Architecture

Hansel Miranda
Technical Curriculum Developer, Google

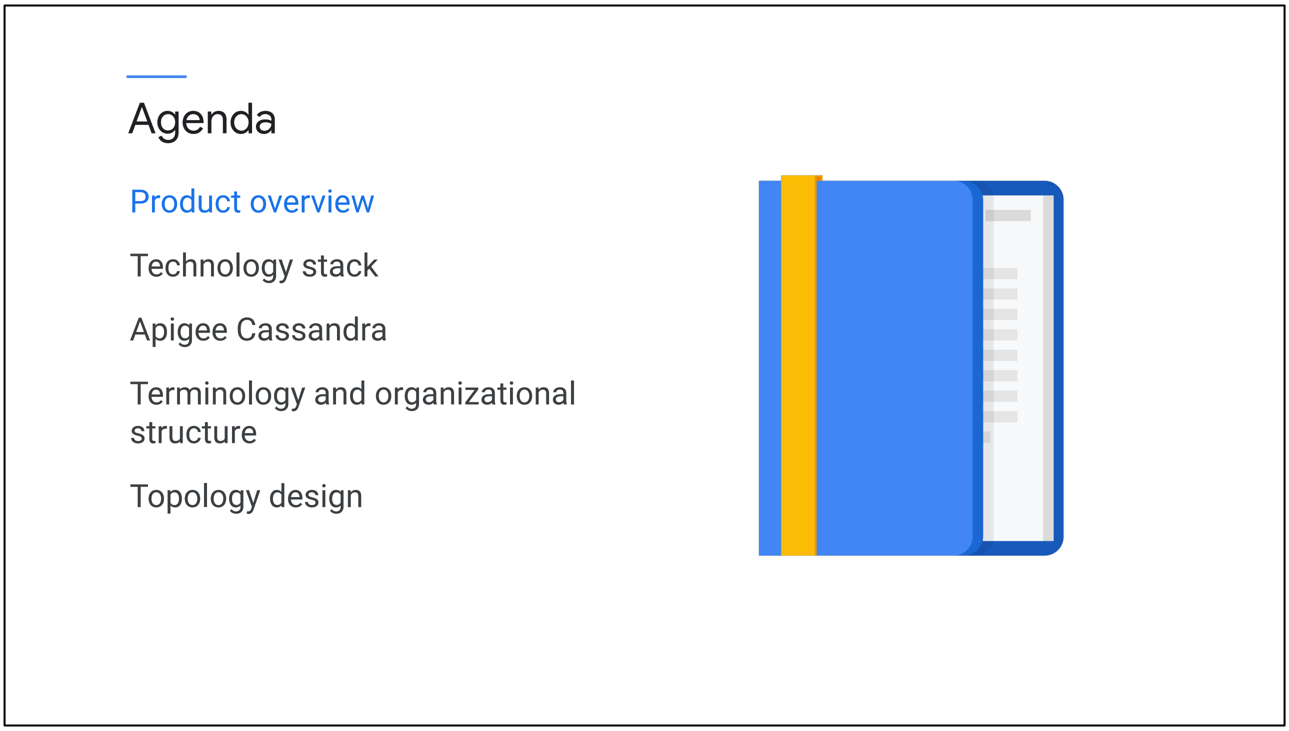
Welcome to the Architecture module of the On-premises Installation and Fundamentals course of Google Cloud's Apigee API Platform.

This module represents the starting point on your journey using Apigee.

The goal is to help you build a good understanding of Apigee's product capabilities and architecture characteristics.

This knowledge represents the foundation you will need to master the Apigee API platform for private cloud.

Let's begin.



Agenda

Product overview

Technology stack

Apigee Cassandra

Terminology and organizational structure

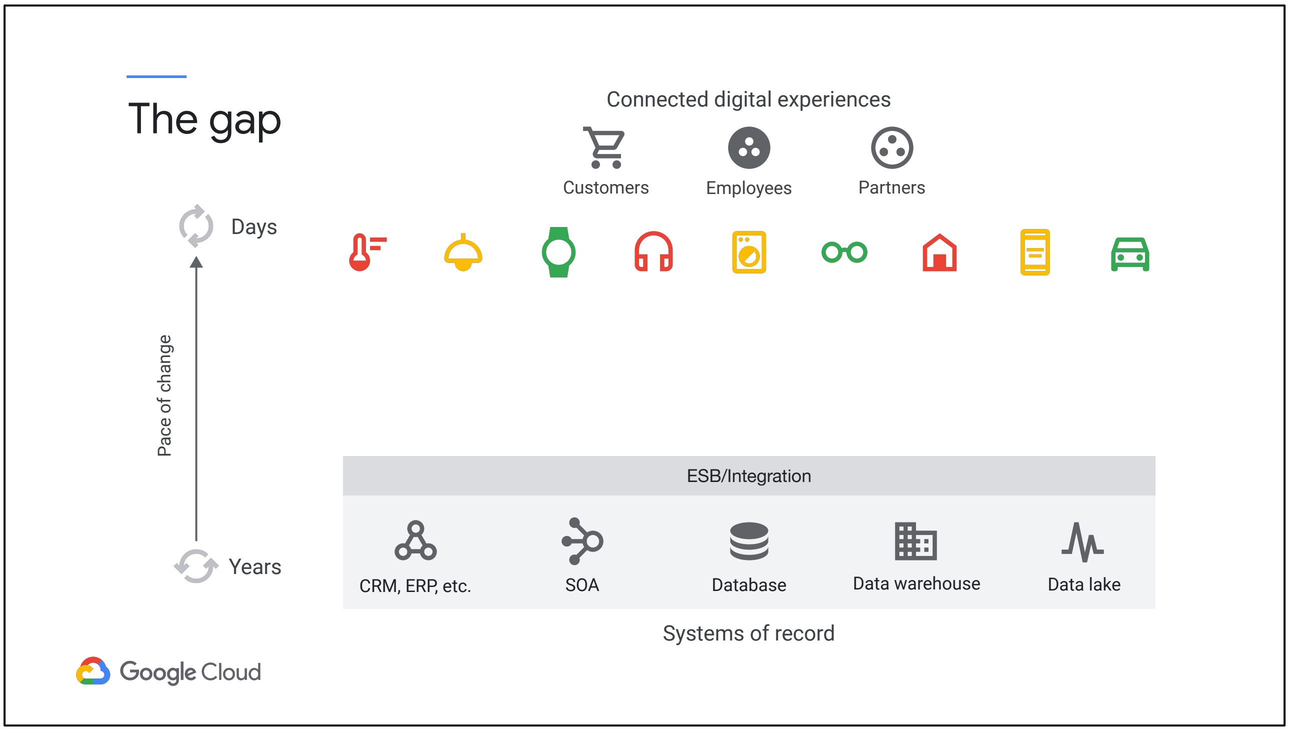
Topology design

In this module, we start with an overview of Apigee, Google Cloud's API Management Platform.

We will discuss the business problems that can be solved using Apigee, review the features that Apigee provides, and introduce the components of Apigee and deployment options for the Apigee platform.

You will learn the architecture and technology stack used in Apigee Edge for private cloud.

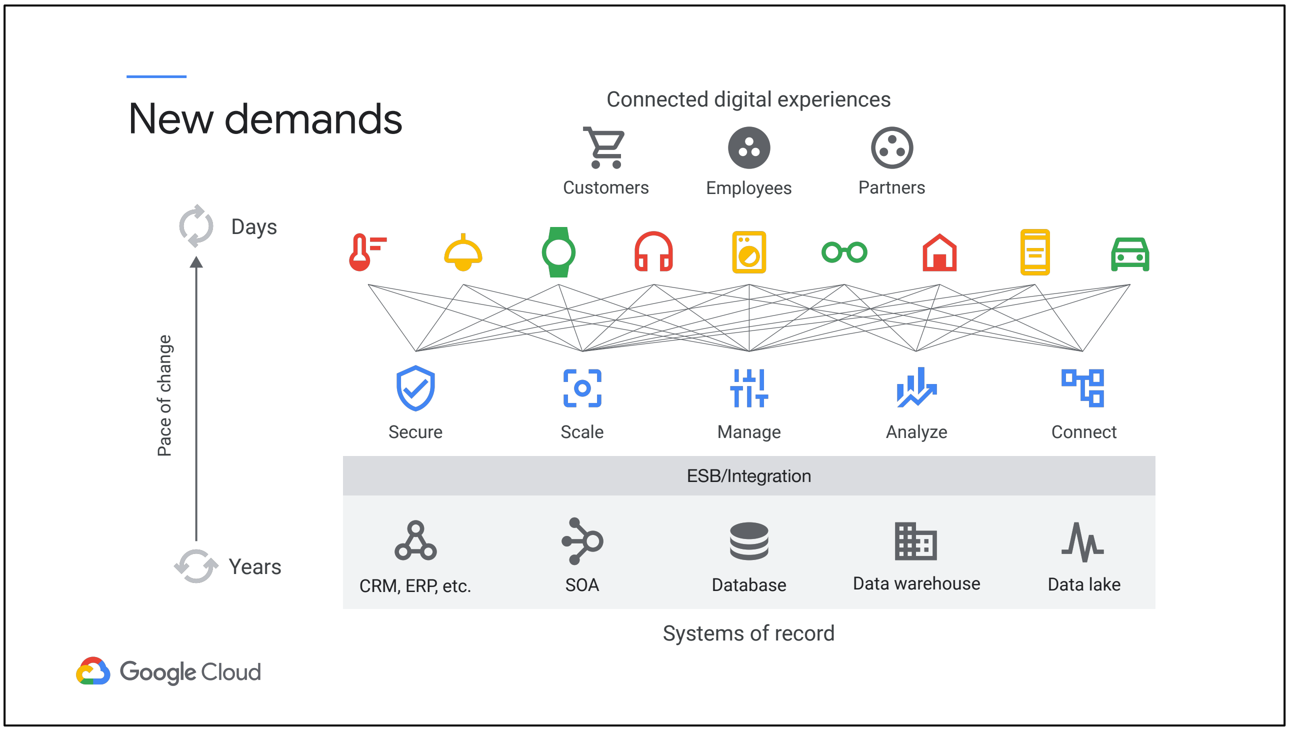
We will discuss the terminology and organizational structure, and you will learn about topology design and its use in planning your Apigee private cloud infrastructure.



The gap

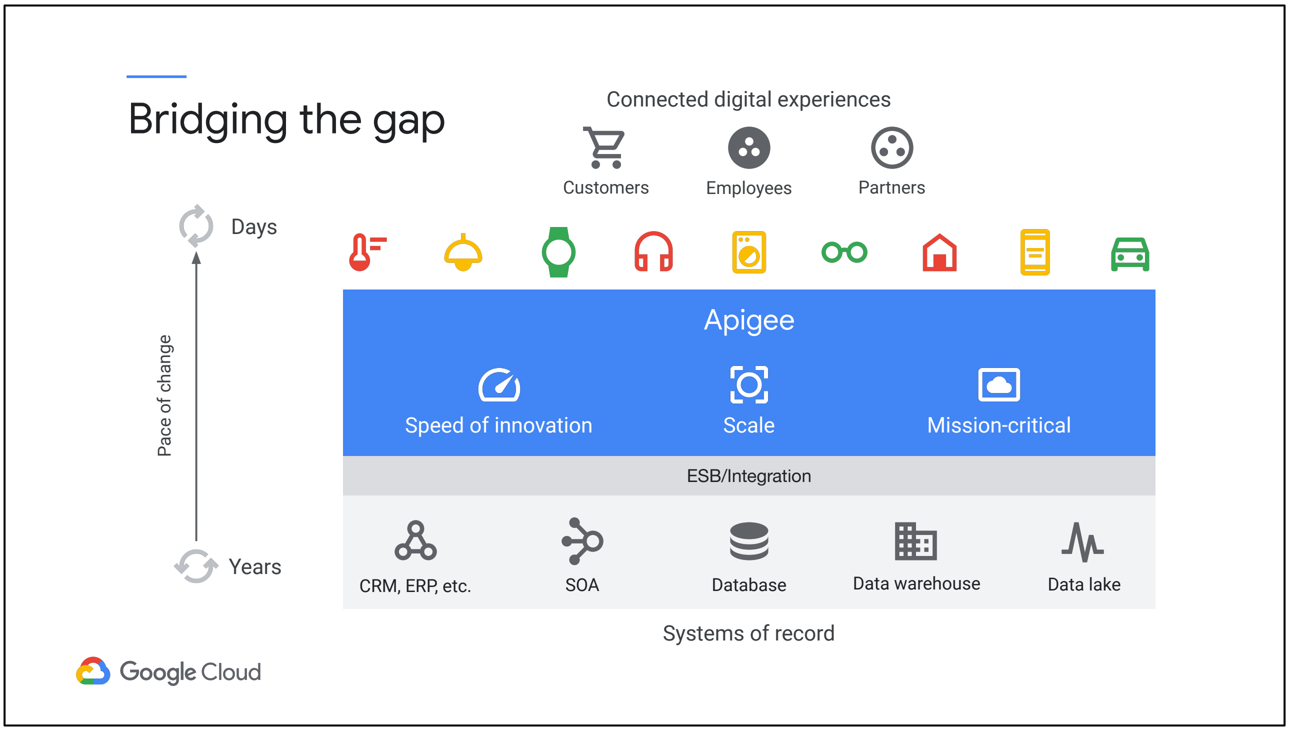
Modern applications provide features and capabilities that legacy systems cannot directly provide.

The pace of change in these digital experiences is much faster than that in traditional systems, and so a gap exists between applications that provide connected experiences and those that provide the business and data systems of record.



