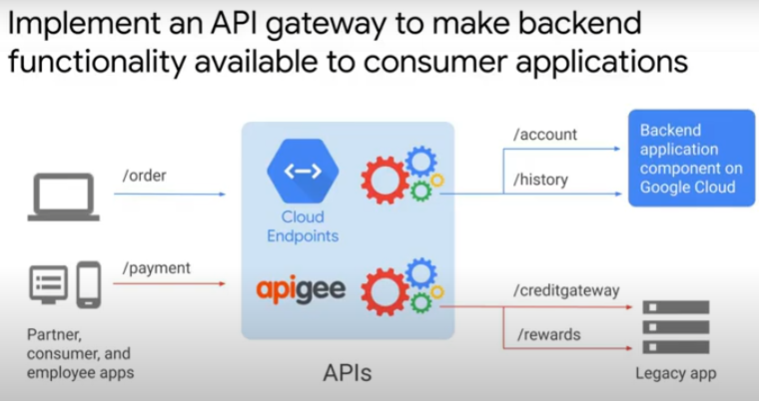
Course: Securing and Integrating Components of your Application

Module 5: Managing APIs with Cloud Endpoints

**Cloud Endpoint Concepts**

To learn:

1. how to develop openAPI configuration for REST API
2. techniques to restrict access and secure API
3. techniques to deploy new versions of API
4. techniques to deploy API to test in production environments
5. how to monitor metrics for API

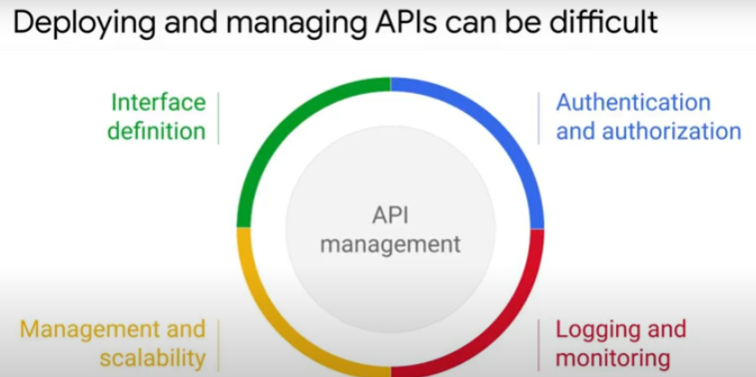


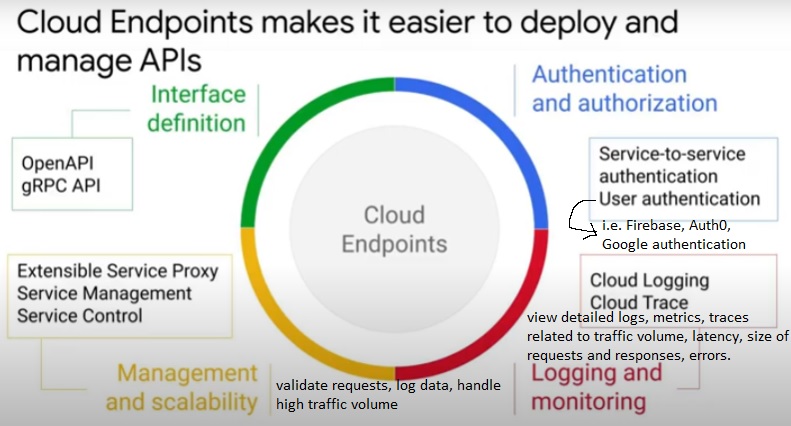
API gateway

* is a consistent interface for clients
* a layer of abstraction
* insulates clients from server-side details and changes to microservices
* can use Cloud Endpoints to implement API Gateways
* the backends can be running on different infrastructures, e.g. App Engine, GKE or Compute Engine
* implement adaptor layer or facade if backend includes legacy application that cannot be refactored and migrated to the Cloud. Clients won’t need to continue using old protocols and deal with disparate interfaces.
* Use Apigee API platform to design, secure, analyze and scale your APIs

