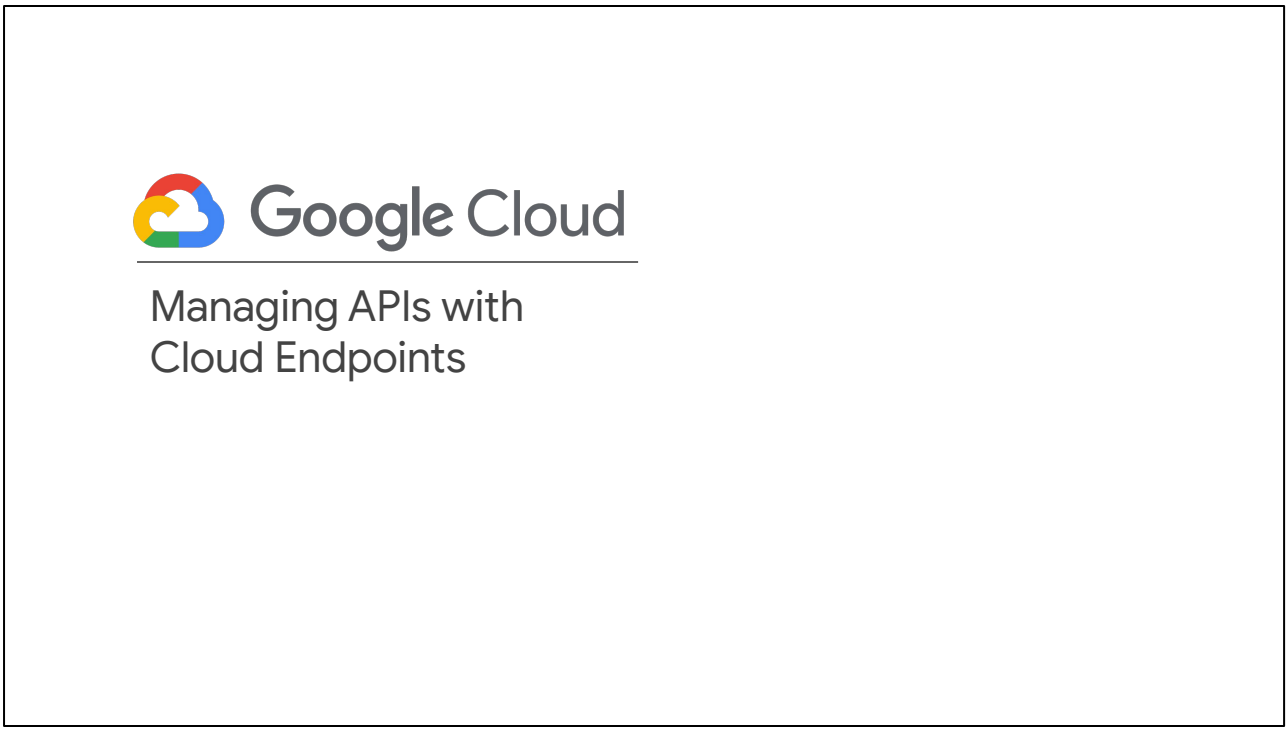
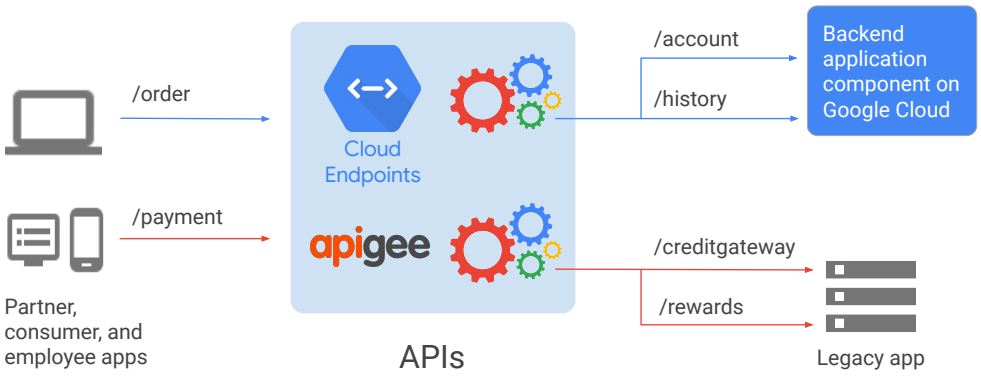


Let's say you have some functionality that you wish to expose consumers as an API. That can be challenging to deploy and manage that API on your own. Cloud Endpoints enables you to deploy and manage the API on Google Cloud. In the module: Managing APIs with Cloud Endpoints, you'll learn how to develop an OpenAPI configuration for your rest API. You'll learn techniques to secure and restrict access to the API, deploy new versions of the API, and deploy the API to test and production environments. You'll also learn how to monitor metrics for your API.

Implement an API gateway to make backend functionality available to consumer applications