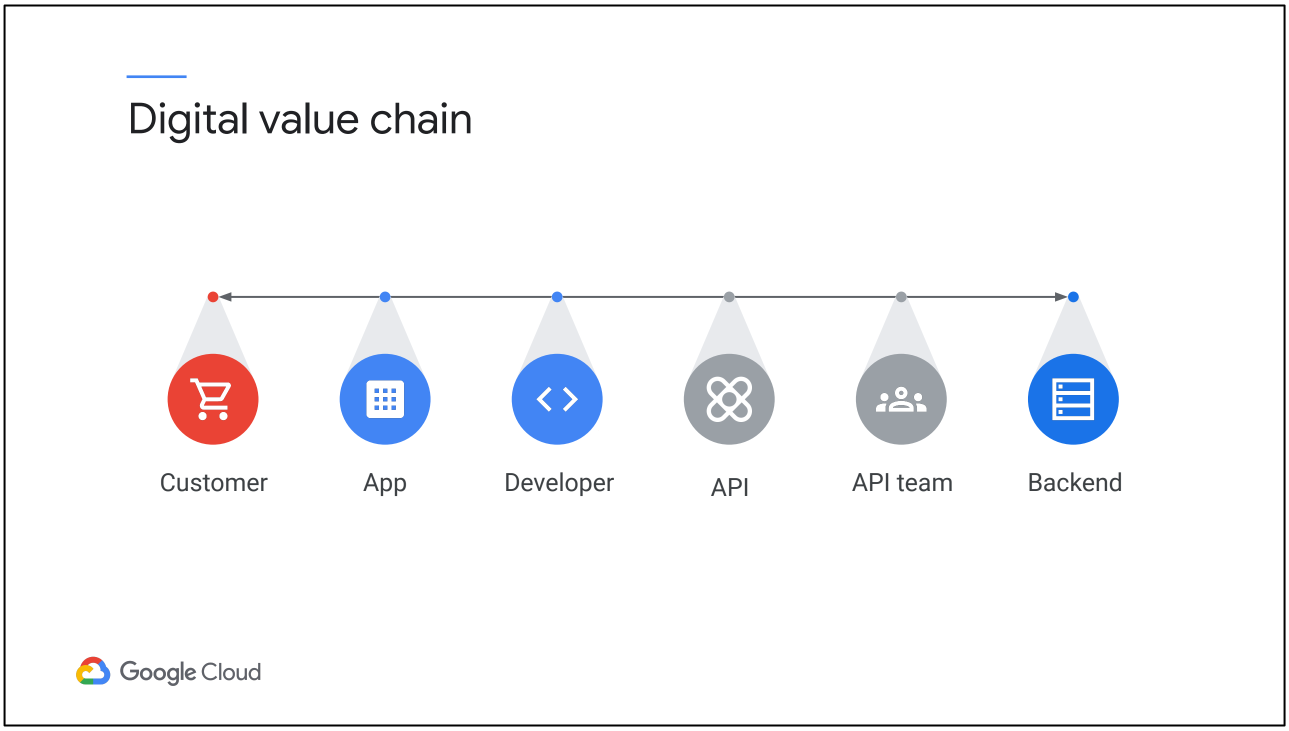
Applications that provide these connected experiences to end users must be able to do so securely and at scale.

They have to manage the entire application lifecycle, connect to different backend systems, and be able to track and analyze the interactions between consumers and producers of data and services.

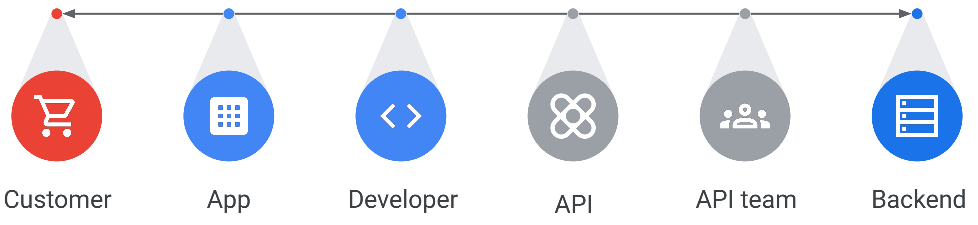


Bridging the gap

Apigee is a fully featured API management platform that bridges the gap and enables application developers and API providers to create connected digital experiences for end users.



Digital value chain



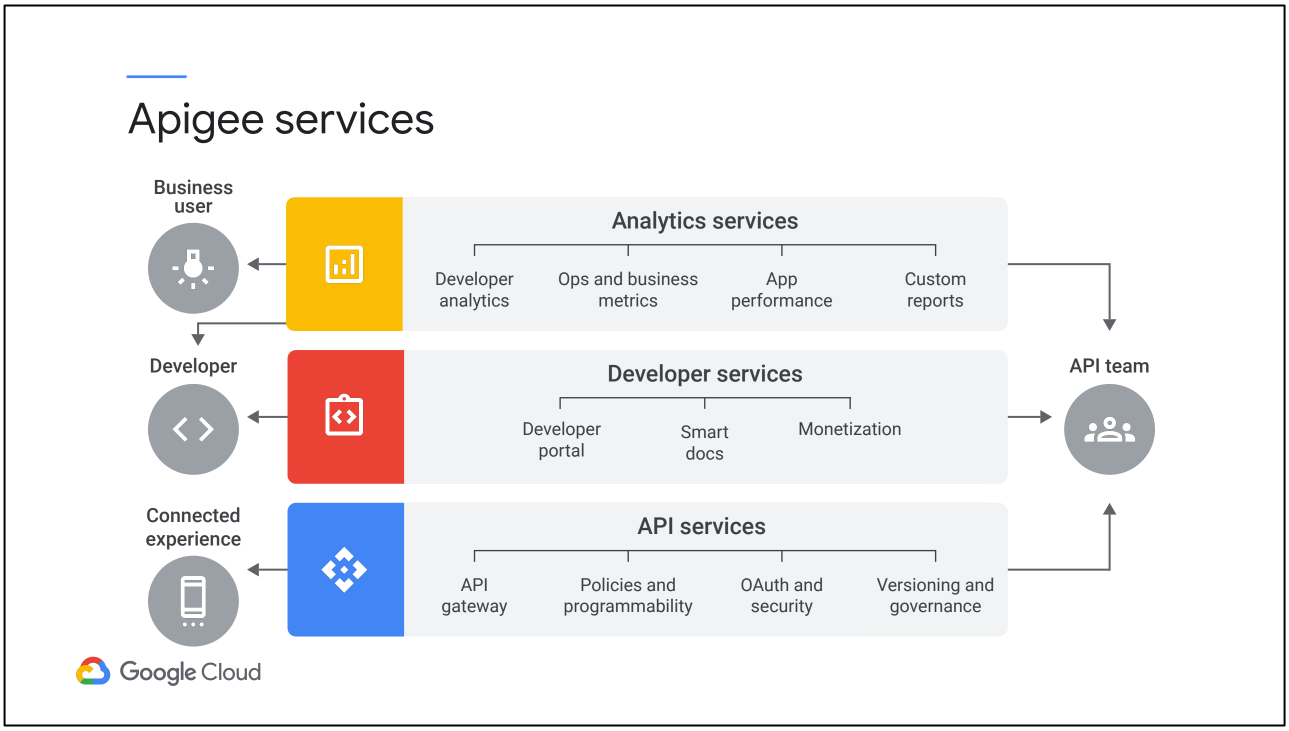
Google Cloud

The Digital Value Chain helps enterprises provide customers or end users with connected digital experiences using their backend data and services.

Web or mobile apps are built by internal enterprise developers or by external third-party companies.
App developers leverage the APIs offered by your company.

These **APIs** are built and managed by the **API Team** within the enterprise. APIs integrate with **backend** services and other service endpoints, thus providing a clean and seamless interface to applications.

By focusing on the consumption side of the digital value chain, Apigee improves the app developer experience and makes it easier for your APIs to be successfully consumed.



The Apigee platform includes an API services layer that provides the runtime API gateway functionality.

This layer includes the API processing engine, which enables you to add policies and programming to your API proxy.

During proxy execution, these policies are applied to requests from applications and responses from backend systems.

API Services also provide security functionality in the form of OAuth and other security policies for identity verification, authentication, and access control.

It supports features for request and response mediation, revision control and versioning, orchestration, and much more.

To enable the digital value chain, the Apigee platform includes Developer Services. This includes a developer portal that enables app developers to consume your APIs.

App developers can register their applications and view the API catalog. Developers can browse your API documentation, and even try out your API using Apigee's SmartDocs feature.

Any API program must be able to measure and track its performance. The Apigee platform includes Analytics services, which enable enterprises to report on various aspects of the API program.

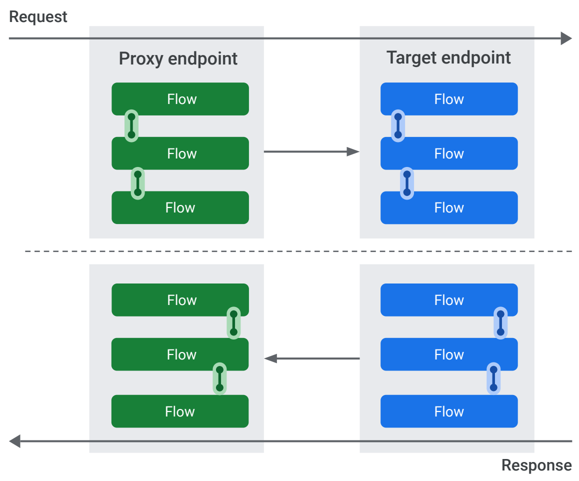
It has many dashboards that chart various dimensions and metrics, including API proxy traffic, response times, target response times, cache performance, developer engagement, and traffic composition.

You can view these dashboards in the Apigee UI or retrieve analytics data using the management API.

You can also create your own custom reports to chart dimensions that are not in the default dashboard reports, or use custom dimensions.

Analytics services is part of the core platform and is a critical component used in API management.
Using this service enables you to gauge the operational health of the platform and determine the overall business value and success of the API program.

API services



Traffic management
Quota
Spike Arrest
Response Cache
Lookup Cache
Populate Cache
Invalidate Cache
Reset Quota

Mediation
JSON to XML
XML to JSON
Raise Fault
XSL Transform
SOAP Message Validation
Assign Message
Extract Variables
Access Entity
Key Value Map Operations

Extension
Java Callout
Python
JavaScript
Service Callout
Flow Callout
Statistics Collector
Message Logging

Security
Basic Authentication
XML Threat Protection
JSON Threat Protection
Regular Expression Protection
OAuth v2.0
Get OAuth v2.0 Info
Set OAuth v2.0 Info
Delete OAuth v2.0 Info
Verify API Key
Access Control
LDAP
Generate SAML Assertion
Validate SAML Assertion
Generate JWT
Verify JWT
Decode JWT
Generate JWS
Verify JWS
Decode JWS

Policies

Google Cloud

API Services provide the runtime execution context for your API proxy.

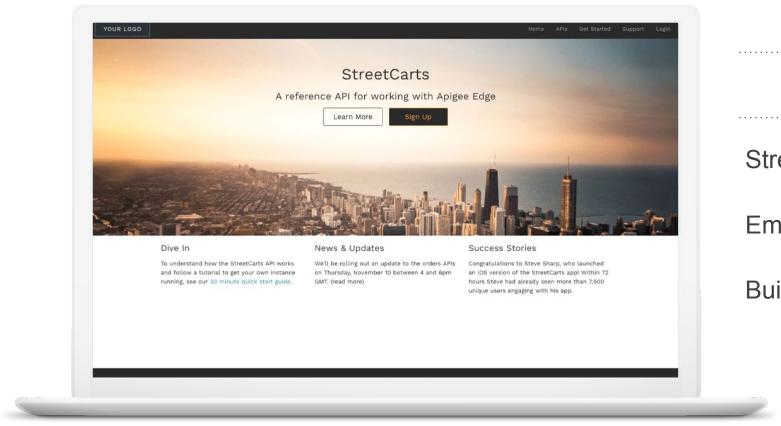
To develop the API proxy logic, you include policies in the proxy or target endpoints of the API proxy within the request or response flows.

A policy is a configuration file written in XML that provides a specific functionality.

Apigee has various pre-built policies that implement traffic management, request and response mediation, and security.

You can also write custom code that can be executed by extension policies.

Developer services: Developer portal



StreetCarts

A reference API for working with Apigee Edge

Dive In

To understand how the StreetCarts API works and follow a tutorial to get your own instance running, see our 30 minute quick start guide.

News & Updates

We'll be rolling out an update to the orders APIs on Thursday, November 10 between 4 and 6pm GMT. (read more)

Success Stories

Congratulations to Steve Sharp, who launched an iOS version of the StreetCarts app! Within 72 hours Steve had already seen more than 7,500 unique users engaging with his app.

Streamline API adoption

Empower developers

Build an interactive community

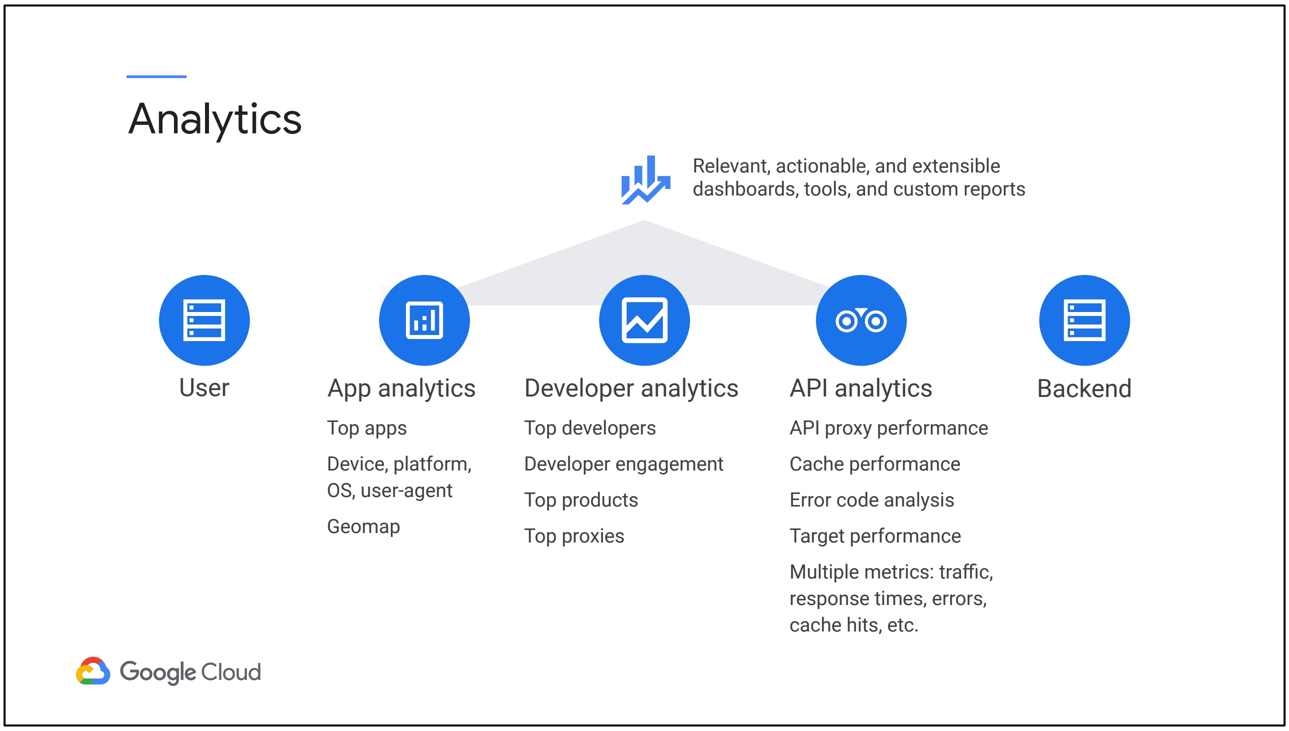
Google Cloud

Apigee supports three developer portal solutions: an integrated portal, a Drupal-based portal, and a do-it-yourself, or DIY, portal.

