management, observability, and access to services across environments. Anthos provides a uniform and comprehensive experience across those capabilities, no matter what Kubernetes provider you leverage. Connect is a lightweight agent to provide the following at economies of scale, and across providers:

- Multi-cluster management and cluster visibility.
- Application services deployment and lifecycle management.

Beta

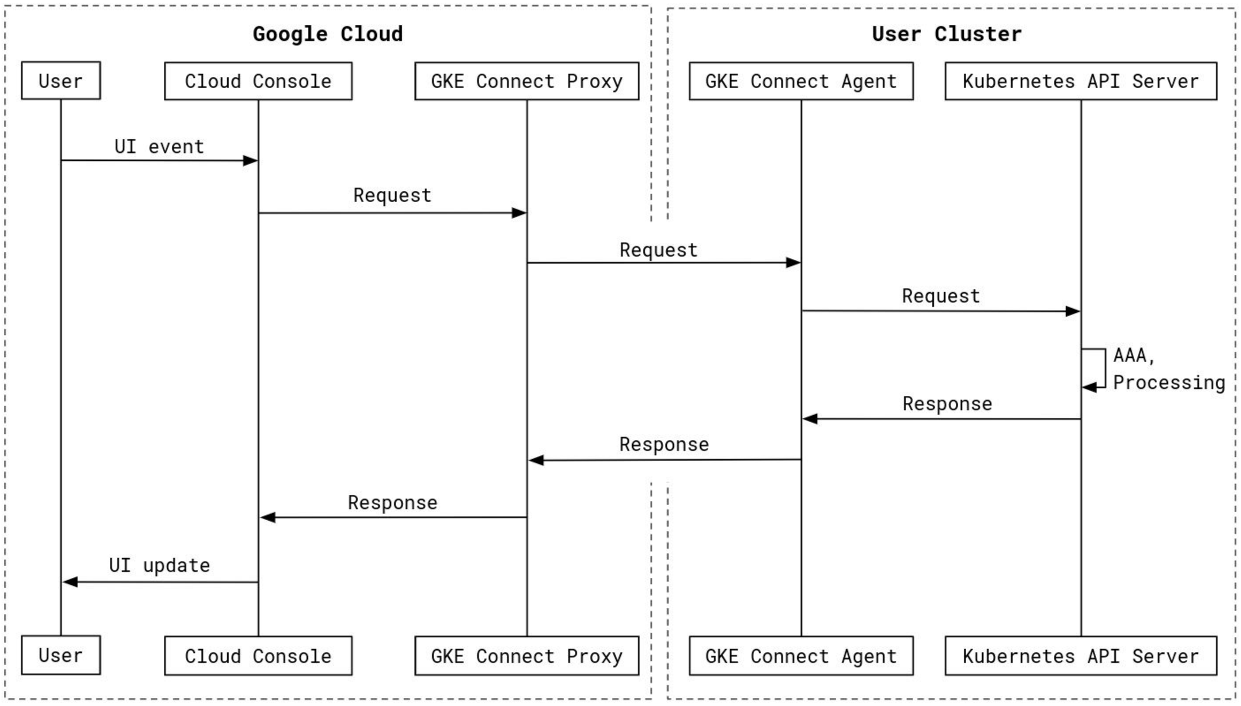
This product or feature is in a pre-release state and might change or have limited support. For more information, see Product launch stages.

This document discusses the following:

- How Connect's design puts security first.
- Best practices for the Connect agent deployment.
- How to improve your Kubernetes deployment security posture.

Architecture of Connect

Connect allows end-users and Google Cloud Platform (GCP) services to access Google Kubernetes Engine API (GKE API) servers that aren't on the public internet. The Connect agent runs in the Kubernetes cluster (one agent per cluster), and connects to a Connect proxy. GCP services that need to interact with the GKE cluster connect to the proxy, which forwards requests to the agent. The agent, in turn, forwards them to the GKE API server as depicted in the following diagram.



When the agent is deployed, it establishes a persistent TLS 1.2 connection to GCP to wait for requests. GCP services, when enabled by admins, can generate requests for your Kubernetes clusters. These requests may also come from direct user interaction with the GCP Console.