Google Cloud

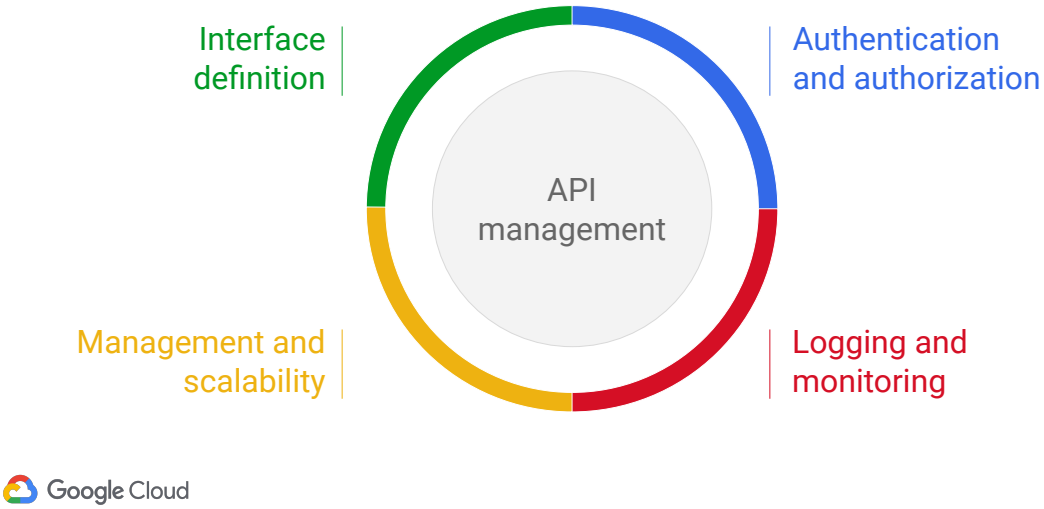
An API gateway enables clients to retrieve data from multiple services with a single request. An API gateway creates a layer of abstraction and insulates the clients from the partitioning of the application into microservices.

You can use Cloud Endpoints to implement API gateways. Additionally, the API for your application can run on backends, such as App Engine, Google Kubernetes Engine (GKE), or Compute Engine.

If you have legacy applications that cannot be re-factored and moved to the cloud, consider implementing APIs as an adaptor layer or façade. Each consumer can then invoke these modern APIs, instead of invoking outdated APIs using old protocols and disparate interfaces.

Using the Apigee API platform, you can design, secure, analyze, and scale your APIs for legacy backends.

Deploying and managing APIs can be difficult



Google Cloud

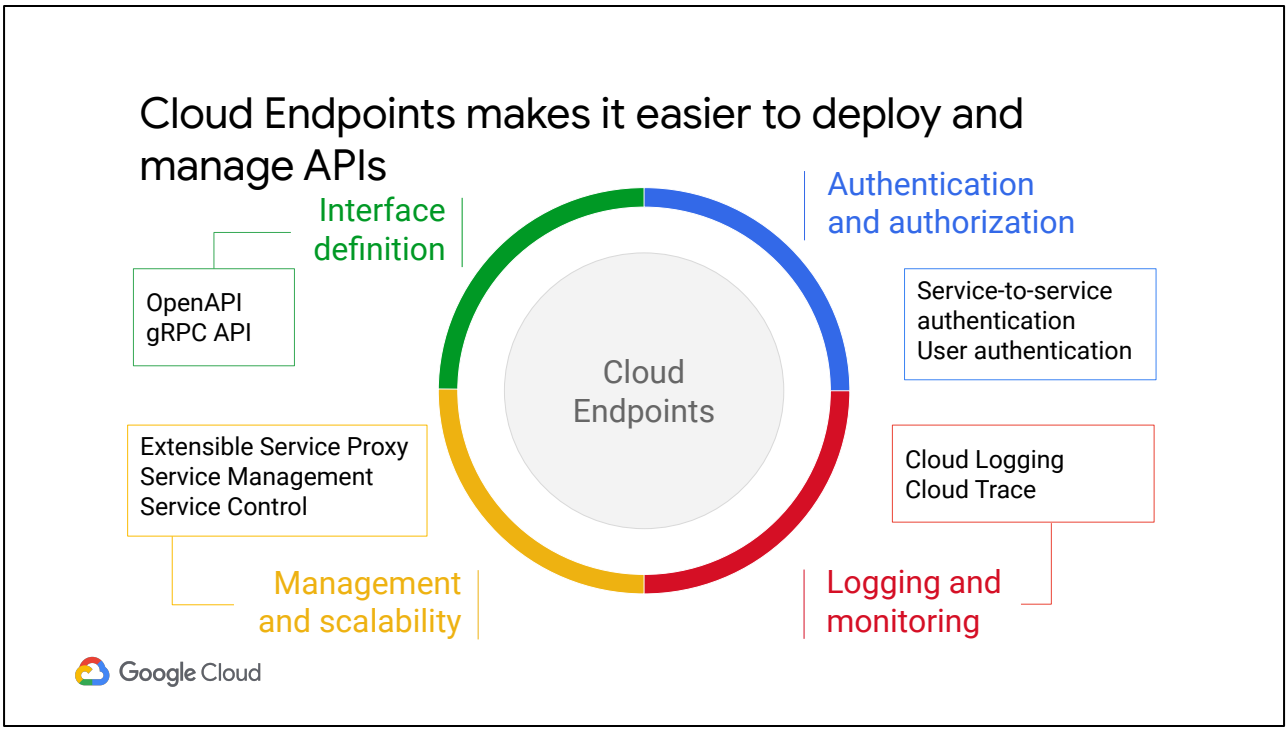
There are a few issues to consider when deploying and managing APIs on your own.

What language and format will you use to describe the interface? For example, will you use a format such as WSDL or OpenAPI, to express the schema and constraints of the API?