The integrated portal is a lightweight portal available in the Apigee hybrid UI. It is the fastest and easiest way to onboard app developers and enable them to register their apps, browse API documentation, and start using your APIs. You can customize the look and feel of the integrated portal using custom themes, menus, and javascript code.

The Drupal portal is based on Drupal 8/9. It is a fully customizable portal that you host and manage. In addition to the customization options available with the integrated portal, you can implement Apigee monetization, include blogs and forums, and have custom developer and app registration flows.

The DIY portal is a fully customizable portal that you develop and host using the Apigee management APIs. It involves the most effort and highest risk and takes the longest to implement.



Analytics

Relevant, actionable, and extensible dashboards, tools, and custom reports

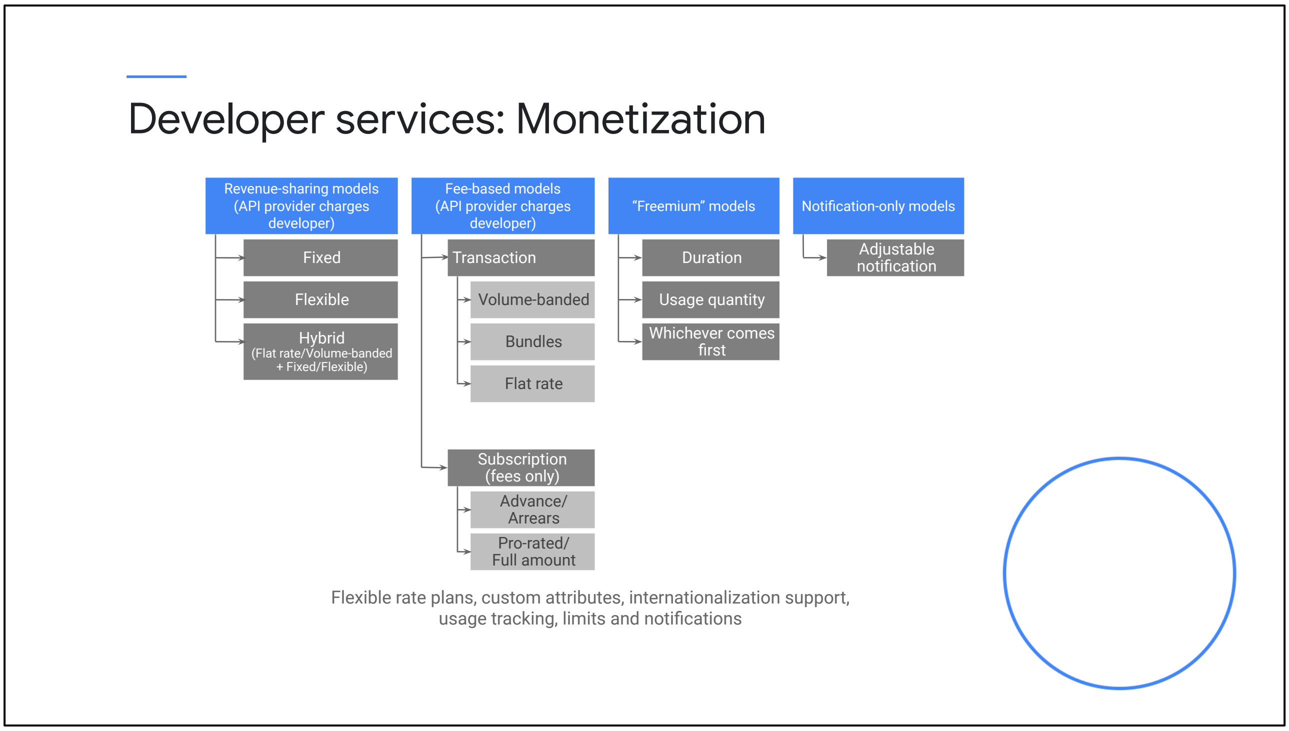
Apigee analytics provides pre-built dashboards for the entire API digital value chain.

These dashboards include charts that report on app analytic dimensions, such as the top apps by API consumption, device type, platform, user-agent, and geographic location.

Developer analytic dashboards include charts on the top developers, API products, and API proxies consumed.

API proxy analytics dashboards have charts on proxy performance, cache performance, error codes, and target performance.

These charts report on different metrics such as total traffic, response times, and cache hits.



Developer services: Monetization

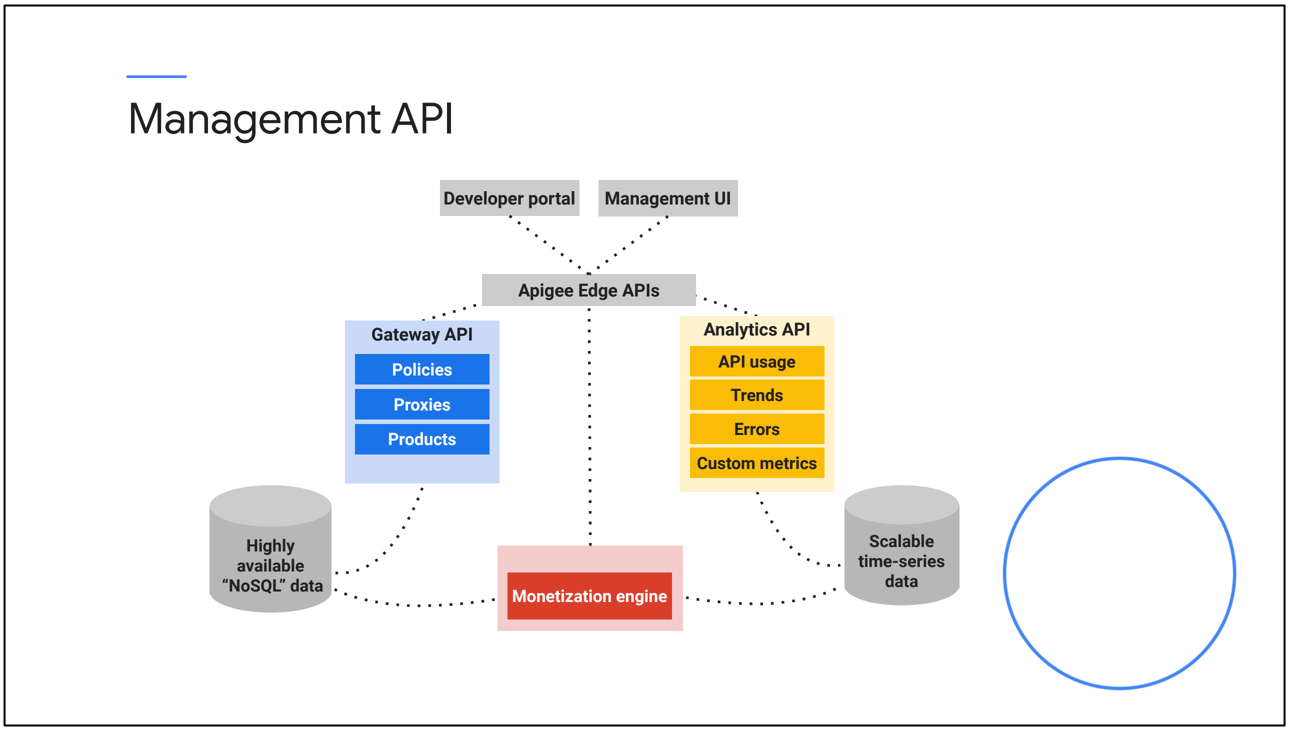
Flexible rate plans, custom attributes, internationalization support, usage tracking, limits and notifications

Monetization is a flexible, easy to use solution to monetize and realize the value of your APIs.

It enables you to charge your API developers or share revenue with them for the use of your APIs.

It allows you to create different revenue sharing and fee-based charging models, manage billing, and process payments.

Monetization integrates with your developer portal allowing you to create and manage subscription plans for your API products.



Management API

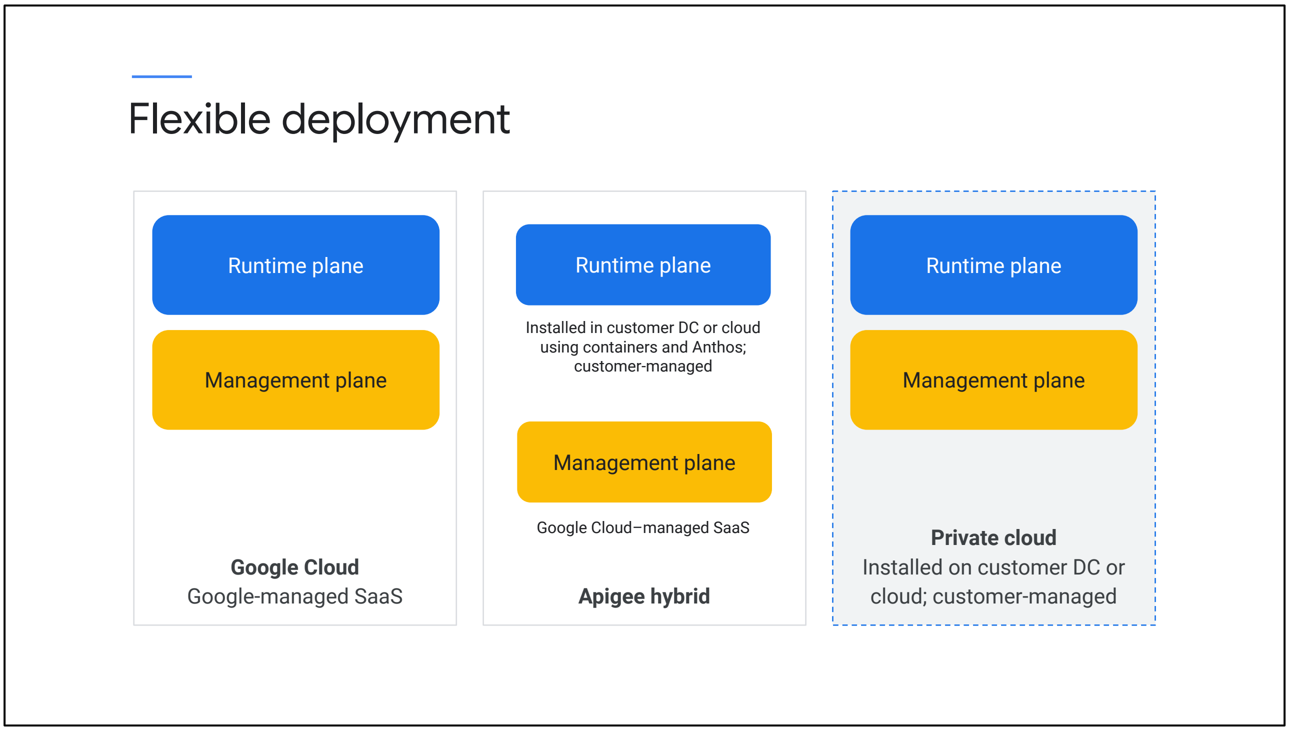
Apigee Edge exposes all of its functionality through APIs.

Those APIs can be used for development as well as operational tasks.

For example, you can deploy APIs, create users, and run analytics reports using APIs.

The management API is consumed by the Apigee Edge enterprise UI as well as the installation and upgrade processes.

You can use the management API for offline development, CICD automation, and other operational activities.



Flexible deployment

Runtime plane
Management plane
Google Cloud
Google-managed SaaS
Runtime plane
Installed in customer DC or cloud using containers and Anthos; customer-managed
Management plane
Google Cloud–managed SaaS
Apigee hybrid
Runtime plane
Management plane
Private cloud
Installed on customer DC or cloud; customer-managed

Apigee offers flexible platform deployment options:

The first option is: *Apigee SaaS (Public cloud, Apigee X)*
Apigee's Google-managed software-as-a-service deployment model simplifies customer adoption and dramatically accelerates time to market for new APIs.

Developers can immediately start building and running APIs at scale. This is the most popular deployment option that enables customers to focus on addressing business needs, while letting Google manage the operational overhead of running the software at scale in a secure and reliable way.

Apigee X is the latest implementation of Apigee Edge for public cloud that is hosted on Google Cloud and managed by Apigee.

Another deployment option is: *Apigee hybrid*
Customers who want to reduce infrastructure management costs but retain the flexibility of running APIs in multiple clouds or on-premises can choose the hybrid deployment model.

This model allows the customer to manage and deploy containerized versions of the API runtime wherever necessary, while delegating the management plane operations to Google.