The GCP service sends the request to the proxy. The proxy then forwards the request to the deployed agent responsible for a cluster, and finally the agent forwards the request to the GKE API server. The GKE API server applies Kubernetes authentication, authorization, and audit-logging policies, and returns a response. The response is passed back through the agent and the proxy to the GCP service. At each step in the process, components perform per-connection and per-request authentication and authorization.

The GKE API server applies the same authentication, authorization, and audit-logging policies to all requests, including requests through Connect. This process ensures that you're always in control of the access to your cluster.

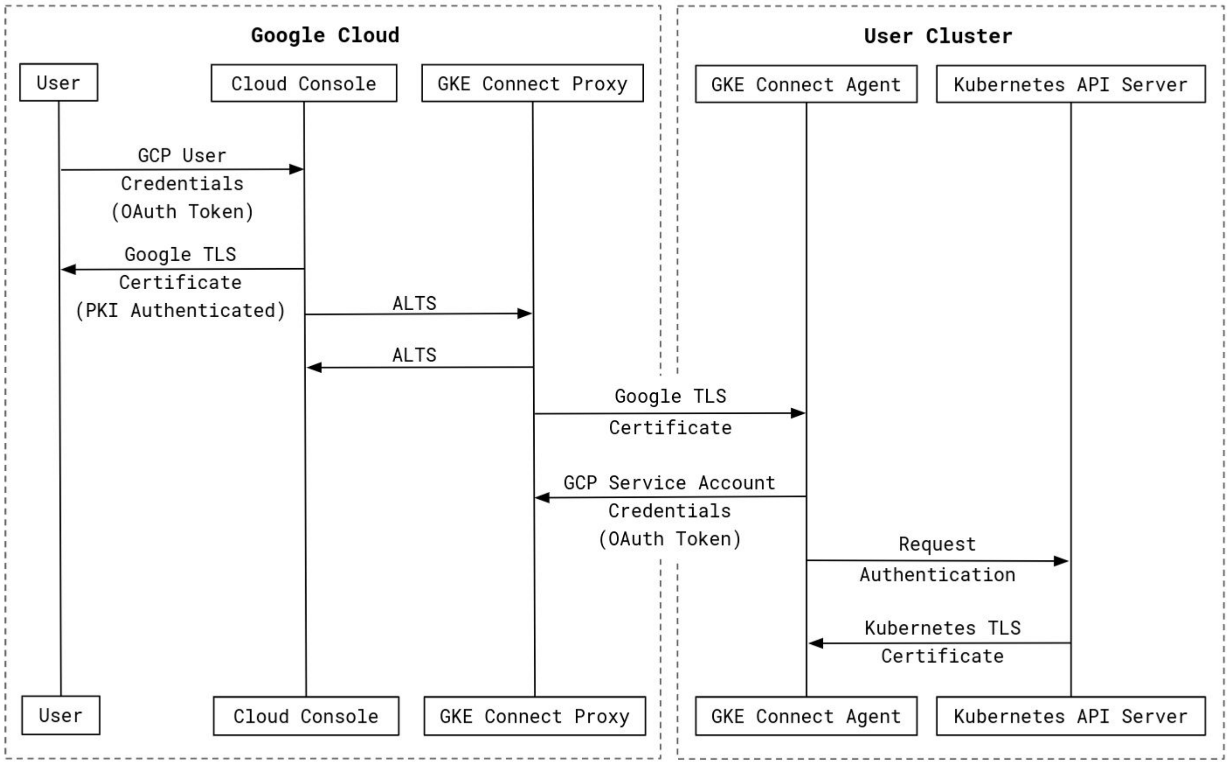
Connect and defense-in-depth

Defense-in-depth is intrinsic to everything GCP does within its infrastructure and security practices. We take a layered approach to every aspect of securing our organization and our customers in order to protect valuable data, information, and users. This is the same principle by which we've designed Connect.

In addition to Google's own defense-in-depth strategy and design, admins should evaluate the content provided here alongside your security posture and standards. This section calls out additional measures that administrators can take that complement Kubernetes hardening best practices.

Component-to-component security

Each component of a Connect request authenticates its peers, and only shares data with peers that are both authenticated and authorized for that data, as illustrated in the following diagram.