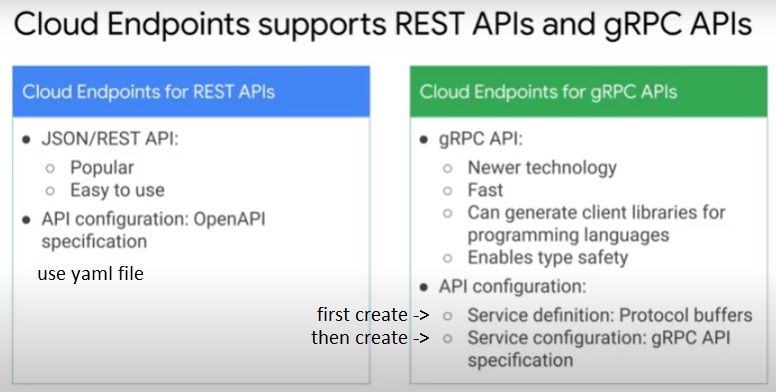
Issues with API deployment and management:

* what language and format to use for interface? OpenAPI?
* How to authenticate services and users?
* Can scale to meet usage demand?
* Log details of invocation and provide monitoring metrics?





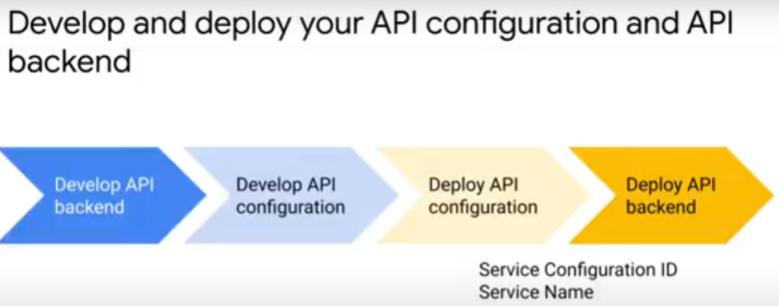
**REST APIs and gRPC APIs**



* Cloud Endpoints support transcoding HTTP JSON calls into gRPC calls
  + clients can use HTTP calls to access gRPC APIs