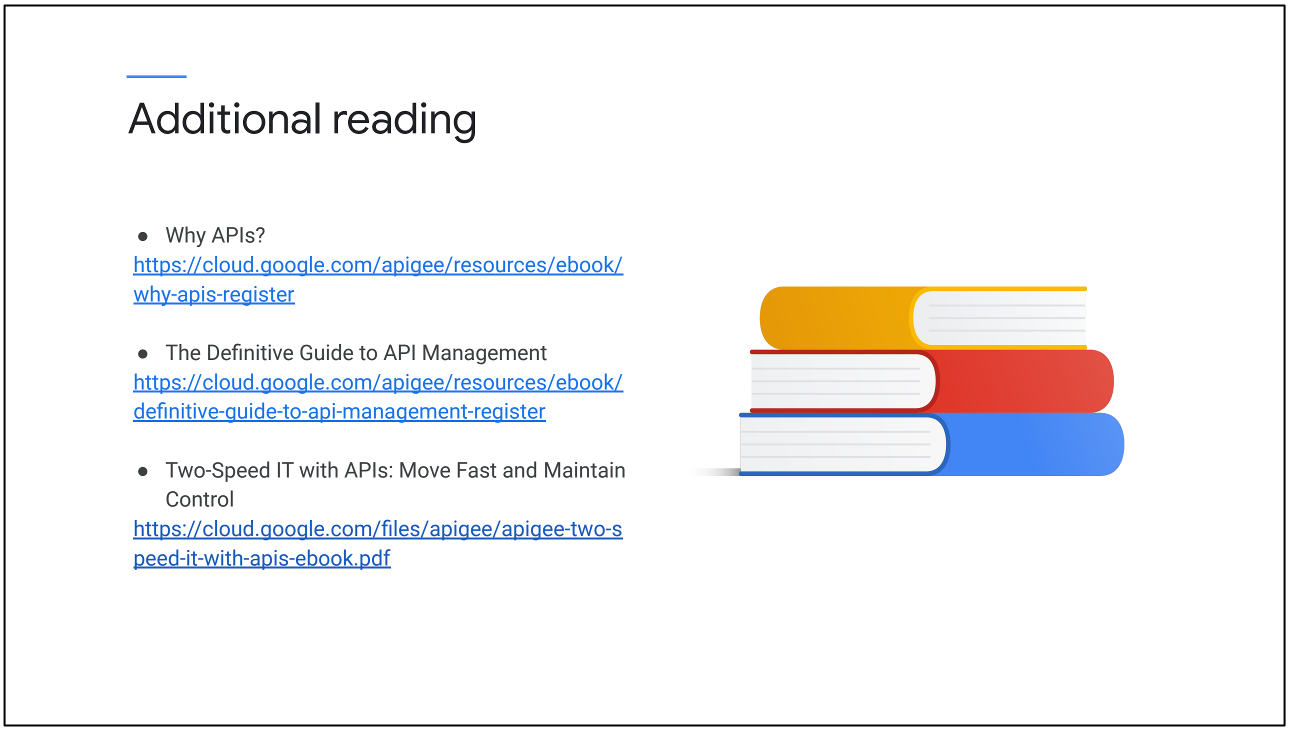
Apigee's hybrid deployment model takes advantage of Anthos, Google Cloud's hybrid and multi-cloud application platform, which simplifies the management of runtime components across multiple clouds and data centers.

The third deployment option is: *Apigee for Private Cloud*
For customers who are willing to manage the Apigee infrastructure, Apigee can be deployed and managed in a private cloud.

Apigee can be installed in an on-premises data center or in an infrastructure-as-a-service cloud.

This course teaches you how to install and manage Apigee for private cloud.

Compare Apigee deployment models



Additional reading

- Why APIs?
  https://cloud.google.com/apigee/resources/ebook/why-apis-register

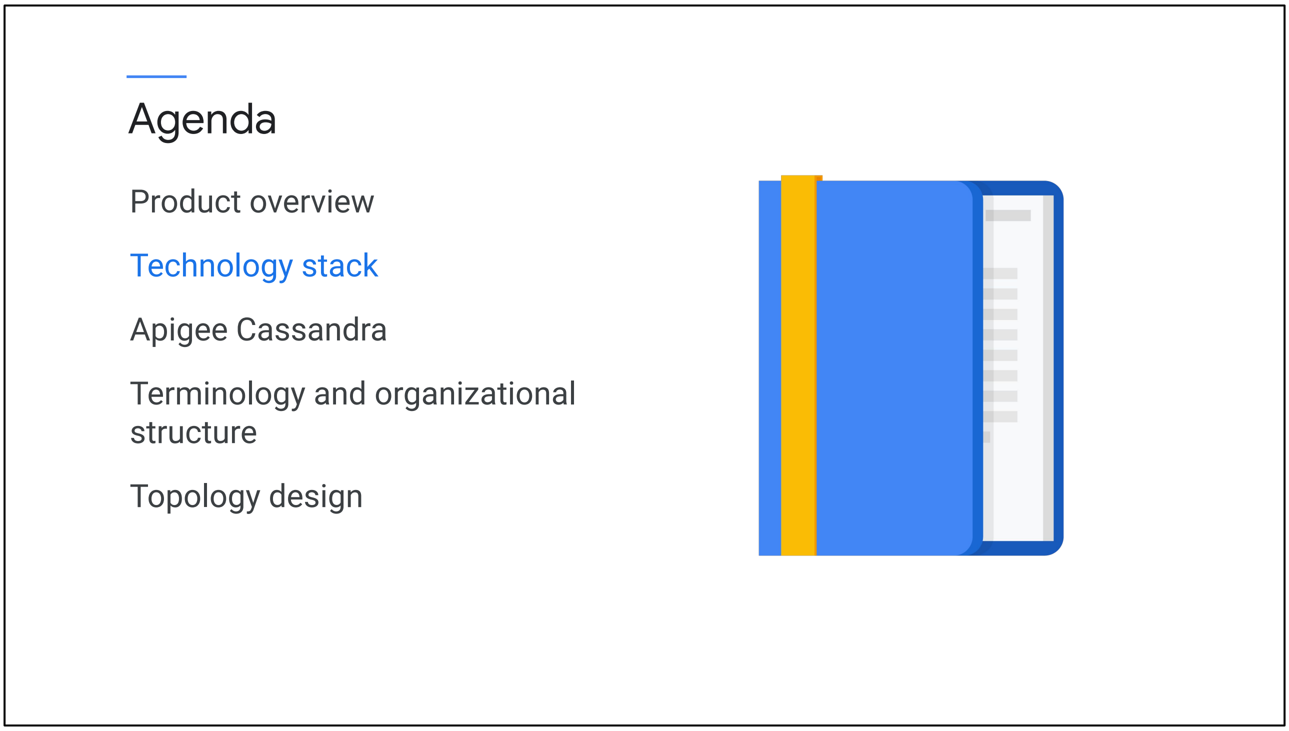
- The Definitive Guide to API Management
  https://cloud.google.com/apigee/resources/ebook/definitive-guide-to-api-management-register

- Two-Speed IT with APIs: Move Fast and Maintain Control
  https://cloud.google.com/files/apigee/apigee-two-speed-it-with-apis-ebook.pdf

There is a variety of reading material available to you to learn about APIs and API management.

Here are a few links to get you started.

The Apigee website at docs.apigee.com has much more information including product specific documentation.



Agenda

Product overview

Technology stack

Apigee Cassandra

Terminology and organizational structure

Topology design

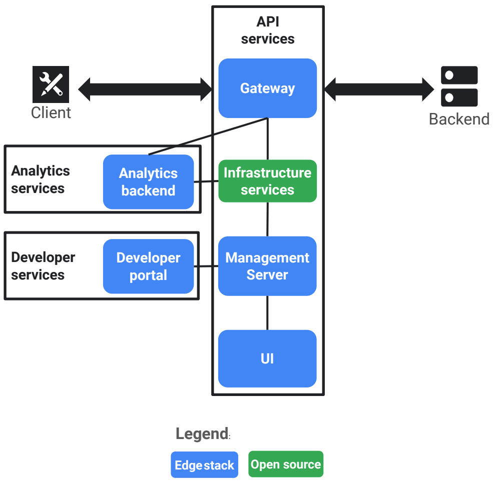
Understanding each individual component is critical to your understanding of the Apigee platform.

In this lesson, we focus on the capabilities of Apigee Edge for private cloud, how each component fits into the platform, and details about the underlying technology stack.

Capabilities

Edge is composed of several stateless components that use infrastructure services to persist data:

- **Gateway:** Routing and API call processing
- **UIs:** Enterprise UI, Developer Portal
- **Infrastructure services:** Persistence of runtime, analytics, and management
- **Management Server:** Provides REST API for all configuration and management tasks

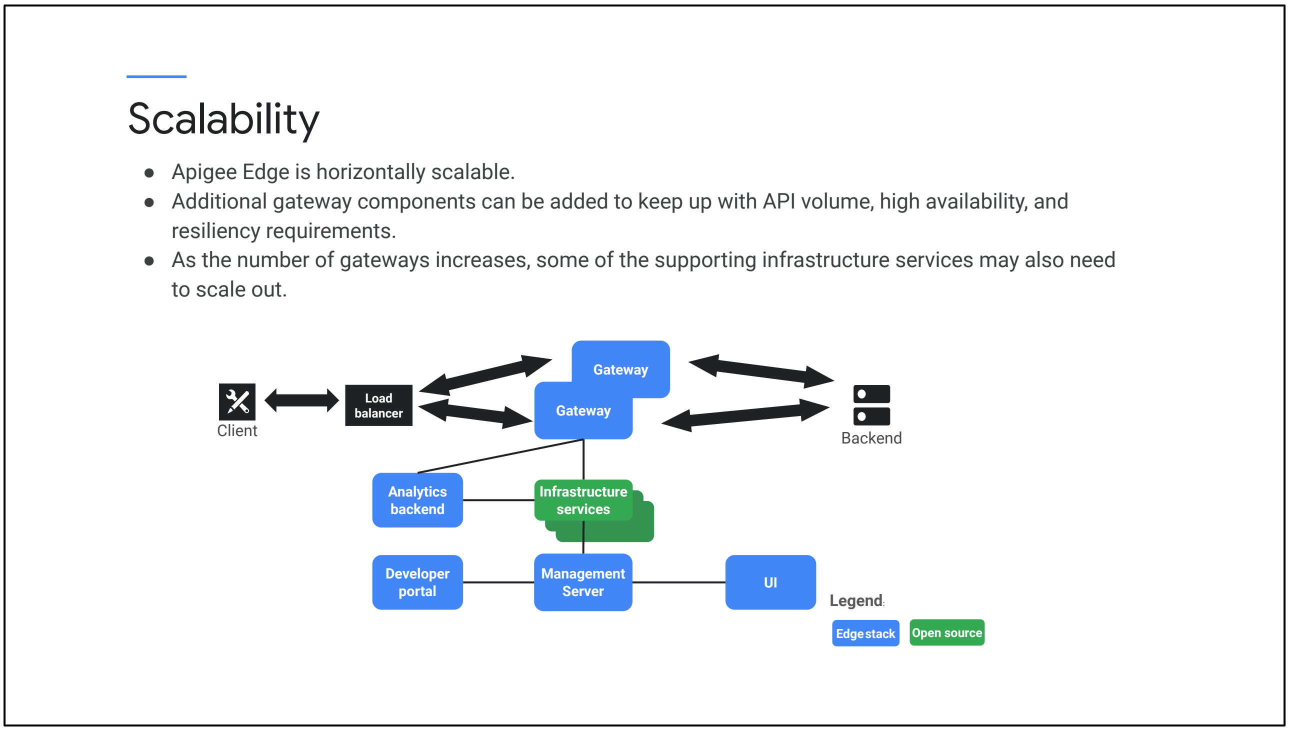


Apigee Edge has three core functional areas: API services, developer services, and analytic services.

On the diagram, Apigee capabilities are shown in blue, open source capabilities are green, items in grey represent customer-specific infrastructure and API clients.

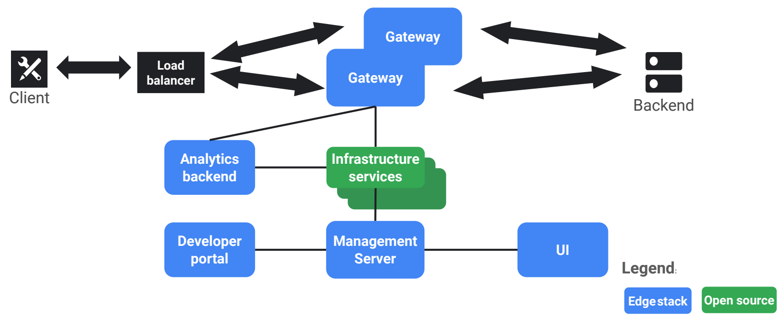
Apigee capabilities such as gateway and analytics, rely on underlying open source infrastructure to support the storage and replication of data.

Components that make up the gateway represent the runtime for APIs that are deployed on the platform.



Scalability

- Apigee Edge is horizontally scalable.
- Additional gateway components can be added to keep up with API volume, high availability, and resiliency requirements.
- As the number of gateways increases, some of the supporting infrastructure services may also need to scale out.

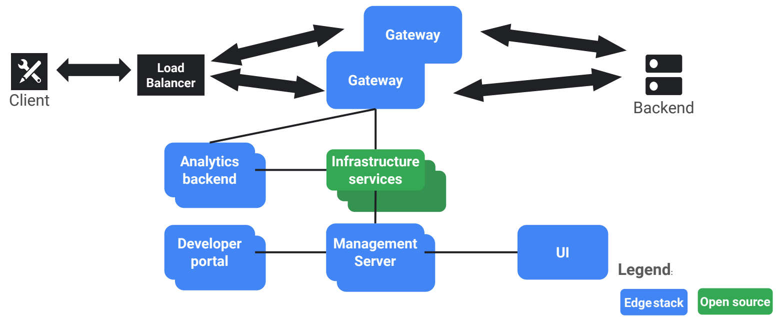


Apigee Edge is horizontally scalable.

In most scenarios, adding capacity to handle an increase in API requests is as simple as adding additional gateway components.

Scalability, cont.

- Management Server, analytics backend and developer portal can also be set up for high availability.
- Multiples instances of these services within a single zone or across zones are possible.
- Gateway, infrastructure services, analytics, and other services can scale out independently of each other.

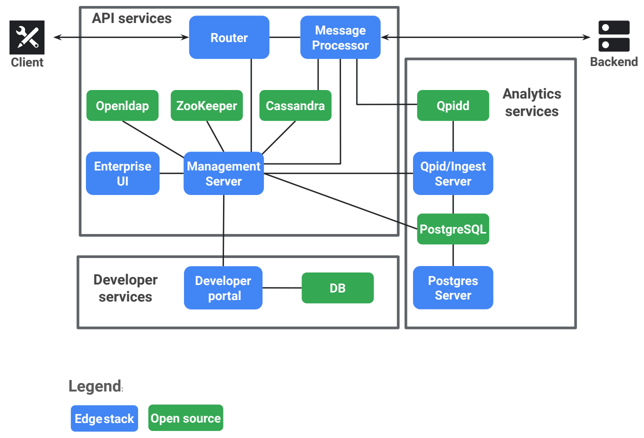


Depending on your API processing and data requirements, some of the supporting infrastructure components may also need to be scaled out.

Components associated with other capabilities such as the UI, system management and the developer portal, only need to scale based on requirements associated with those capabilities.

Gateway, infrastructure, analytics and other service components can scale out independently of each other.