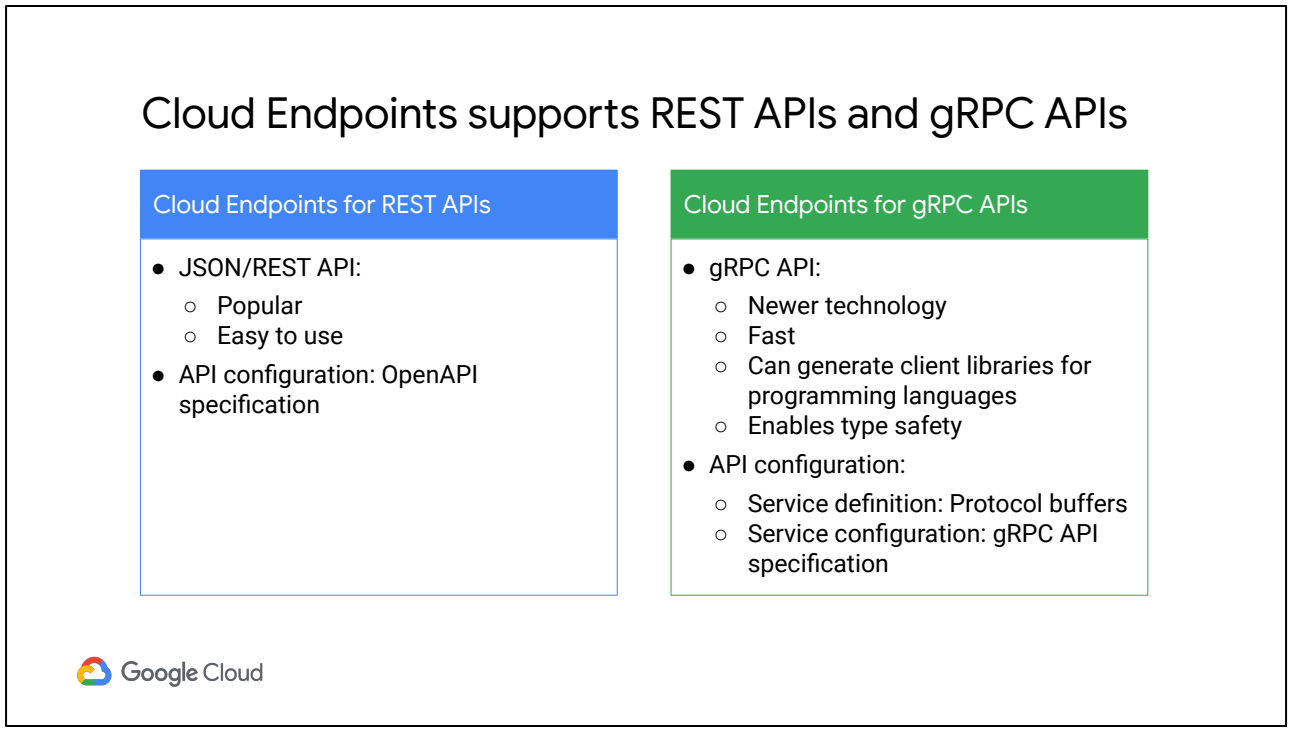
How will you authenticate services and users who invoke your API?

How will you ensure that your API scales to meet demand?

Does your infrastructure log details about API invocations and provide monitoring metrics?



Cloud Endpoints provides the infrastructure support that you need to deploy and manage robust, secure, and scalable APIs. Cloud Endpoints supports the OpenAPI specification and gRPC API specification. Cloud Endpoints supports service-to-service authentication and user authentication with Firebase, Auto zero, and Google authentication. The Extensible Service Proxy, Service Management, and Service Control APIs together validate requests, log data, and handle high volumes of traffic. Using Cloud Endpoints, Cloud Logging, and Cloud Trace, you can view detailed logs, trace lists, and metrics related to traffic volume, latency, size of requests and responses, and errors.



JSON HTTP 1.1-based REST APIs are popular and easy to use. To enable Cloud Endpoints for REST APIs, create the API configuration in a YAML file, based on the OpenAPI specification.

gRPC is a newer, faster technology. You can generate client libraries for various programming languages. Your application can then make type-safe remote server calls as if they were local calls. To enable Cloud Endpoints for gRPC APIs, create your service definition by using protocol buffers, and then create a service configuration by using the gRPC API specification.

Cloud Endpoints supports transcoding of HTTP JSON calls into gRPC. Your clients can access your gRPC API while using plain old HTTP JSON calls. In the rest of this module, we'll focus on JSON-based REST APIs.

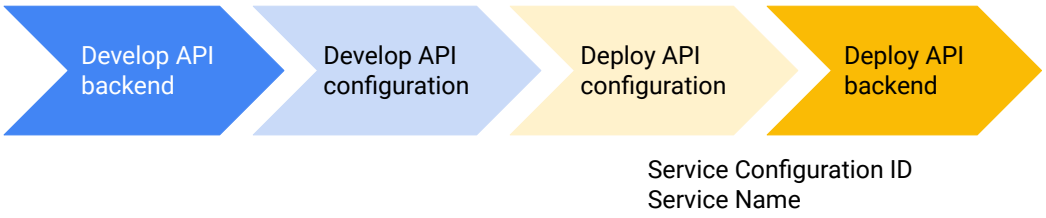
For more information about Cloud Endpoints for REST APIs, see https://cloud.google.com/endpoints/docs/openapi/open-api-spec.

For more information about gRPC, see:

- gRPC: https://grpc.io/
- Announcing gRPC Alpha for Google Cloud Pub/Sub: https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub
- Cloud Endpoints for gRPC APIs: https://cloud.google.com/endpoints/docs/grpc/about-grpc

- Transcoding HTTP/JSON to gRPC: https://cloud.google.com/endpoints/docs/grpc/transcoding

Develop and deploy your API configuration and API backend



Google Cloud

Let's dive into Cloud Endpoints for REST APIs.

This diagram illustrates the high-level steps to make your API backend available as Cloud Endpoints API.

After you develop your REST API backend, create an API configuration file that describes your Cloud Endpoints API.

Deploy the API configuration by using the “gcloud service management deploy” command. The gcloud command returns the service configuration ID and service name. Specify this service configuration ID and service name in your API’s backend configuration files, such as the “app.yaml” file for App Engine flexible environment deployments.

Finally, deploy the API backend.