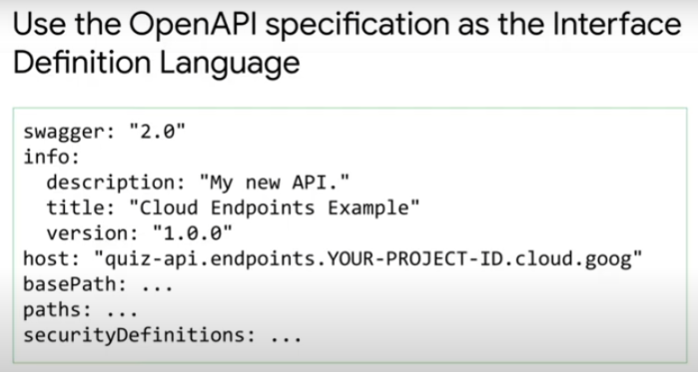
**Cloud Endpoints for REST APIs**

High level overview



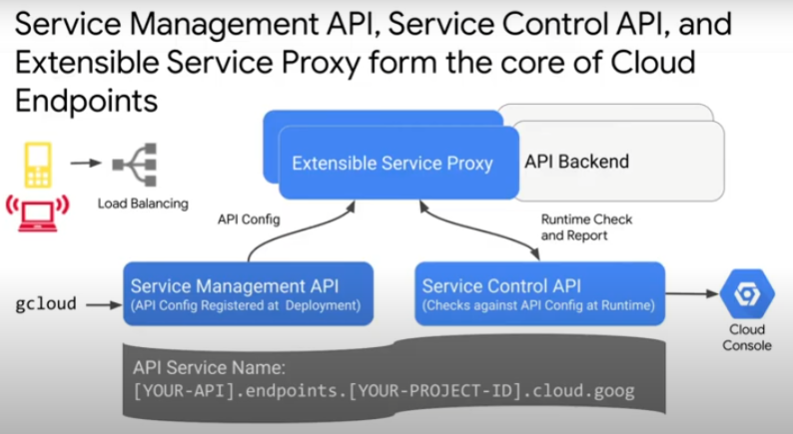
* https://cloud.google.com/endpoints/docs/deploy-api
* to publish REST API to Cloud Endpoints, need an OpenAPI configuration yaml file that describes the API.
* Cloud Endpoints use Service Management - an infrastructure service of Google Cloud to create and manage API and services
* command to deploy OpenAPI configuration yaml file to Service Mangement: *gcloud endpoints services deploy openapi.yaml*
* on successful deployment of OpenAPI configuration, gcloud command returns service configuration ID and service name.
* Specify service configuration ID and service name in API’s backend configuration files, e.g. app.yaml for App Engine flexible environment deployments.
* Deploy API backend.
* Can use scripts

Using OpenAPI specification



* an API configuration describes the API
  + describes surface of API and its security definitions
    - specify user authentication, and service-to-service authentication
* based on OpenAPI specification
* a yaml file
* for resources on creating OpenAPI specification, refer to Swagger editor and resources.

Service Management, Service Control and Extensible Service Proxy



https://cloud.google.com/endpoints/docs/openapi/architecture-overview

Service Management

* configures API management rules
* OpenAPI configuration is registered with Service Management
  + shared with Extensible Service Proxy
* uses the value of the “host” field in the yaml file to create a new Cloud Endpoints service. Format of name is in image. Configuration of service according to yaml file.
* Uses DNS compatible names to uniquely identify services
  + Google project names are globally unique. So can use to create unique service names.
  + Can also map own DNS name to API.

Extensible Service Proxy (ESP)

* injects the functionalities (authentication, monitoring and logging) of Cloud Endpoints
* an NGINX based proxy
* runs in front of API backend
* retrieves API configuration from Service Management. Use it to validate incoming requests.
* Can be deployed as container in front of application or as sidecar – in container beside each instance of application.
* For serverless platforms, i.e. Cloud Run, Cloud Functions and App Engine, ESPv2 is deployed as Cloud Run service as remote proxy.
* can validate JWTs and Google ID tokens
* Employs techniques such as heavy caching and asynchronous calls to remain lightweight and performant.
* ESPv2 is an Envoy-based proxy that works with OpenAPI specification v2.
* https://cloud.google.com/endpoints/docs/choose-endpoints-option

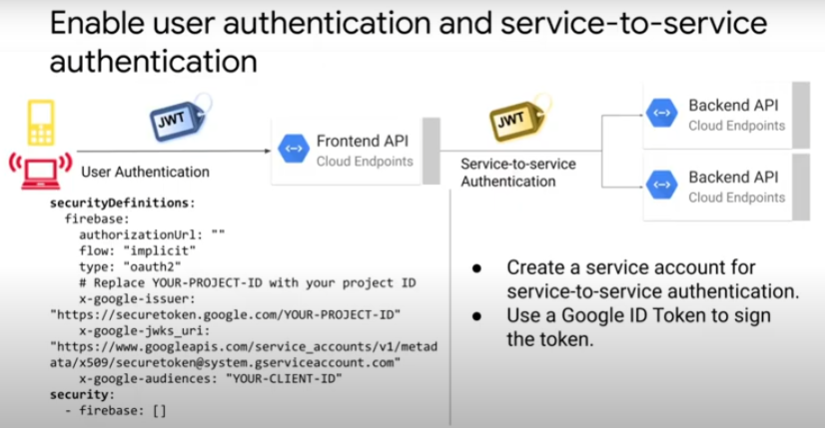
Incoming calls can be from web, mobile applications or other services.