Component view



- Router handles all incoming API traffic and dispatches it. The Router terminates the HTTP request, handles the SSL traffic.
- Message Processor handles API traffic for a specific organization and environment and executes all policies.
- Management Server offers an API to manage interaction between components.
- Cassandra stores application configurations and edge runtime data.
- ZooKeeper contains configuration data about all the services of the zone and notifies the different servers of configuration changes.
- OpenLDAP contains organization users and roles.
- Qpid/Ingest Server manages queuing system for analytics data.
- Postgres Server aggregates analytics raw data.

Here are the service components that make up Apigee Edge for private cloud.

Let's start with the critical path that requests to your APIs traverse on the platform.

On this diagram, the critical path is represented by the horizontal line extending from the client to the backend system.

The first component on this path is the router. The router is responsible for exposing virtual hosts and load balancing incoming API requests.

The message processor represents the runtime container for APIs executed on Apigee.

API calls executing on the message processors may perform one or more calls to backend systems. All calls to backend systems are originated by message processors.

Some APIs generate runtime data that needs to be stored within Apigee.

API policies such as API key validation, OAuth, key value map, and cache require access to a data store for the storage and retrieval of runtime data.

Apigee uses Cassandra as its runtime data store. Cassandra is part of the critical path when API proxies use policies that retrieve or store data during runtime execution.

As long as the router, message processor and Cassandra components are up and running, your API runtime availability is not generally impacted.

APIs that are executed on the message processors generate analytics events which

The Apigee Qpid server component is responsible for moving data from the Qpid queues to the postgreSQL database.

The raw analytics data in PostgreSQL is aggregated in batches by the Apigee Postgres server component.

This aggregated analytics data is used to power some of the Apigee Edge analytics reports.

In the center and left of the diagram, are the Apigee Edge management components.

The management server component exposes the management API. This API enables you to add users, deploy APIs, and perform many other management functions on the platform.

To perform these functions, the management server uses the Cassandra, ZooKeeper, and Openldap components to read and store relevant data that is associated with runtime or configuration state.

ZooKeeper stores data related to the configuration of the system that includes wiring information between components and configuration state.

Openldap stores information for role-based access control. Users, roles, and permissions that are needed for administrative access to the platform are stored here.

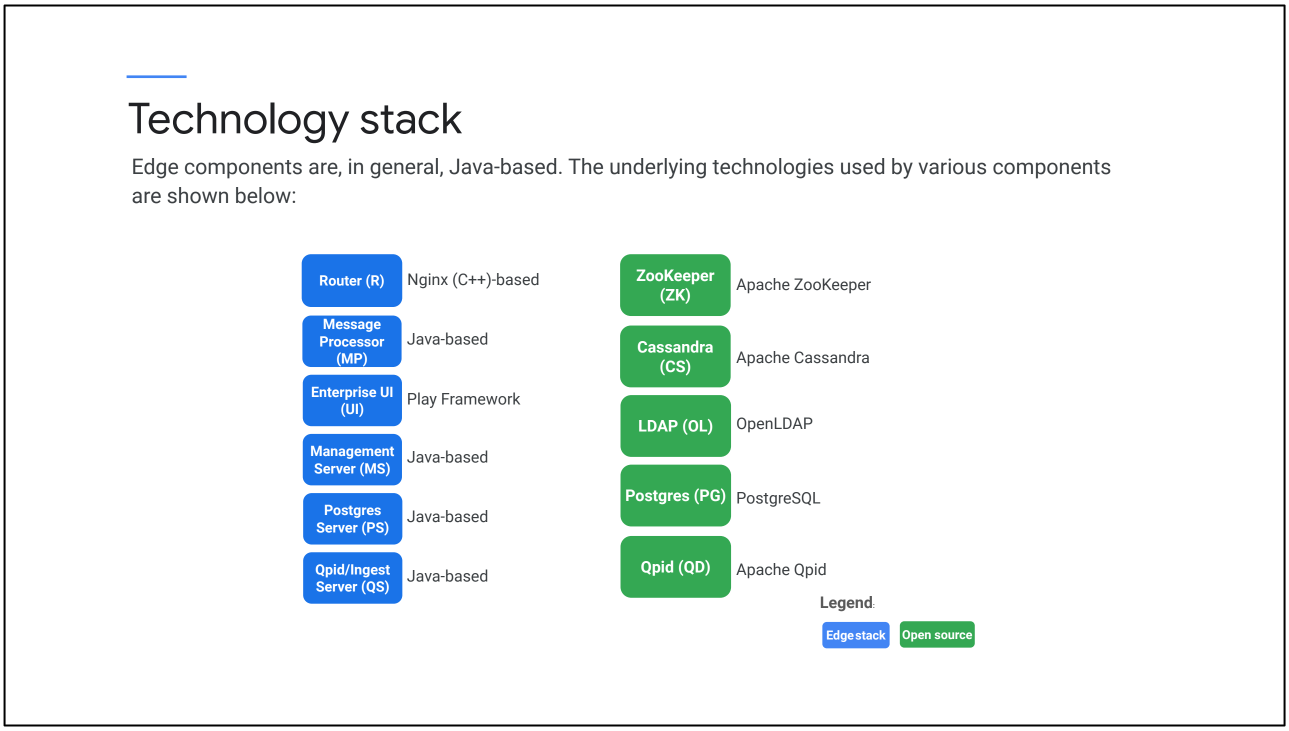
The Enterprise UI is the administrative or management UI that is used by API teams to develop and manage APIs. The UI consumes the management API to perform all of its functions.

All features in the Enterprise UI are also available through the management API.

The developer portal is a Drupal based web application that provides services to application developers that consume your APIs.

The portal uses its own dedicated database to store data about app developer user accounts, sessions, and modules.

Take some time to review the diagram to understand the relationship between the components and the functions they serve within the Apigee private cloud platform.



Technology stack

Edge components are, in general, Java-based. The underlying technologies used by various components are shown below:

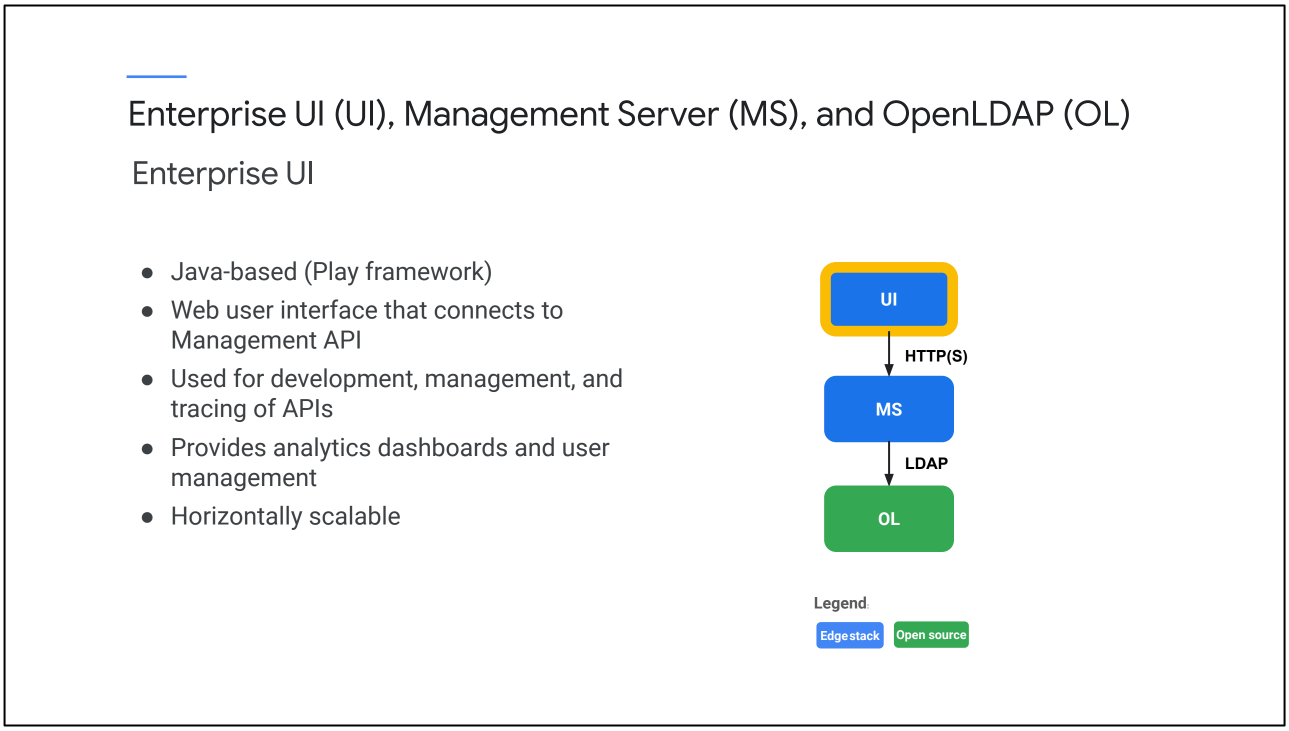
Router (R)	Nginx (C++)-based
Message Processor (MP)	Java-based
Enterprise UI (UI)	Play Framework
Management Server (MS)	Java-based
Postgres Server (PS)	Java-based
Qpid/Ingest Server (QS)	Java-based
ZooKeeper (ZK)	Apache ZooKeeper
Cassandra (CS)	Apache Cassandra
LDAP (OL)	OpenLDAP
Postgres (PG)	PostgreSQL
Qpid (QD)	Apache Qpid

Legend:

Edge stack | Open source

The Apigee Edge private cloud components shown in blue are developed by Apigee and mostly based on Java.

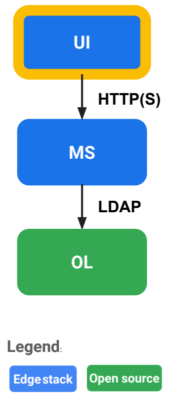
The components in green are stable open-source projects and used by the platform for specific purposes as discussed earlier.



Enterprise UI (UI), Management Server (MS), and OpenLDAP (OL)

Enterprise UI

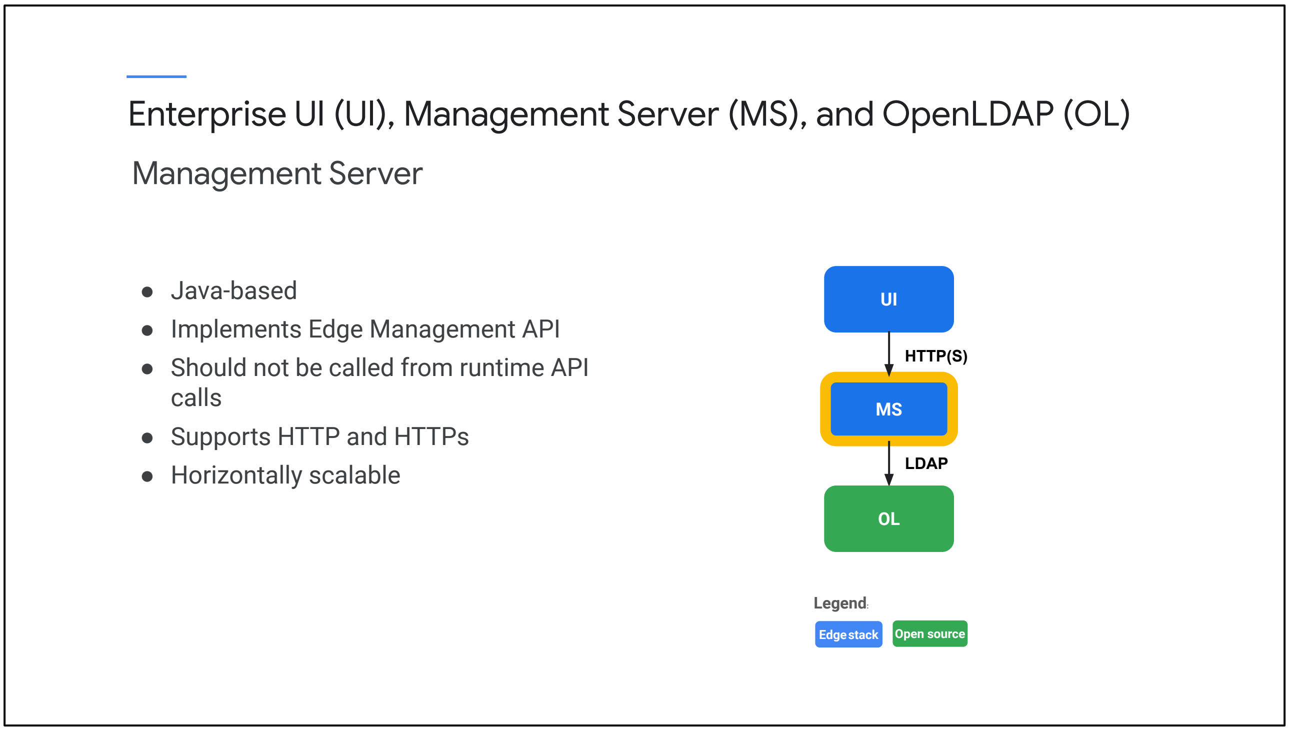
- Java-based (Play framework)
- Web user interface that connects to Management API
- Used for development, management, and tracing of APIs
- Provides analytics dashboards and user management
- Horizontally scalable



The Enterprise UI provides a web based interface to manage your Apigee organizations, environments, API proxies and other entities.

You can also view analytics data and manage user access.

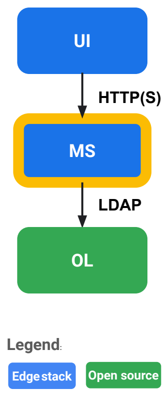
The Enterprise UI connects to the Management API and is horizontally scalable.