Use the OpenAPI specification as the Interface Definition Language

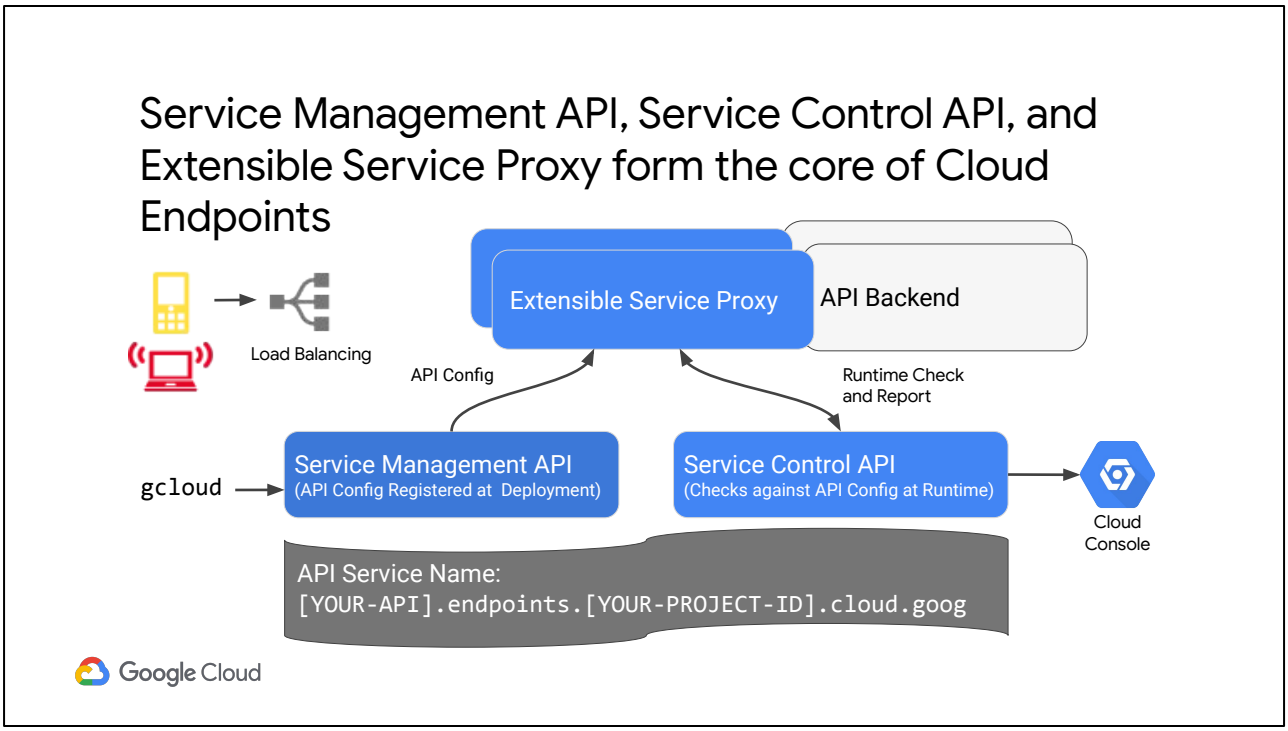
```
swagger: "2.0"
info:
  description: "My new API."
  title: "Cloud Endpoints Example"
  version: "1.0.0"
host: "quiz-api.endpoints.YOUR-PROJECT-ID.cloud.goog"
basePath: ...
paths: ...
securityDefinitions: ...
```

Create an API configuration file that describes your cloud endpoints API. The API configuration is a YAML file that describes the API using the OpenAPI specification. The OpenAPI specification and its extensions enable you to describe the surface of the API and security definitions. You can specify security definitions for user authentication and service-to-service authentication.

Various tools are available to help you create and manage your OpenAPI specification. For more information about the schema of the objects in the OpenAPI Specification, see: https://github.com/OAI/OpenAPI-Specification.

Various tools are available to help you create and manage your OpenAPI specification. For more information, see:

- Swagger Editor: http://editor.swagger.io/
- Swagger Tools and Integrations: https://swagger.io/open-source-integrations/
- OpenAPI / Swagger Resource List for API Developers: https://blog.runscope.com/posts/openapi-swagger-resource-list-for-api-developers



When you deploy the API configuration, it is registered with the Service Management API and shared with the Extensible Service Proxy.

Service management uses the host value in your deployment configuration file to create a new Cloud Endpoints service with a name in the format shown here, that is “your API name.endpoints.yourprojectid.cloud.goog”. This API service name is created, if it doesn't already exist, and then Cloud Endpoints configures the service according to your OpenAPI configuration file. Cloud Endpoints uses DNS-compatible names to uniquely identify services. Because projects in Google Cloud are guaranteed to have a globally unique name, you can use your project name to create a unique API service name, such as “`quizapi.endpoints.myprojectid.cloud.goog` ”. You can also map your own DNS name to your API.

The Extensible Service Proxy is an NGINX [engine X] based proxy that runs in front of the API backend and injects Cloud Endpoints functionalities such as authentication, monitoring and logging. The proxy uses techniques such as heavy caching and asynchronous calls to remain lightweight and highly performant.