Each component of a Connect request uses the following to authenticate each other:

- Transport Layer Security (TLS)
- Application Layer Transport Security (LTS)
- OAuth

Each component of a Connect request uses to following to authorize each other:

- Cloud Identity and Access Management (Cloud IAM)
- Whitelists

Each connection between the Kubernetes cluster and GCP is encrypted, and at least one peer of each connection uses certificate-based authentication. This process helps to ensure that all token credentials are encrypted in transit, and only received by authenticated and authorized peers.

User authentication to GCP

When using the GCP Console, users go through the standard Google login flow. GCP provides a TLS certificate that the user's browser authenticates, and the user logs in to a GCP or GSuite account to authenticate to GCP.

Access to a project through the GCP Console or other APIs is controlled by Cloud IAM permissions.

GCP service-to-service authentication

GCP uses ALTS for internal service-to-service authentication. ALTS allows GCP services, such as the proxy, to create an authenticated, integrity-protected connection.

GCP services must be internally authorized to use the proxy to connect to a remote Connect instance because the proxy uses a whitelist of service identities to limit access.

Authenticating GCP

The agent requires connectivity to the following endpoints to communicate with GCP when you're installing or upgrading the agent:

- `Cloudresourcemanager.googleapis.com` resolves metadata regarding the GCP project the cluster is being connected to.
- `Gkehub.googleapis.com` creates GCP-side membership information for the cluster you're connecting to GCP.

The agent requires connectivity to the following endpoints to communicate with GCP during normal operation:

- `Oauth2.googleapis.com` obtains short-lived OAuth tokens for agent operations against `gkeconnect.googleapis.com`.
- `Gkeconnect.googleapis.com` establishes the channel used to receive requests from GCP and issues responses.
- `www.googleapis.com` authenticate service tokens from incoming GCP service requests.

The agent uses TLS to authenticate and encrypt each connection. The agent authenticates GCP TLS certificates by using a set of root certificates built into the binary, to avoid inadvertently trusting certificates added to the agent's container. The agent only executes API calls against correctly authenticated endpoints. This process helps to ensure that service account certificates and the GKE API requests are sent by GCP, and that any responses are sent only to GCP.

For the list of domains that the agent communicates with during normal operation, see Ensure network connectivity

You can configure the agent to connect to GCP through an HTTP proxy. In this configuration, the agent uses the `HTTP/1.1 CONNECT` against the HTTP proxy, and establish a TLS connection to GCP. The HTTP proxy only sees the encrypted traffic between the agent and GCP. The end-to-end integrity and security of connection between the agent and GCP is unaffected.

Authenticating the agent

The agent authenticates to GCP by using a GCP service account that you create. When the cluster admin deploys the agent, they provide a private key for this service account, and takes responsibility for the key's privacy. When the agent connects to GCP, it authenticates with this service account, and asks to receive requests for its configured project.

GCP authenticates the service account credentials, and also checks that the GCP service account has the `gkehub.endpoints.connect` IAM permission in the project. This permission is usually granted through the `gkehub.connect` role. Without this permission, the agent's request is denied and it can't receive requests from GCP.

GKE API server

The agent uses the Kubernetes client library to create a TLS connection to the GKE API server. The Kubernetes runtime provides the agent's container with a TLS certificate authority (CA) certificate that the agent uses to authenticate the API server.