Enterprise UI (UI), Management Server (MS), and OpenLDAP (OL)

Management Server

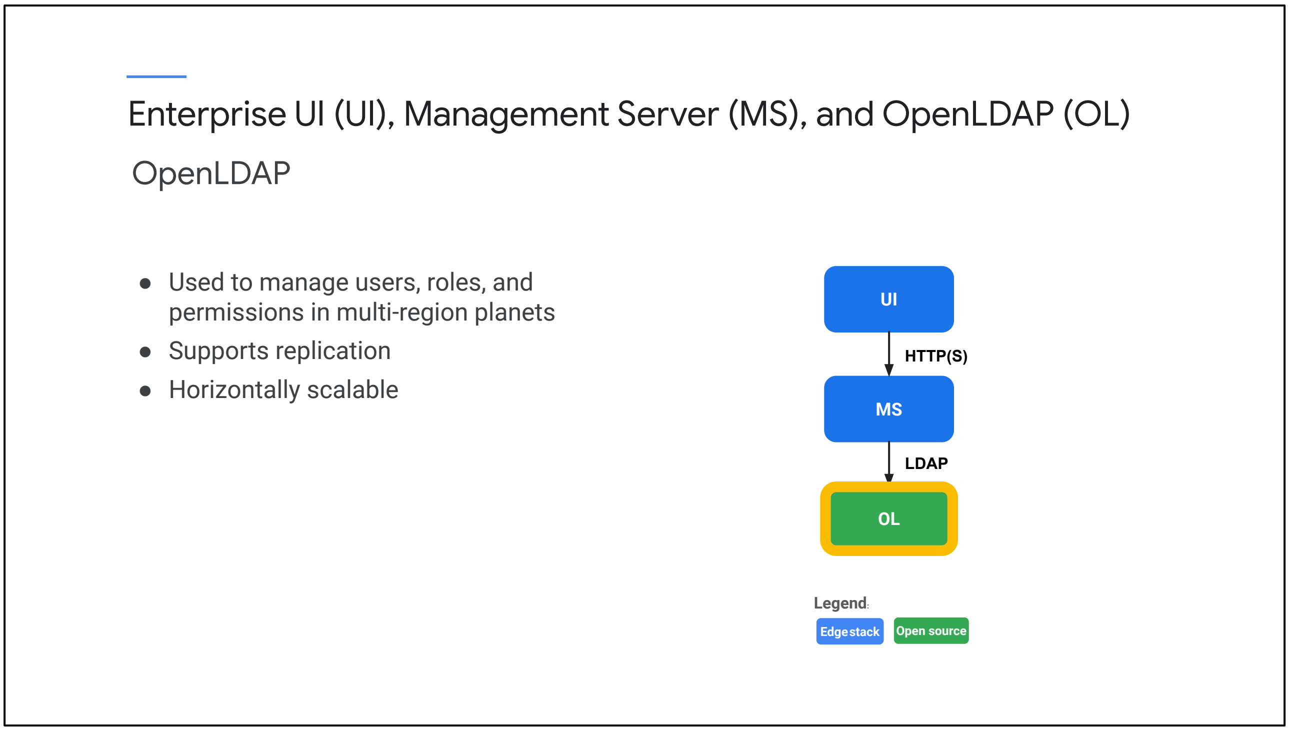
- Java-based
- Implements Edge Management API
- Should not be called from runtime API calls
- Supports HTTP and HTTPs
- Horizontally scalable



The Management Server is a Java based component that exposes the Apigee Edge management API.

You use the management API for configuration management, proxy deployment and other management functions.

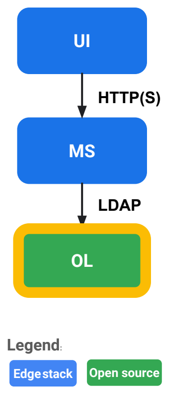
The Management Server uses HTTP or HTTPS for communication and is horizontally scalable.



Enterprise UI (UI), Management Server (MS), and OpenLDAP (OL)

OpenLDAP

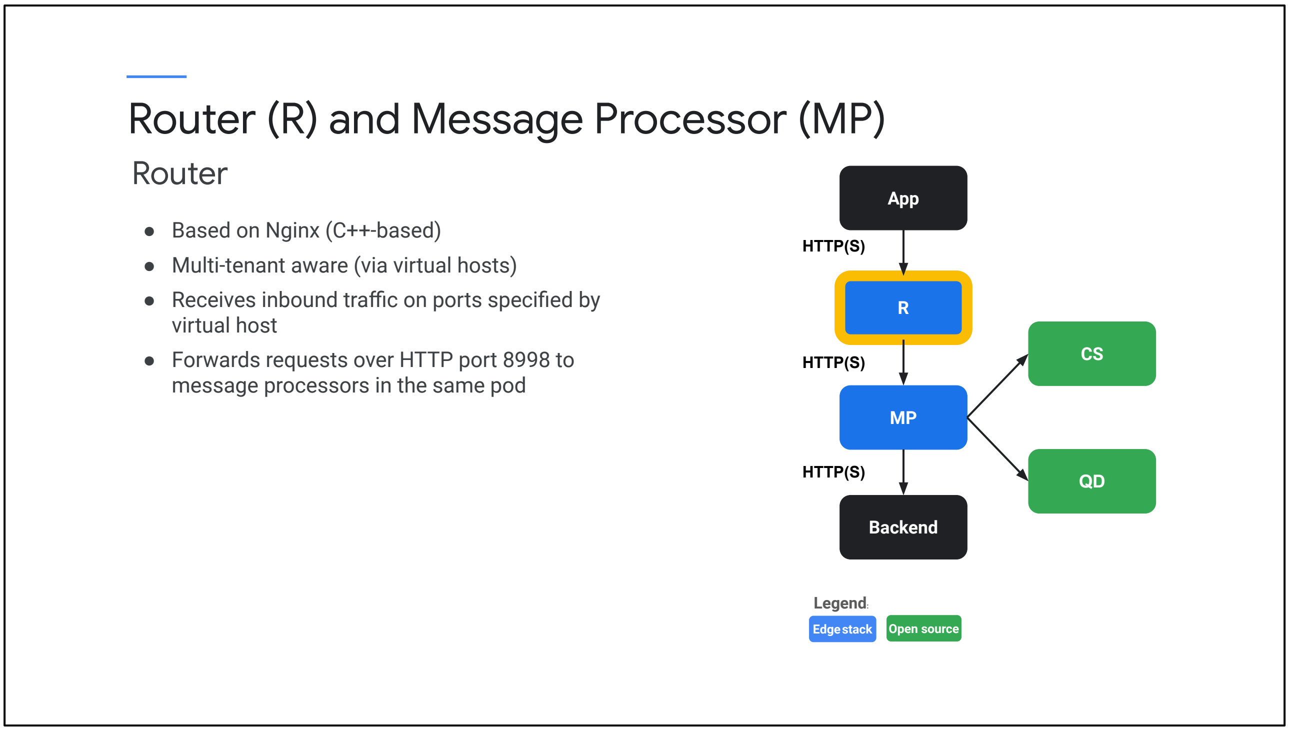
- Used to manage users, roles, and permissions in multi-region planets
- Supports replication
- Horizontally scalable



Openldap is an open-source implementation of the Lightweight Directory Access Protocol or LDAP.

Apigee Edge uses this component to implement role-based access control to the API platform.

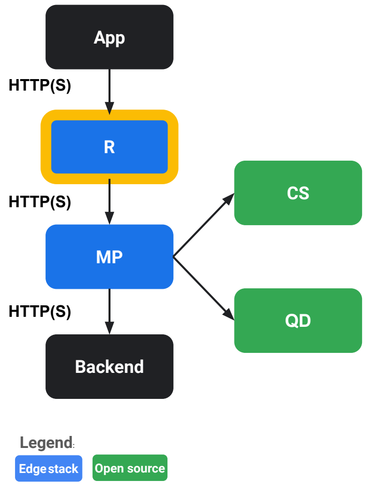
The Openldap component is horizontally scalable with its data replicated between other servers.



Router (R) and Message Processor (MP)

Router

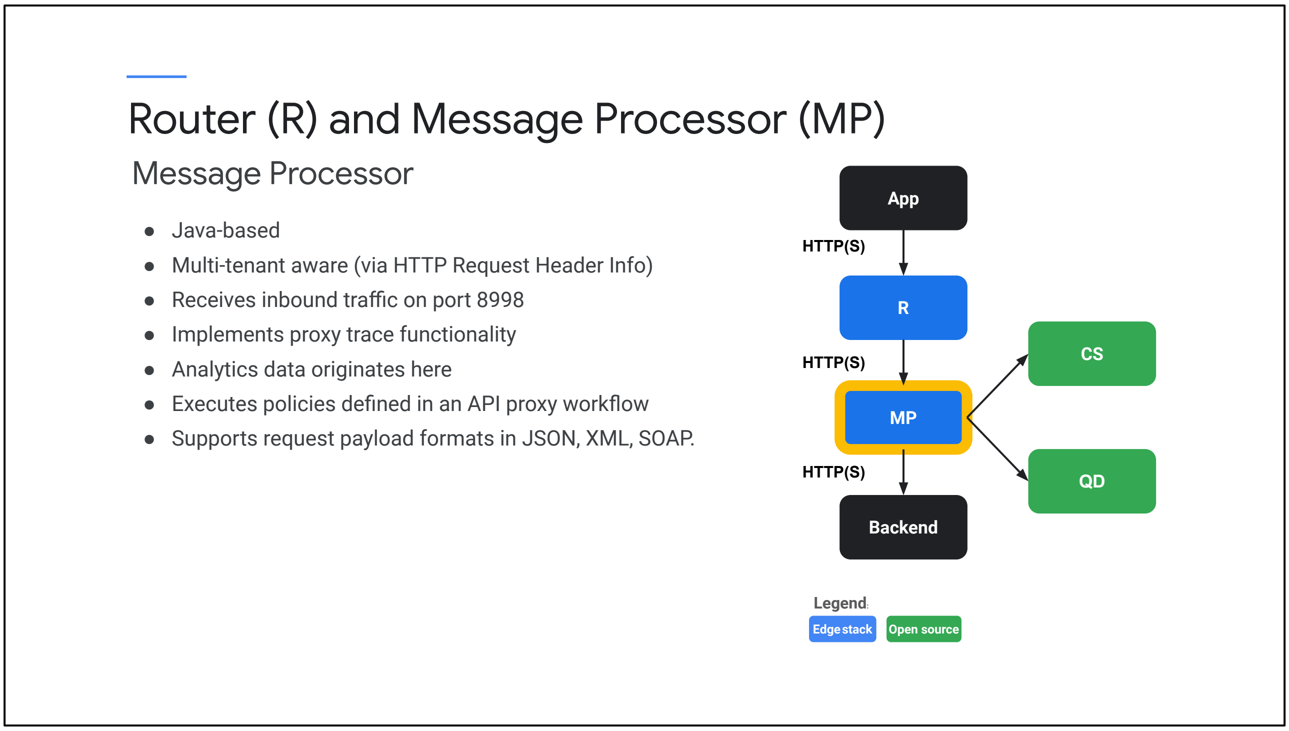
- Based on Nginx (C++-based)
- Multi-tenant aware (via virtual hosts)
- Receives inbound traffic on ports specified by virtual host
- Forwards requests over HTTP port 8998 to message processors in the same pod



The Router component is based on the highly performant open-source Nginx reverse proxy.

It accepts incoming requests from API client applications and routes them to the message processor components.

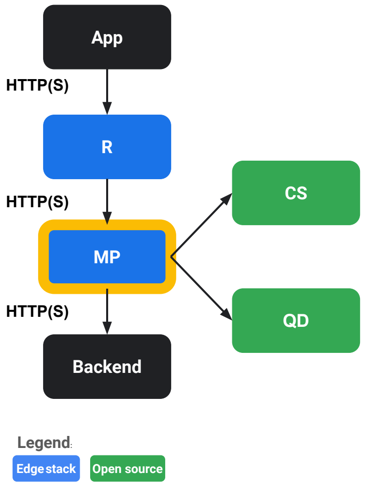
Request routing is based on the virtual host configuration in Apigee Edge.



Router (R) and Message Processor (MP)

Message Processor

- Java-based
- Multi-tenant aware (via HTTP Request Header Info)
- Receives inbound traffic on port 8998
- Implements proxy trace functionality
- Analytics data originates here
- Executes policies defined in an API proxy workflow
- Supports request payload formats in JSON, XML, SOAP.

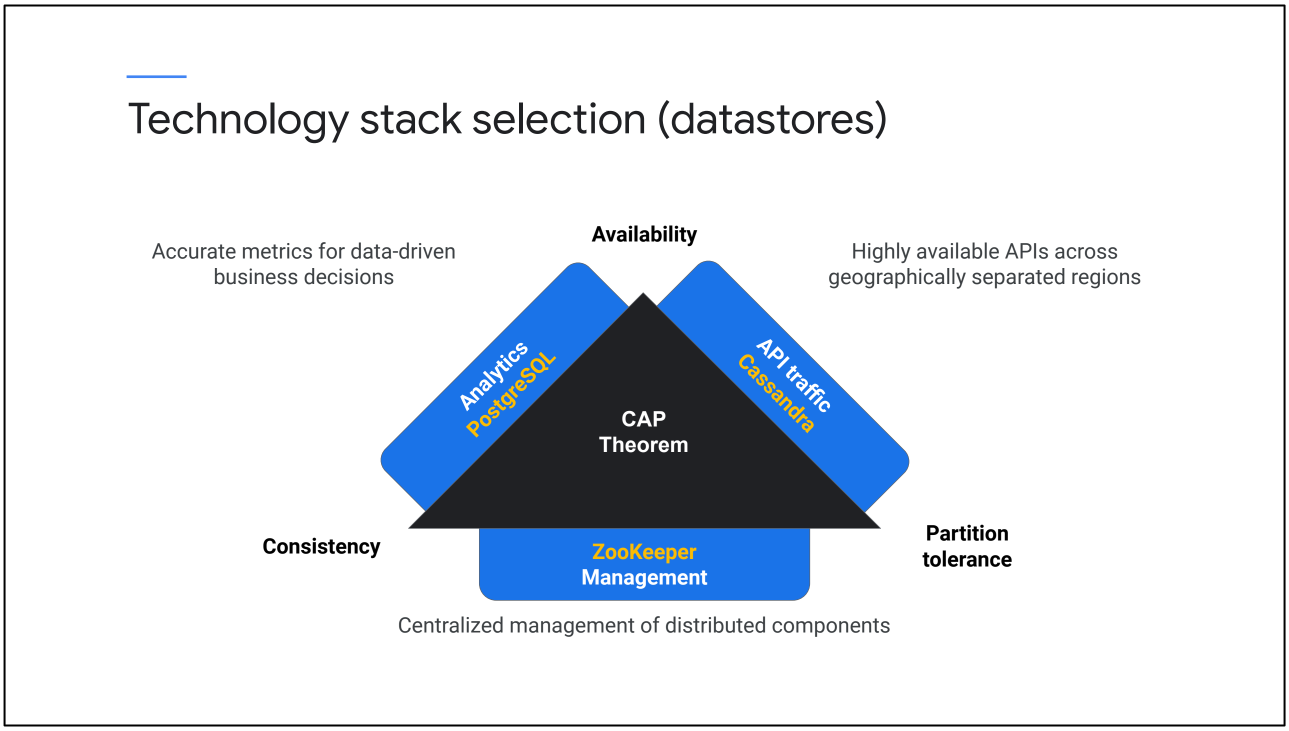


The Message Processor is an Apigee developed java based component.