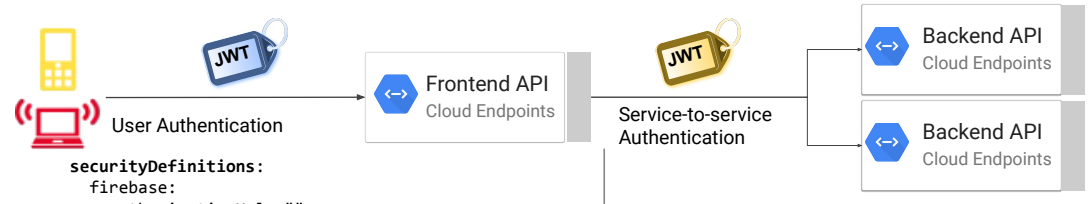
At runtime, Cloud Endpoints can receive calls from any source such as mobile applications, web applications and other services. The calls are load balanced and routed to the Extensible Service Proxy. The Extensible Service Proxy works with the Service Control API to check the request against the API configuration and verify that the request can be passed through to the backend. If the request authenticates

successfully, the Extensible Service Proxy passes it on to the API backend. The Service Control API logs information about incoming requests. These log messages and metrics can be viewed by using the Cloud Console.

For more information, see:

- Cloud Endpoints Architectural Overview: https://cloud.google.com/endpoints/docs/openapi/architecture-overview
- Naming Your API: https://cloud.google.com/endpoints/docs/openapi/naming-your-api-service

Enable user authentication and service-to-service authentication




```
securityDefinitions:
  firebase:
    authorizationUrl: ""
    flow: "implicit"
    type: "oauth2"
    # Replace YOUR-PROJECT-ID with your project ID
    x-google-issuer:
"https://securetoken.google.com/YOUR-PROJECT-ID"
    x-google-jwks_uri:
"https://www.googleapis.com/service_accounts/v1/metad
ata/x509/securetoken@system.gserviceaccount.com"
    x-google-audiences: "YOUR-CLIENT-ID"
security:
  - firebase: []
```

- Create a service account for service-to-service authentication.
- Use a Google ID Token to sign the token.

Google Cloud

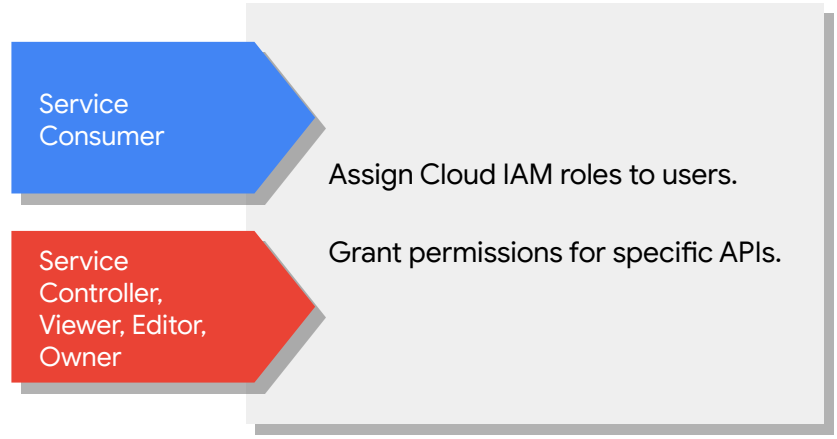
With Cloud Endpoints API, you can authenticate users who are attempting to invoke your frontend APIs. Cloud Endpoints supports user authentication with Firebase, Auth0 *[auth zero]*, Google authentication, and other custom authentication methods. After the user signs in, the authentication providers send a signed JSON Web Token (or JWT, pronounced “jot”) to Cloud Endpoints. Cloud Endpoints checks that the JWT is signed by the provider that you specified in your API configuration.

For service-to-service authentication, create a service account. Use a Google ID token to sign the request.

For more information, see:

- Authenticating service-to-service calls with Google Cloud Endpoints (Google Cloud Next '17): https://www.youtube.com/watch?v=4PgX3yBJEyw
- Service-to-service authentication: https://cloud.google.com/endpoints/docs/openapi/service-to-service-auth