* Load balanced (use Cloud Load Balancing or ingress proxy)
  + then routed to Extensible Service Proxy

Service Control

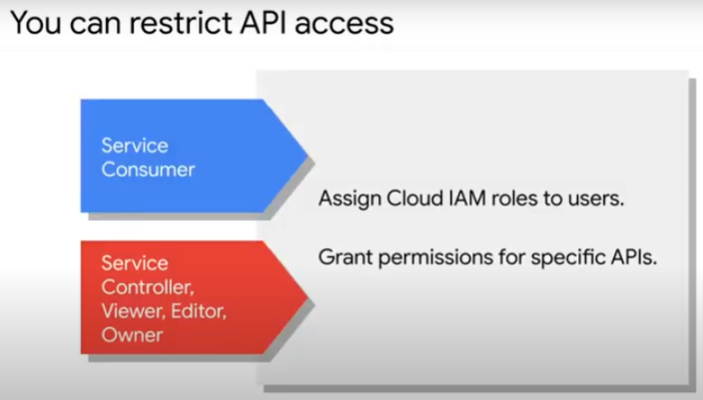
* applies API management rules at runtime
  + authentication – verifies keys and determines if call permitted
  + report - logging and monitoring.
    - View using Cloud Console

Authentication



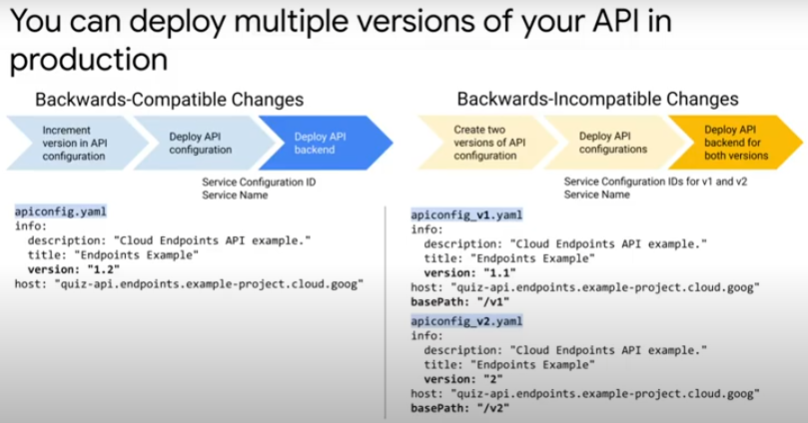
* users invoking frontend API can be authenticated
* security definitions specified in OpenAPI configuration yaml file
* supports Firebase, auth0, Google authentication and other custom authentication methods.
* Flow: user signs in -> authentication provider sends signed JSON web token to Cloud Endpoints. ESP or ESPv2 validates JWT using public key of signer specified in OpenAPI configuration, and validates audience field of JWT.
* If API key needed, ESP or ESPv2 calls Service Control API to validate key. Service Control API checks its policy, and also ensures project associated with key has enabled the API.
  + If valid, orginal request, together with a JWT validation header, forwarded to backend.

Restricting access to API



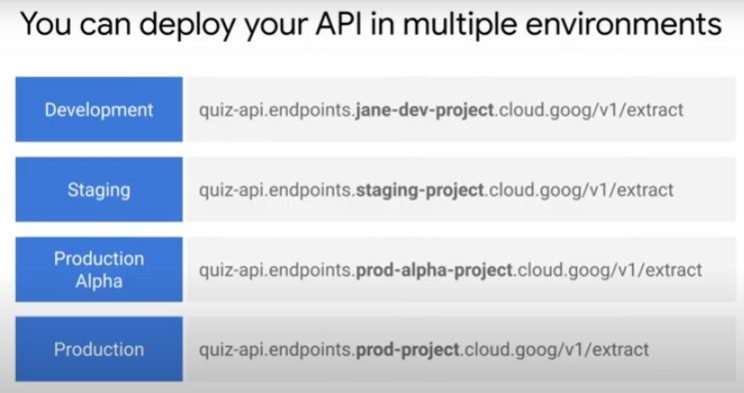
* To restrict access to APIs, assign IAM roles to specific users
* can cover specific APIs or entire Cloud project
* to enable clients to use your service and invoke your APIs in their own application, grant “Service Consumer” role.
* Service Controller, Viewer, Editor and Owner roles give greater permission to manage configuration and project