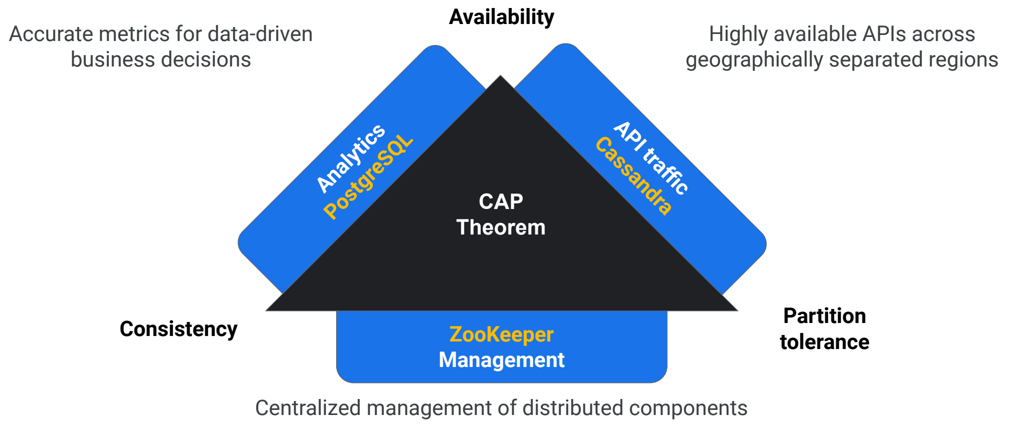
It is responsible for executing the logic of the API proxy through policies and custom code.

This component enables you to trace API requests in real-time, providing insight on policy execution and response times.

The message processor also collects analytics data for every client request.



Technology stack selection (datastores)


Availability
Accurate metrics for data-driven business decisions
Highly available APIs across geographically separated regions
Analytics
PostgreSQL
API traffic
Cassandra
CAP Theorem
Consistency
Partition tolerance
ZooKeeper
Management
Centralized management of distributed components

Apigee Edge for private cloud uses three different types of datastores to support its API processing, analytics and configuration management capabilities.

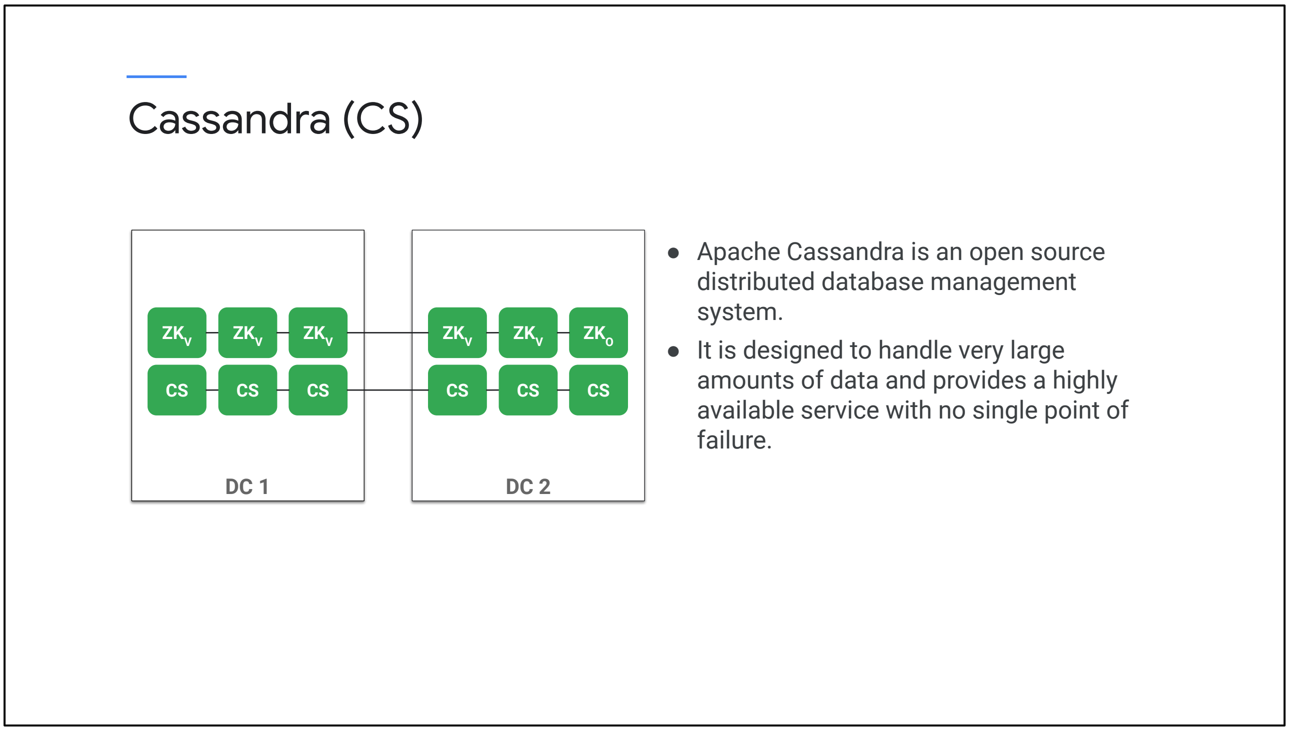
The CAP theorem states that it is impossible for a distributed computer system to simultaneously provide guarantees of:

Consistency, where all nodes see the same data at the same time,

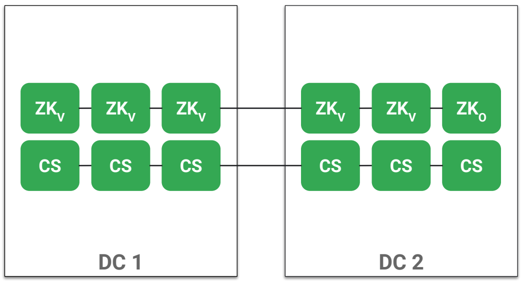
Availability, that every request receives a response about whether it succeeded or failed and

Partition tolerance, where the system continues to operate despite arbitrary message loss or failure of part of the system.

To support these properties for the different capabilities, Apigee Edge employs three different types of datastores: Cassandra, ZooKeeper and PostgreSQL.



Cassandra (CS)



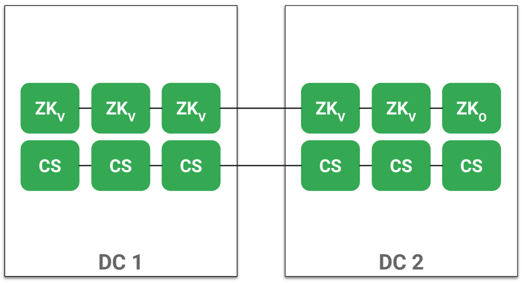
- Apache Cassandra is an open source distributed database management system.
- It is designed to handle very large amounts of data and provides a highly available service with no single point of failure.

Apache Cassandra is a distributed NoSQL database that is used by Apigee Edge as its runtime data store.

It can store large amounts of data and provides a highly available service with no single point of failure.

Cassandra (CS)

Cassandra characteristics

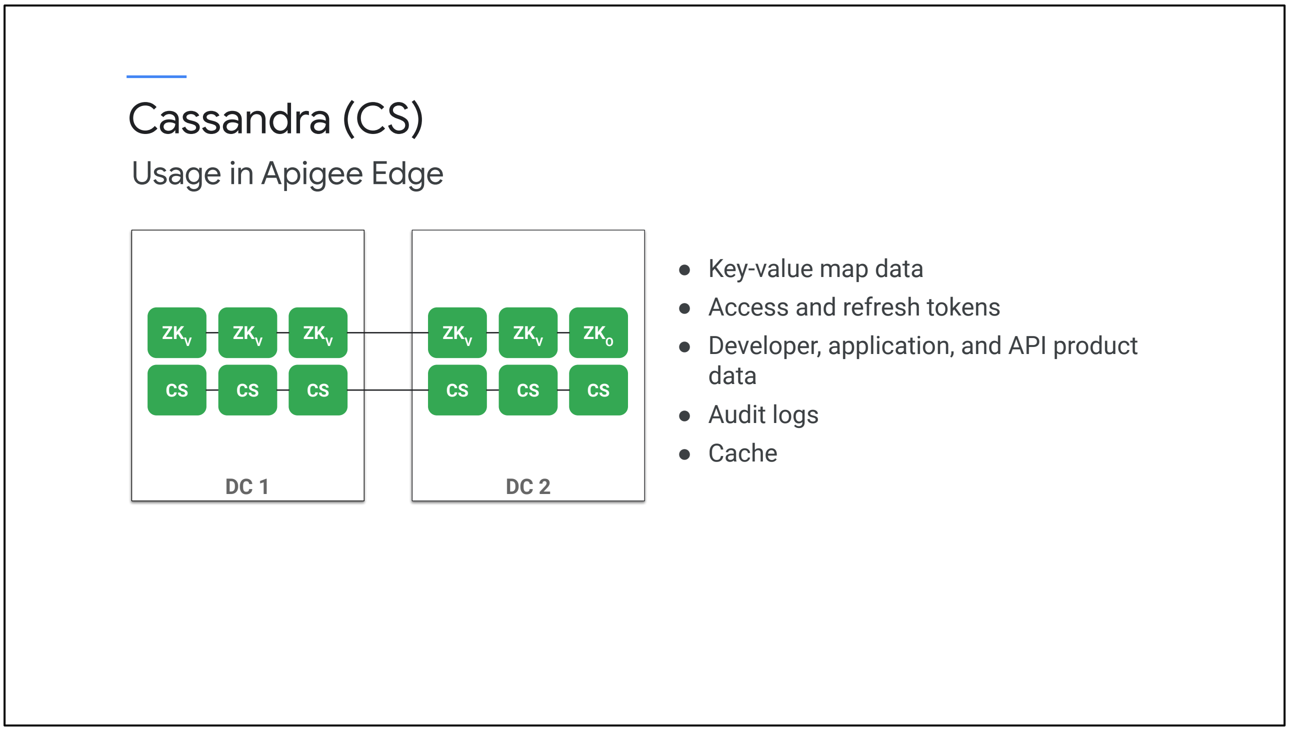


- All nodes in a Cassandra ring are equal
- An application can read from and write data to any node
- Eventual consistency
- Multiple copies of same data
- Read and write consistency is managed by the application

There isn't a single point of failure because applications could read from and write data to any Cassandra node.

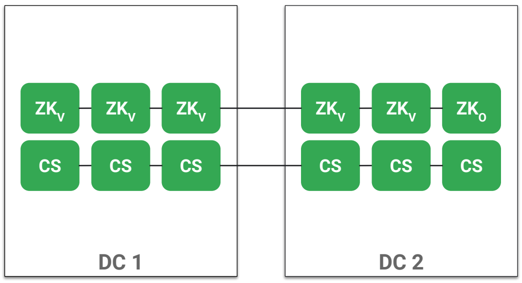
Every transaction can have a different consistency level that is managed by the application.

You can also configure how many copies of your data you can have in the Cassandra cluster.



Cassandra (CS)

Usage in Apigee Edge



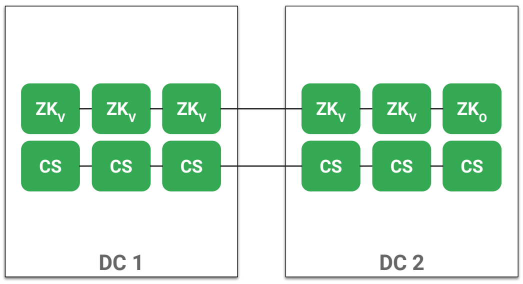
- Key-value map data
- Access and refresh tokens
- Developer, application, and API product data
- Audit logs
- Cache

Apigee Edge stores various types of data in Cassandra, that include:

- Developer and application data
- API products
- Oauth tokens
- Cache and KVM data
- API proxy bundles
- Audit logs and

other kinds of data that is needed for API proxy execution.

ZooKeeper (ZK)



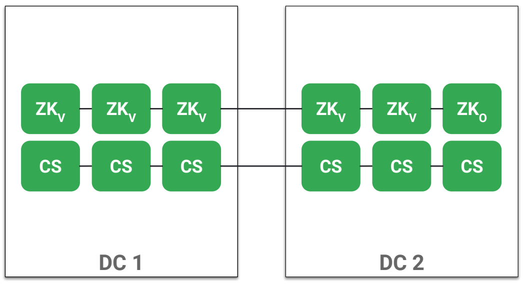
Apache ZooKeeper is a data store that provides a:

- distributed configuration service
- synchronization service
- naming registry for large distributed systems.

ZooKeeper is a centralized configuration management system that contains configuration data of the various Apigee components, and is used to synchronize the different services with configuration changes made to the platform.