You can restrict API access

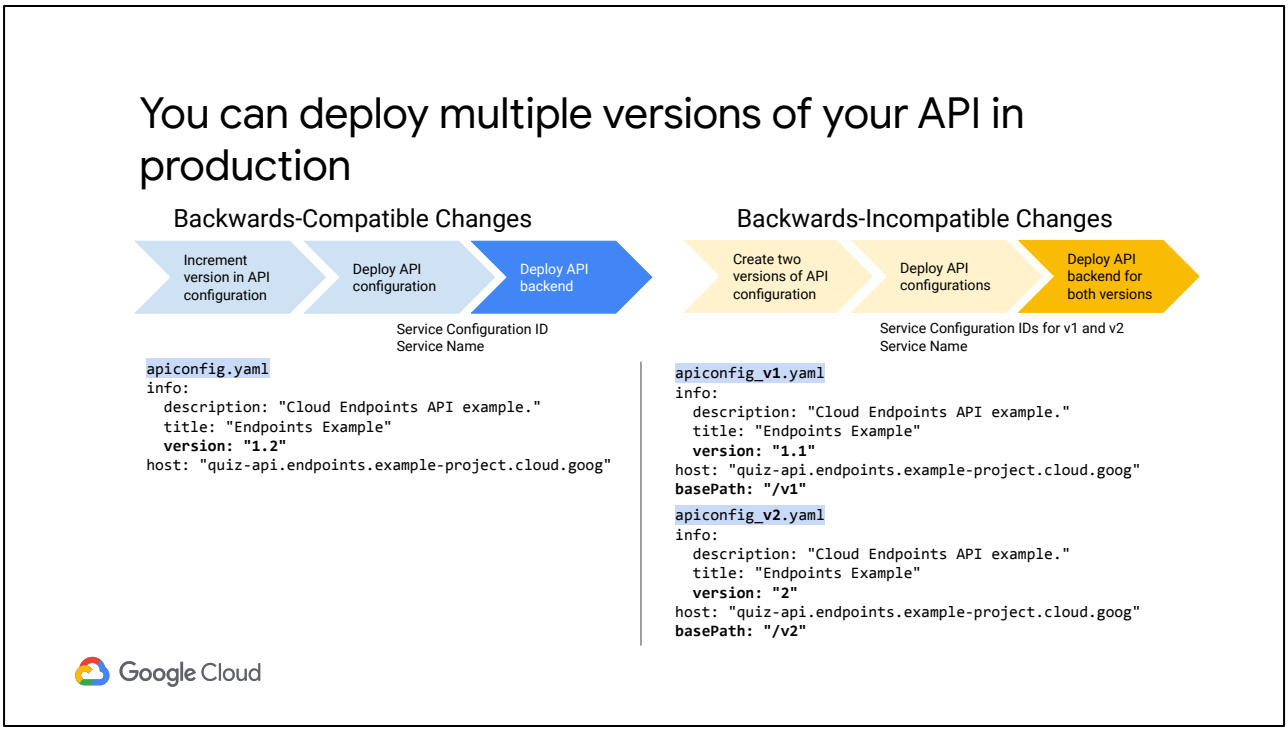


Google Cloud

To restrict access to your Cloud Endpoints API, you can assign Cloud Identity and Access Management (Cloud IAM) roles to specific users. You can grant access to a specific API or to the entire Cloud project.

To give users the ability to enable your service in their own cloud project and invoke its APIs, assign the Service Consumer role to them. This is the most common use-case. You can assign the Service Controller, Viewer, Editor, or Owner roles to give users greater permissions to view and manage service configurations and projects.

For more information about granting API access, see https://cloud.google.com/endpoints/docs/openapi/api-access-overview.



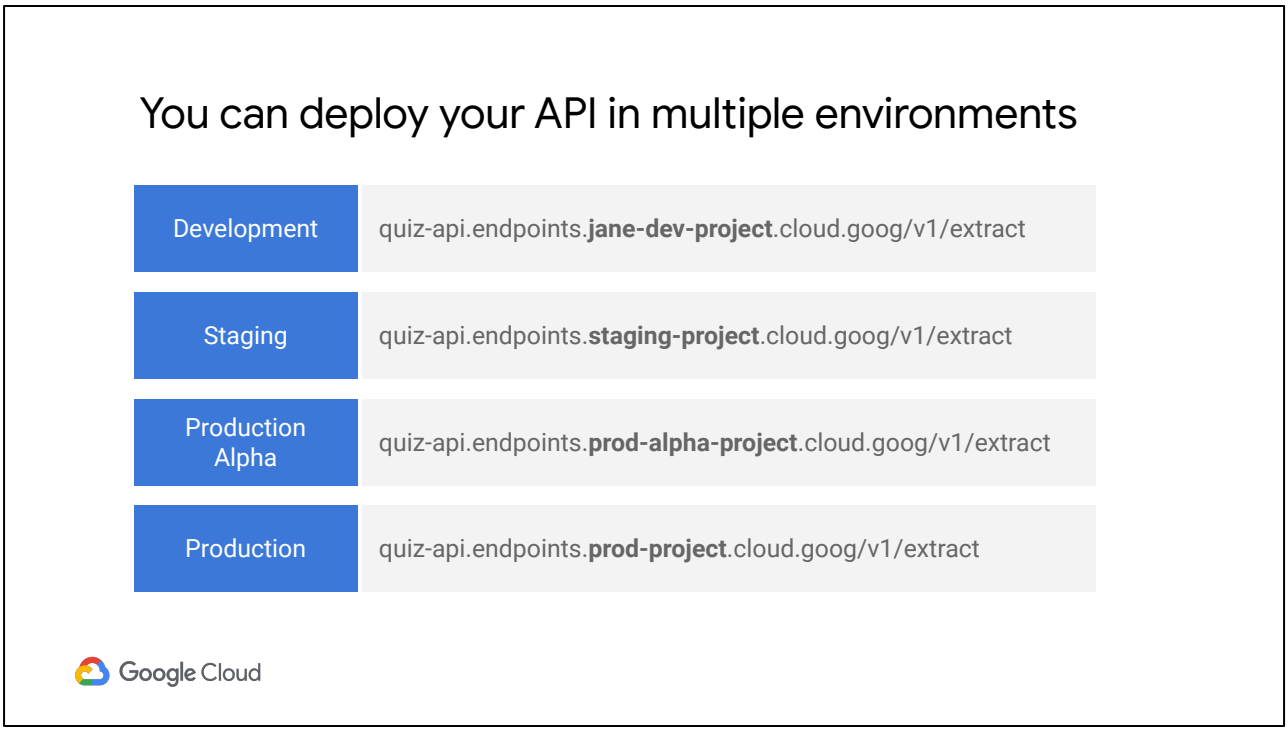
You might make small changes, bug fixes and performance enhancements to your API backend. These changes are usually backwards compatible. In such cases, it's a good practice to increment the version attribute in the Cloud Endpoints API configuration and then redeploy the API configuration and the API backend. In the example shown, we've increased the version number from 1.1 to 1.2 because this is a backwards-compatible change.

If you make changes that are not backwards compatible and will break functionality for consumers of your API, deploy two versions of your Cloud Endpoints API by creating a separate API configuration for each version. The “g-cloud service management deploy” command will return a different service configuration ID for each version. Update the API backend configuration of each version with a corresponding service configuration ID. In the example shown, we have different API configurations for version one of the API and version two of the API.

You can also delete versions of your API when they're no longer in use. Make sure to announce your deprecation and phase out plans to the consumers of the API well in advance. This will give them time to evaluate and migrate to newer versions of your API.