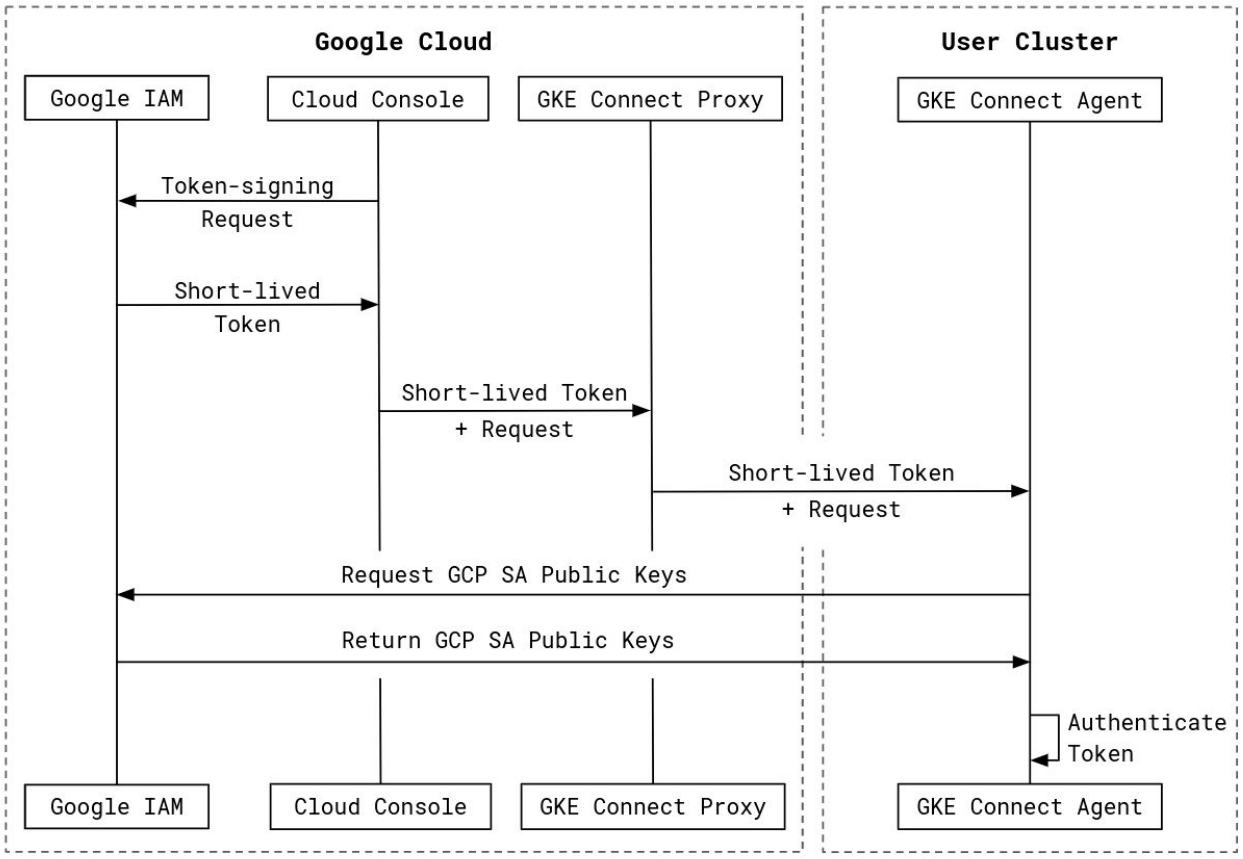
The API server authenticates each request separately, as described in the next section.

Request security

Each request sent from GCP through Connect includes credentials that identify the request's sender: both the GCP service that generated the request, and (where applicable) the end user for whom the request is sent. These credentials allows the GKE API server to provide authorization and auditing controls for each request.

Service-to-agent authentication

Each request sent to the agent includes a short-lived token identifying the GCP service that sent the request, as illustrated in the following diagram.