API versioning and backwards compatibility



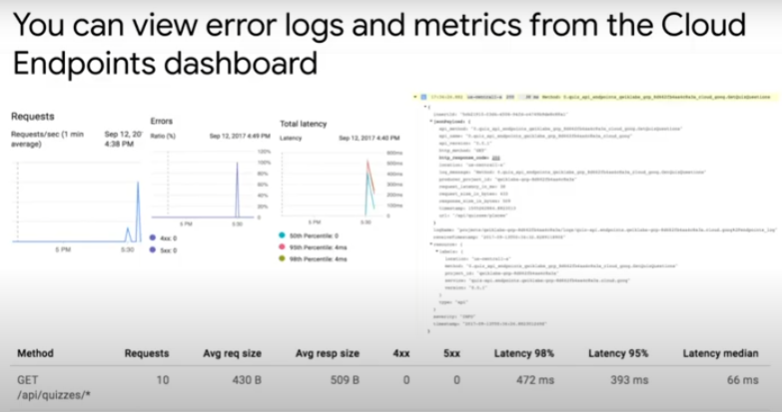
* There may be changes, bug fixes to API backend
* If changes are backwards compatible, it is good practice to increment version attribute in OpenAPI configuration yaml file, and re-deploy Endpoints service (OpenAPI config and backend).
* If changes not backward compatible, deploy 2 versions of API.
  + save separate OpenAPI configuration file for each version.
  + *gcloud endpoints services deploy* command will return different *Service Configuration IDs* for each version.
  + Update 2 separate backend API configuration files with each Service Configuration ID.
  + If not backwards compatible, will break functionality for API consumers
* If API version is to be deprecated and phased out, just delete.
  + Inform consumers well in advance for them to evaluate and migrate to newer versions

Deploy APIs in different environments for different stages of life-cycle



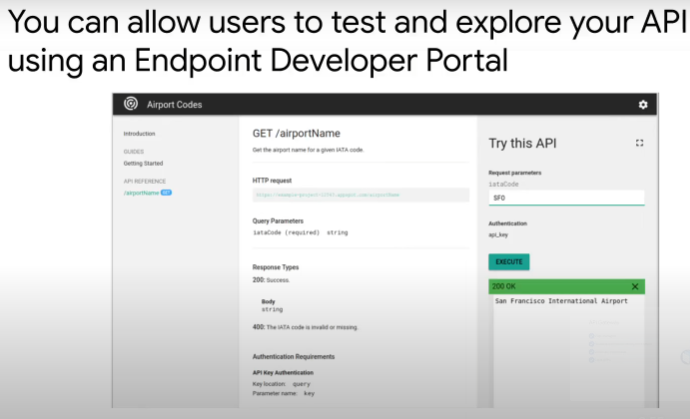
* different stages when developing and deploying APIs
* create seperate Google project for each environment
  + an API with different Cloud Endpoints for each
  + for different use cases, e.g.
    - run unit tests in development environment,
    - integration tests during staging