For more information, see:

- Versioning an API: https://cloud.google.com/endpoints/docs/versioning-an-api
- Deleting an API and API instances: https://cloud.google.com/endpoints/docs/openapi/deleting-an-api-and-instances

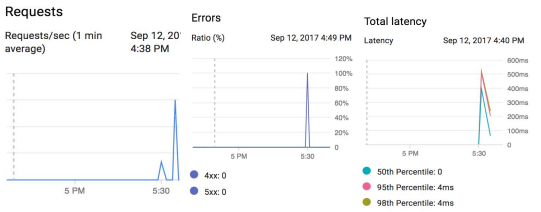


When developing and deploying APIs, you will have separate environments for developing and staging limited releases, such as private alphas, and for production.

You can create separate projects for each environment and deploy a Cloud Endpoints API and backend with separate endpoints for your consumers. For example, developers can run their unit tests against the development environment. You can run integration tests against the staging endpoint. A separate project for private alphas provides a higher degree of isolation from other environments.

For more information about naming conventions for projects, see https://cloud.google.com/endpoints/docs/openapi/planning-cloud-projects.

You can view error logs and metrics from the Cloud Endpoints dashboard

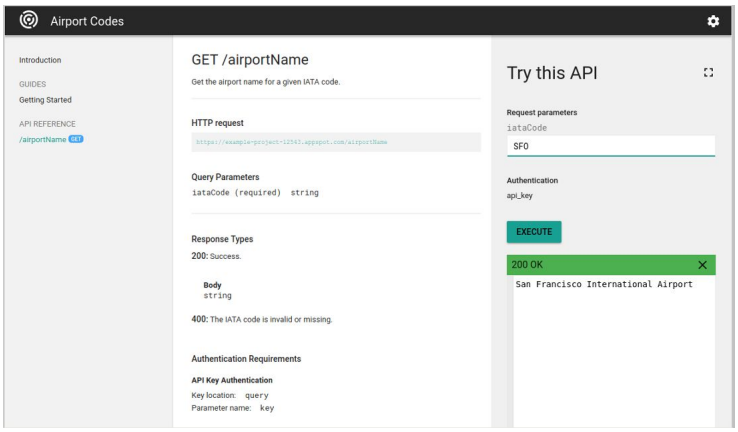


Method	Requests	Avg req size	Avg resp size	4xx	5xx	Latency 98%	Latency 95%	Latency median
GET /api/quizzes/*	10	430 B	509 B	0	0	472 ms	393 ms	66 ms

Google Cloud

In the Cloud Endpoints dashboard, you can see metrics related to requests, 4xx and 5xx errors, and latency. You can also view logs in Cloud Logging to see detailed information about requests to each API.

You can allow users to test and explore your API using an Endpoint Developer Portal



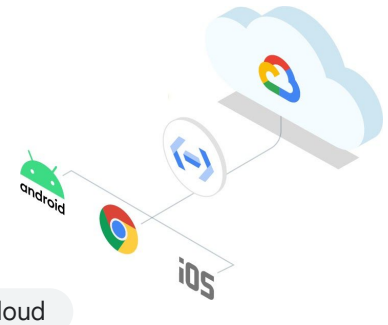
Google Cloud

You can use Cloud Endpoints Portal to create developer portals, websites that users of your Cloud Endpoints APIs can access to explore and test your APIs. On your portals, developers who are using your APIs in their own code can find the SmartDocs API reference documentation for your APIs.

SmartDocs uses the OpenAPI document that Cloud Endpoints Frameworks creates to generate API reference documentation. SmartDocs includes a Try this API panel, so developers can interact with your APIs without leaving the documentation.

Cloud Endpoints

- Full control over the API gateway
- Uses gRPC
- You must manage the lifecycle
- You pay for Compute Engine VMs
- Service proxy required for services outside Google Cloud



Google Cloud

Let's explore some of the reasons why you might choose one platform over another. First, let's look at some of the reasons why you might choose Cloud Endpoints.

Cloud Endpoints is a Google Cloud API management solution that gives you full control over your API gateway because it is deployed, configured, and managed by you. It supports gRPC, a modern, open-source, high-performance Remote Procedure Call (RPC) framework that can run in any environment.

On the negative side, Cloud Endpoints forces you to manage the entire lifecycle of the NGINX/Envoy-based gateway, including security updates, traffic migration, and managing the compute platform. You also pay for Compute Engine VMs.

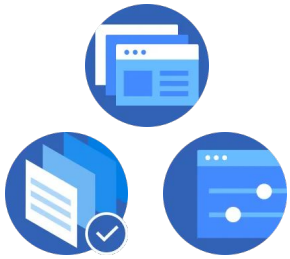
In addition, proxied services outside Google Cloud require a service proxy in Google Cloud.

Now, let’s look at some of the reasons why you might choose API Gateway.

API Gateway

- Fully managed
- Scalable and flexible deployment option
- Relatively inexpensive
- Uses gRPC

- Less control than Cloud Endpoints
- Service proxy required for services outside Google Cloud



Google Cloud

Google Cloud's API Gateway enables you to develop, deploy, secure, and manage your APIs using a fully-managed gateway. You can provide secure access to your backend services through a well-defined REST API that is consistent across all of your services, regardless of service implementation. Why might you choose API Gateway?

API Gateway is fully managed by Google, which allows you to take advantage of all the operational benefits of serverless technology. Google handles management operations like security and configuration updates, while providing you flexible deployment and scalability. API Gateway is often a better solution when using serverless app platforms like Cloud Functions, Cloud Run, and App Engine. It's relatively inexpensive to run as you you don't pay for and manage Compute VMs. API Gateway also supports gRPC.

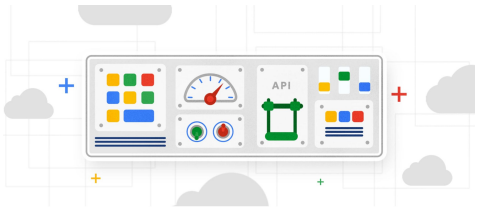
However, since API Gateway is managed by Google, it allows less control than Cloud Endpoints. Also, like Cloud Endpoints, proxied services outside Google Cloud require a service proxy in Google Cloud.