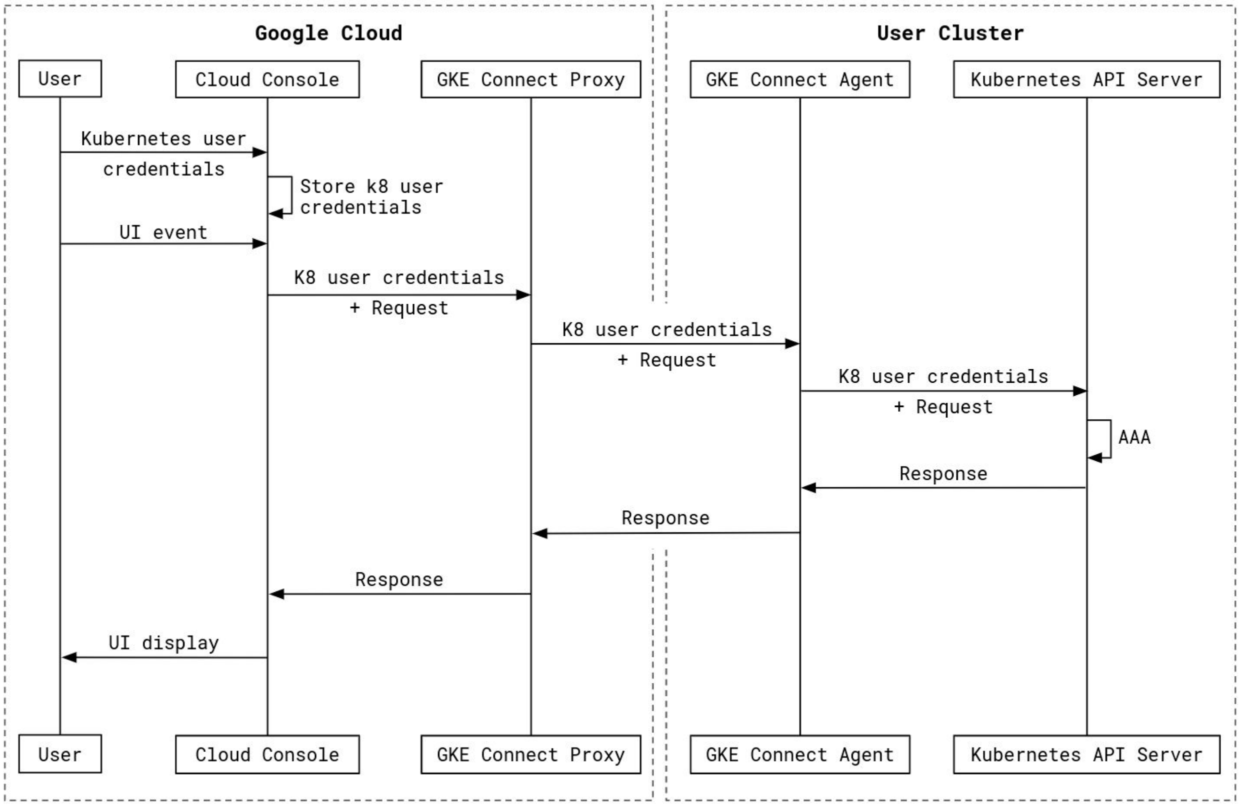


The token is signed by a GCP service account associated exclusively with the GCP service. The agent fetches the service account's public keys to validate the token. This token isn't forwarded to the API server.

The agent validates GCP certificates using CA roots embedded in the binary. This process helps to ensure that it is receiving authentic and unaltered requests from GCP.

End-user authentication

GCP services that access clusters on behalf of a user require that user's credentials to authenticate to the API server, as illustrated in the following diagram.



This policy helps to ensure that the same set of permissions are applied to the user when accessing through Connect. Some GCP services authenticate to the API server on behalf of a user. For example, a user can access the GCP Console to view workloads in Connect-enrolled clusters. When a user accesses these services, they provide credentials that the GKE API server recognizes: a username and password, or any of the tokens that the GKE API server supports.[1]

[1] Connect does not support client certificate authentication.

The GCP Console stores these credentials as part of a user's profile. These credentials are encrypted at rest, are only accessible with the user's GCP or GSuite credentials, and are only used for connections through Connect. These credentials cannot be downloaded again. The credentials are deleted when the user logs out of the cluster, when the cluster registration is deleted in GCP, when the project is deleted, or when the user account is deleted. For more information, see Data deletion on GCP.