Logs, metrics and traces



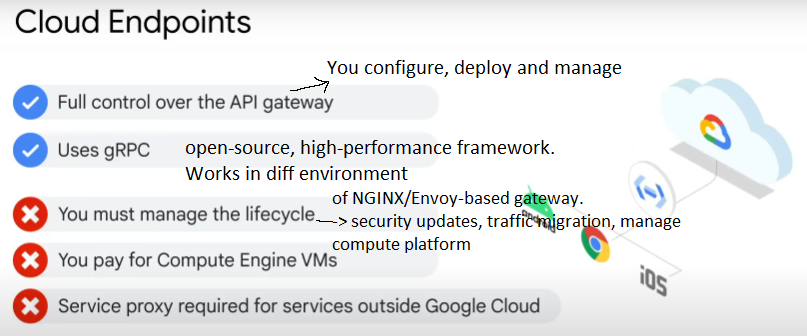
* In Cloud Endpoints dashboard, can see metrics on requests, errors (4XX, 5XX) and latency.
* In Cloud Logging, can see detailed info on requests to each API

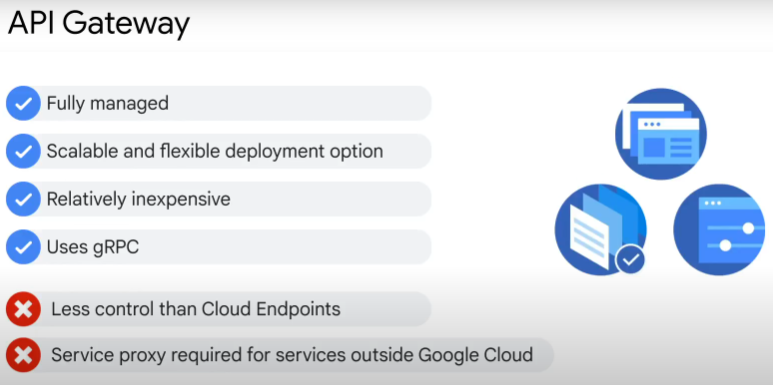
Developers portal from Cloud Endpoints page



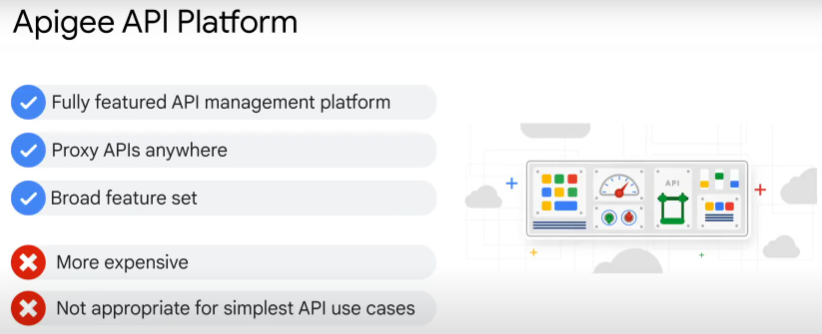
* Can create developers portal from Cloud Endpoints page.
  + Allow users to explore and test your APIs.
  + Has SmartDocs interactive documentation. Includes “Try this API” panel.

Pros and Cons of Cloud Endpoints vs API Gateway vs Apigee API Platform





* better solution for serverless app platforms, e.g. Cloud Function, Cloud Run and App Engine.
* develop, deploy, secure and manage APIs
* provide secure access to backend services. Consistent client interface regardless of server implementation.
* Fully managed by Google
  + Google takes care of security and configuration updates
  + less manual control
* cheaper than Cloud Endpoints because no need to pay for compute VMs



* proxy layer is abstraction / facade for backend service APIs
* works with other Clouds and on-prem too
* features added: security, rate-limiting, quota, analytics etc
* additional features over Cloud Endpoints and API Gateway: developers portal, API products bundling, custom call outs, payload transformation, client libraries, content-based security etc

References:

API Design Guide https://cloud.google.com/apis/design

Authenticating service-to-service calls with Cloud Endpoints

https://youtube.com/watch?v=4PgX3yBJEyw&si=EnSIkaIECMiOmarE