Now let's look at some of the reasons why you might choose the Apigee API Platform.

Apigee API Platform

- Fully featured API management platform
- Proxy APIs anywhere
- Broad feature set

- More expensive
- Not appropriate for simplest API use cases



Google Cloud

Google Cloud's Apigee API Platform is a fully featured platform for developing and managing APIs. By fronting services with a proxy layer, Apigee provides an abstraction or facade for your backend service APIs, allowing you to easily add security, rate limiting, quotas, analytics, and more. Apigee is made to proxy APIs anywhere, not just in Google Cloud.

In addition to the basic API gateway features found in Cloud Endpoints and API Gateway, such as API key and JWT token checks, Apigee provides a developer portal, API products, custom callouts, payload transformation, content-based security, the ability to combine multiple services, and many more features.

Apigee is more expensive to use than Cloud Endpoints or API Gateway. It is typically not appropriate for very simple API use cases.

More resources

API Design Guide

https://cloud.google.com/apis/design/

Authenticating service-to-service calls with Google Cloud Endpoints

https://youtu.be/4PgX3yBJEyw

Google Cloud

Here are some pointers to more information:

The API design guide is a great resource to help you learn how to design APIs in general.

Also, check out this talk from the Google Cloud Next conference.

Deploying an API for the Quiz Application

Duration: 60 minutes

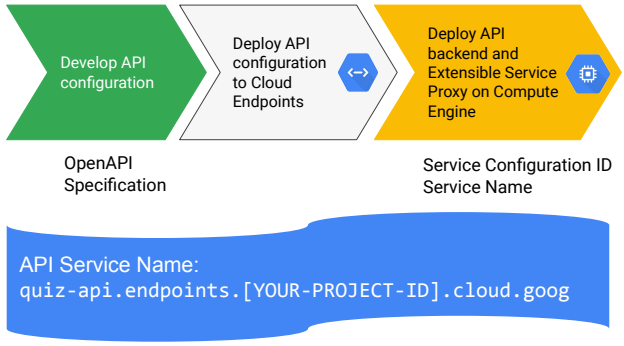
In the lab, Deploying an API for the Quiz Application, you will deploy an API to retrieve quiz questions.

Lab objectives

Create an OpenAPI specification from the existing Quiz application REST API

Deploy the specification as a Cloud Endpoint

Provision a Compute Engine instance with the Extensible Service Proxy to host the Cloud Endpoints API



Google Cloud

You will create an OpenAPI specification from the existing Quiz application REST API. You will then deploy the specification as a Cloud Endpoints API.

Finally, you will provision a Compute Engine instance with the Extensible Service Proxy to host the Cloud Endpoints API.

Quiz

You've deployed an API using Cloud Endpoints. You want to give users the ability to enable your service in their own cloud project and invoke its API. For the majority of use cases, what IAM role should you assign to them?

A. Service Controller

B. Service Consumer

C. Service Editor

D. Service Owner

You've deployed an API using Cloud Endpoints. You want to give users the ability to enable your service in their own cloud project and invoke its API. For the majority of use cases, what IAM role should you assign to them?

a. Service Controller
b. Service Consumer
c. Service Editor
d. Service Owner

Quiz

You've deployed an API using Cloud Endpoints. You want to give users the ability to enable your service in their own Cloud project and invoke its API. For the majority of use cases, what IAM role should you assign to them?

A. Service Controller

B. Service Consumer

C. Service Editor

D. Service Owner

If you've deployed an API using Cloud Endpoints and you want to give users the ability to enable your service in their own Cloud project and invoke its API, for the majority of use cases you should assign them a Service Consumer IAM role. (B)

Quiz

You've updated your API backend. The changes are not backward compatible and will break consumers of your current API. What approach would best serve your customers?

A. Deploy two versions of your Cloud Endpoints API by creating a separate API configuration for each version

B. Disable the original version of your API and deploy an update

C. Notify customers of the loss of backwards compatibility and deploy the updated version of your API

Google Cloud

You've updated your API backend. The changes are not backward compatible and will break consumers of your current API. What approach would best serve your customers?

a. Deploy two versions of your Cloud Endpoints API by creating a separate API configuration for each version.
b. Disable the original version of your API and deploy an update.
c. Notify customers of the loss of backwards compatibility and deploy the updated version of your API.

Quiz

You've updated your API backend. The changes are not backward compatible and will break consumers of your current API. What approach would best serve your customers?

A. Deploy two versions of your Cloud Endpoints API by creating a separate API configuration for each version

B. Disable the original version of your API and deploy an update

C. Notify customers of the loss of backwards compatibility and deploy the updated version of your API

Google Cloud

You've updated your API backend. The changes are not backward compatible and will break consumers of your current API. The approach that would best serve your customers is to deploy two versions of your Cloud Endpoints API by creating a separate API configuration for each version. (A)

Cloud Endpoints enables you to deploy, secure, manage, and monitor your API.

With Cloud endpoints, you can deploy REST or GRPC-based APIs.

You can define the API configuration using industry standard open formats such as OpenAPI for REST APIs and Protocol Buffers for GRPC-based APIs. These APIs can invoke functionality in an API backend that runs on any Google Cloud computer environment, such as Compute Engine, GKE, or App Engine. If you're keen on deploying GRPC APIs in order to take advantage of bi-directional streaming with HTTP to base transport, but you still want to allow clients to make REST API calls, you can specify special mapping roles. The extensible server proxy for GRPC translates RESTful JSON over HTTP into GRPC requests. This approach gives you greater flexibility when integrating with other systems.


Google Cloud