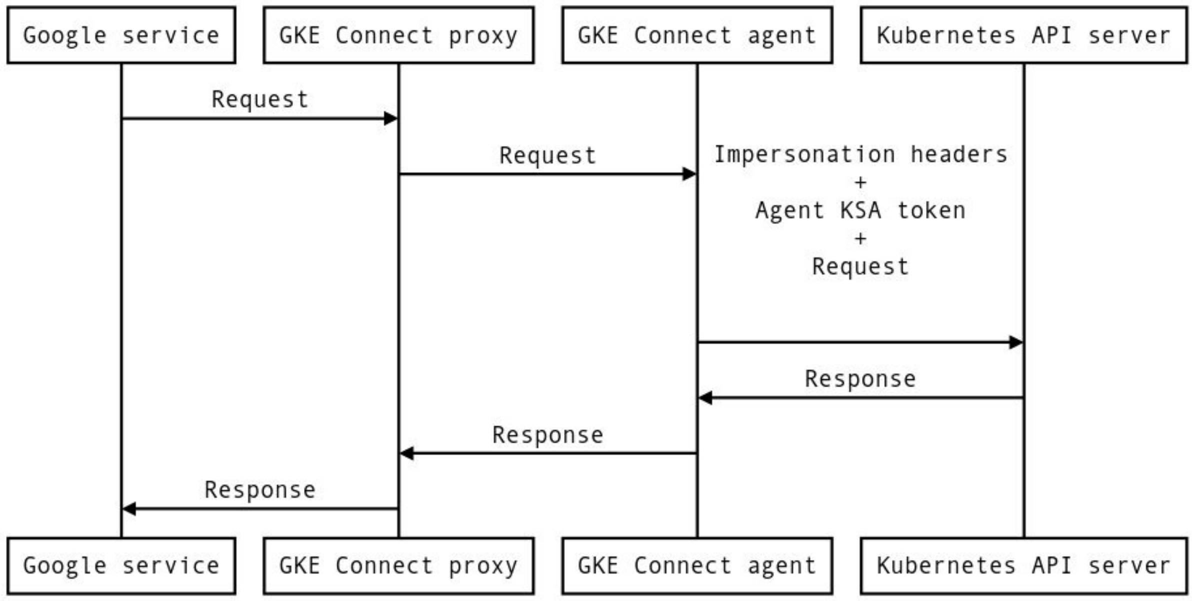
When a user interacts with the GCP Console, it generates requests for the GKE API server. The service sends the user's credentials along with the request through Connect. The agent then presents the request and credentials to the GKE API server.

The GKE API server authenticates the user's credentials, performs authorization on the user's identity, produces an audit event for the action (if configured), and returns the result. Because the user-provided credentials are used to authenticate the request, the GKE API server applies the same authorization and auditing policy for Connect requests as it does for other requests.

Service-to-Kubernetes authentication

GCP services that access the GKE API server outside of a user's context use Kubernetes impersonation to authenticate to the GKE API server. This method allows the GKE API server to provide per-service authorization checks and audit logging, as illustrated in the following diagram.

Google Cloud



Services at GCP can use Connect outside of a user's context. For example, a multicluster ingress service can automatically synchronize ingress resources across clusters. These services don't have credentials that the GKE API server can authenticate: most API servers aren't configured to authenticate GCP service's credentials. However, an API server can delegate limited authentication privileges to another service through impersonation, and the agent can authenticate GCP services sending requests through Connect. Together, these allow requests through the agent to authenticate as GCP service accounts.

When a GCP service sends a request on its own behalf (rather than in a user's context), the agent adds its own Kubernetes credentials, and Kubernetes impersonation headers that identify the GCP service, to the request. The impersonation headers claim a user[2] name of the GCP service account authenticated by the agent.

The GKE API server authenticates the agent's credentials, and also checks that the agent can impersonate the GCP service account. The ability to impersonate is typically controlled by role-based access control (RBAC) rules, and can be limited to specific identities, such as GCP service accounts.

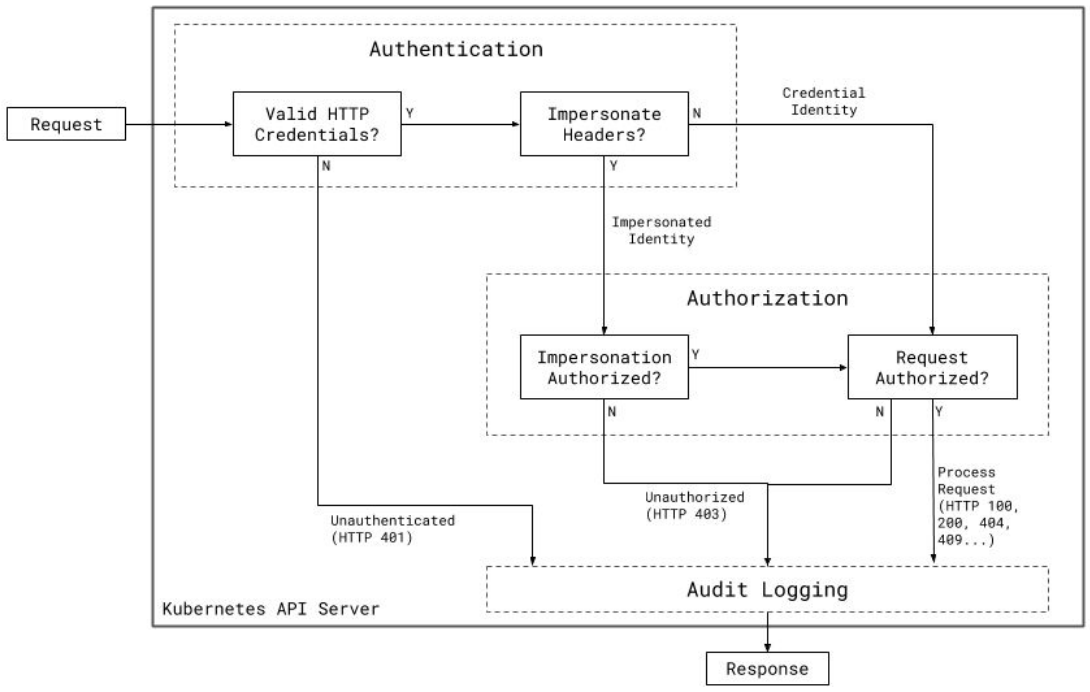
If the agent is authorized to impersonate the requested identity, the API server then performs authorization checks for the GCP service account, and serves the request. The

[2] Note that this does not require an additional Kubernetes service account; the GCP service account is claimed as a user identity.

audit log for the request includes both the agent's identity and the impersonated GCP service account.

In-cluster security

The agent ultimately sends GKE API requests to the GKE API server, as illustrated in the following diagram.



The GKE API server authenticates, authorizes, and audit-logs these requests, just as it does for all other requests it serves.

As a proxy for these requests, the agent has access to sensitive data, such as credentials, requests, and responses. Kubernetes, and the Kubernetes ecosystem, provide a set of tools to prevent other actors from getting that access, and for helping to ensure that the agent only accesses what it's supposed to.

GKE authentication

The GKE API server authenticates the sender of each incoming request to determine what permissions to apply in the authorization stage. As previously described, the request either includes a user's credentials, or includes the agent's Kubernetes credentials and impersonation headers.

Cluster admins remain in control of authentication mechanisms recognized by the GKE API server. Admins might be able to revoke a user's credentials, and can revoke or reduce the privilege of the agent's credentials.

GKE authorization

The GKE API server checks that the authenticated identity is allowed to take the requested action on the requested resource.

The cluster admin can use any of the Kubernetes authorization mechanisms to configure authorization rules. Connect doesn't perform any authorization checks on behalf of the cluster.

Agent security

The agent has access to its own (Kubernetes and GCP) credentials, as well as the credentials, requests, and responses that pass through it. As such, the agent occupies a trusted position in a connected cluster.

The agent is designed with the following security fundamentals:

- The agent is written in Go, which provides garbage-collected memory management, and prevents many unsafe memory operations.
- The agent is deployed in a distroless container image. The agent's image doesn't include a shell, libc, or other code that is extraneous to the agent's execution path.
- The agent's image is built by Google's shared build infrastructure from checked-in code. Only this build system can deploy agent images to Container Registry. GCP developers cannot deploy new images on their own. This process helps to ensure that all edits to the agent's source can be traced back to an author and reviewer for non-repudiation.

The agent runs as a standard deployment in a Kubernetes cluster that deploys at the time that you register your cluster. As a result, all of the options and best practices available for monitoring and securing deployments, `ReplicaSets`, and pods are available for the agent.

These mechanisms are designed to make it difficult to compromise the agent container. However, privileged access to the agent's node can still compromise the agent's environment, therefore it is important for administrators to follow standard Kubernetes security guidelines for protecting cluster infrastructure.

What's next

- Check out more about Anthos.
- Try out other Google Cloud Platform features for yourself. Have a look at our tutorials.