ZooKeeper (ZK)

Usage in Apigee Edge

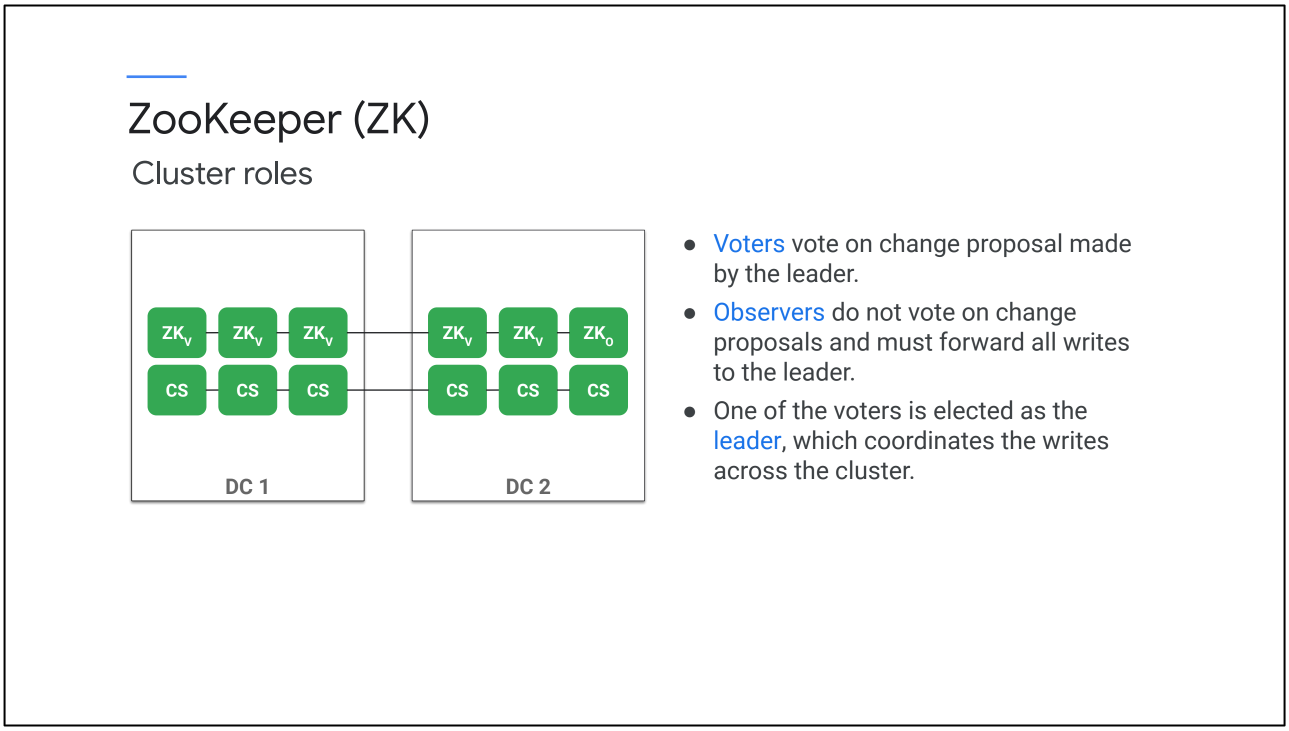


- Distributed configuration registry
- Component location map
- Stores status data
- NOT required to process API requests

All Apigee cluster configuration data is stored in ZooKeeper.

It is used to store configuration data about the location and configuration of the various Apigee components.

Note that message processors do not access ZooKeeper to process runtime API traffic.



ZooKeeper (ZK)

Cluster roles

- Voters vote on change proposal made by the leader.
- Observers do not vote on change proposals and must forward all writes to the leader.
- One of the voters is elected as the leader, which coordinates the writes across the cluster.

A ZooKeeper cluster is called an ensemble. The ZooKeeper service is available as long as a majority of the ensemble nodes are up so it is best to use an odd number of nodes in the cluster.

For example, with four nodes ZooKeeper can only handle the failure of a single node. If two nodes fail, the remaining two do not constitute a majority. With five nodes, ZooKeeper can handle the failure of two of those nodes in the cluster.

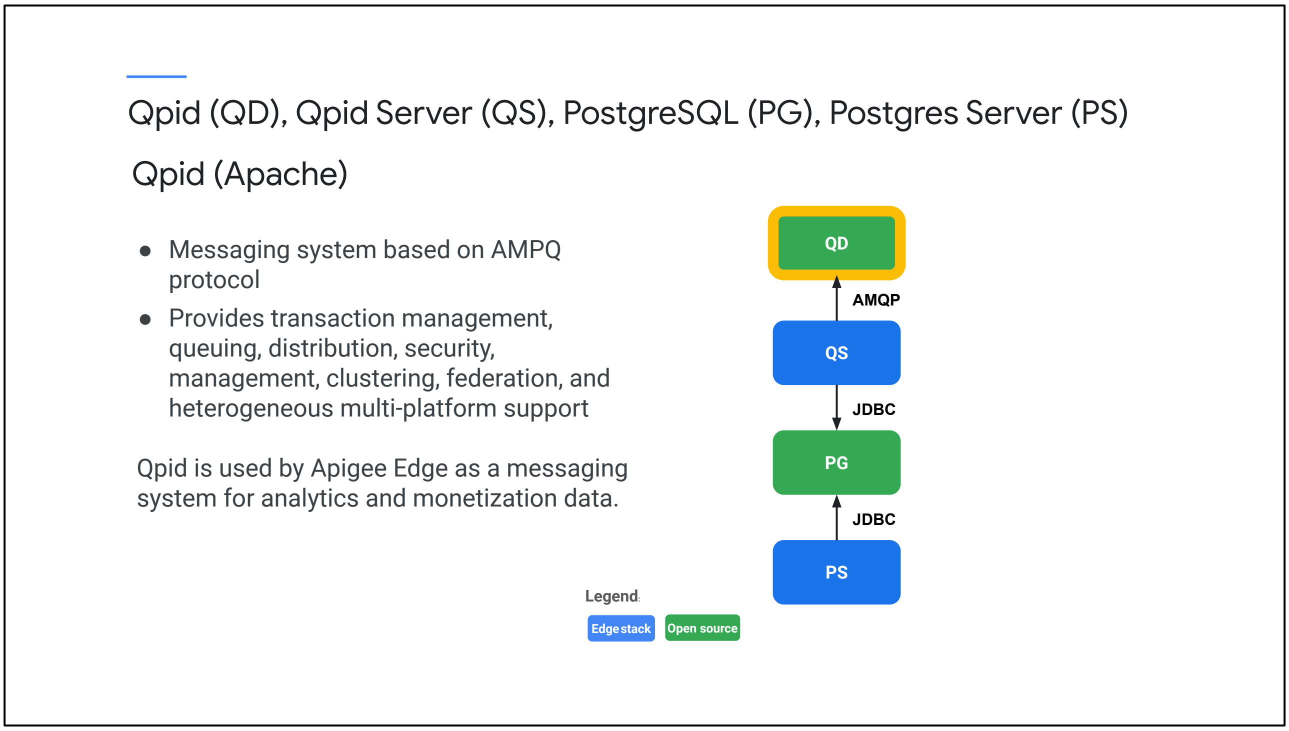
Note that a minimum of three nodes are required for a fault tolerant setup.

There are three possible roles for the ZooKeeper nodes in the ensemble:

A node that can vote on a change proposal made by the **leader** node is called a **voter**.

One of the voters is elected as the **leader**, and is responsible for controlling the coordination of writes across the distributed ZooKeeper nodes, while the rest of the voters are **followers**.

Another type of node is the **Observer** that is used to improve scalability of the cluster. Observers do not vote on change proposals and must forward all writes to the Leader. They are not a critical part of the ensemble because they do not vote.



Qpid (QD), Qpid Server (QS), PostgreSQL (PG), Postgres Server (PS)

Qpid (Apache)

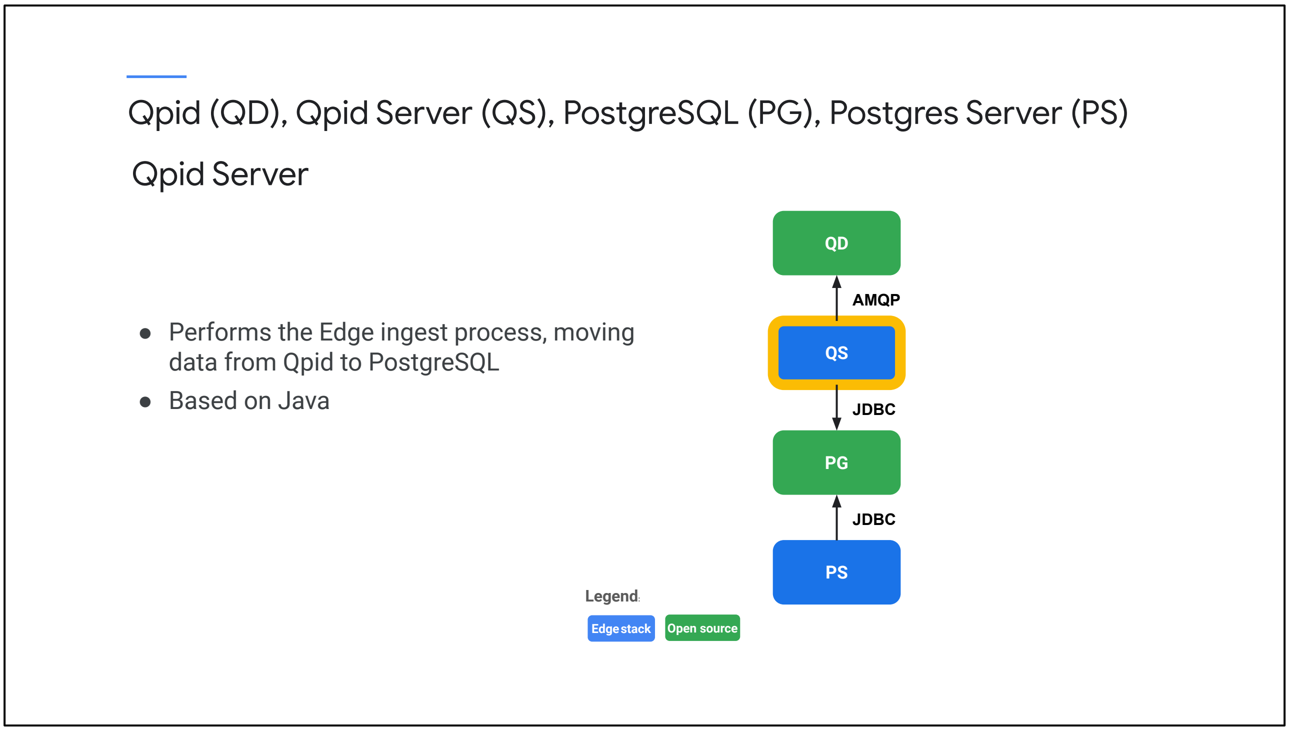
- Messaging system based on AMPQ protocol
- Provides transaction management, queuing, distribution, security, management, clustering, federation, and heterogeneous multi-platform support

Qpid is used by Apigee Edge as a messaging system for analytics and monetization data.

QD
AMQP
QS
JDBC
PG
JDBC
PS
Legend:
Edge stack
Open source

Apache Qpid is an open-source messaging system that implements the Advanced Message Queuing Protocol.

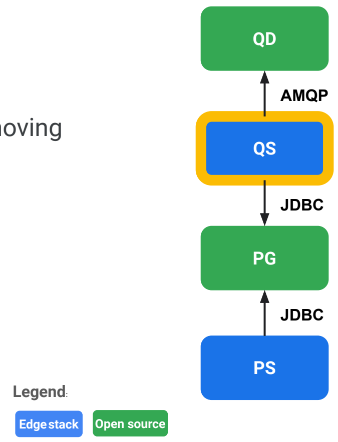
It provides transaction management and queuing for analytics and monetization data that is generated by the Message Processor.



Qpid (QD), Qpid Server (QS), PostgreSQL (PG), Postgres Server (PS)

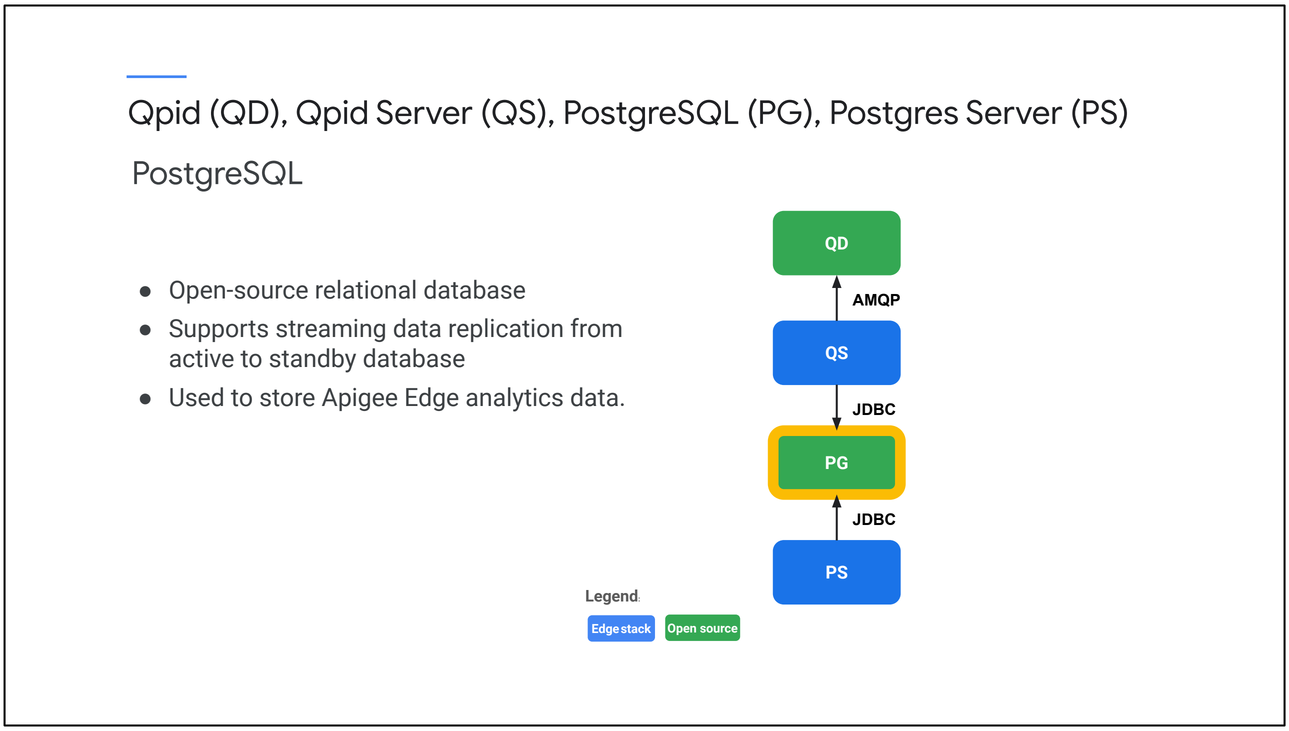
Qpid Server

- Performs the Edge ingest process, moving data from Qpid to PostgreSQL
- Based on Java



The Qpid Server is java-based custom component used in Apigee Edge.

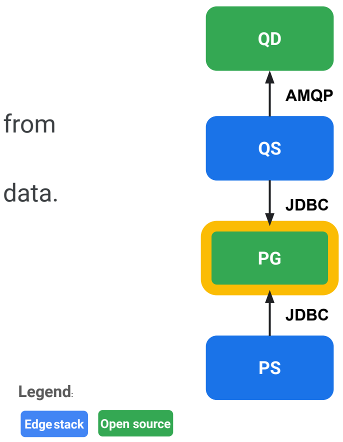
It ingests analytics and monetization data from Apache Qpid queues to the PostgreSQL database.



Qpid (QD), Qpid Server (QS), PostgreSQL (PG), Postgres Server (PS)

PostgreSQL

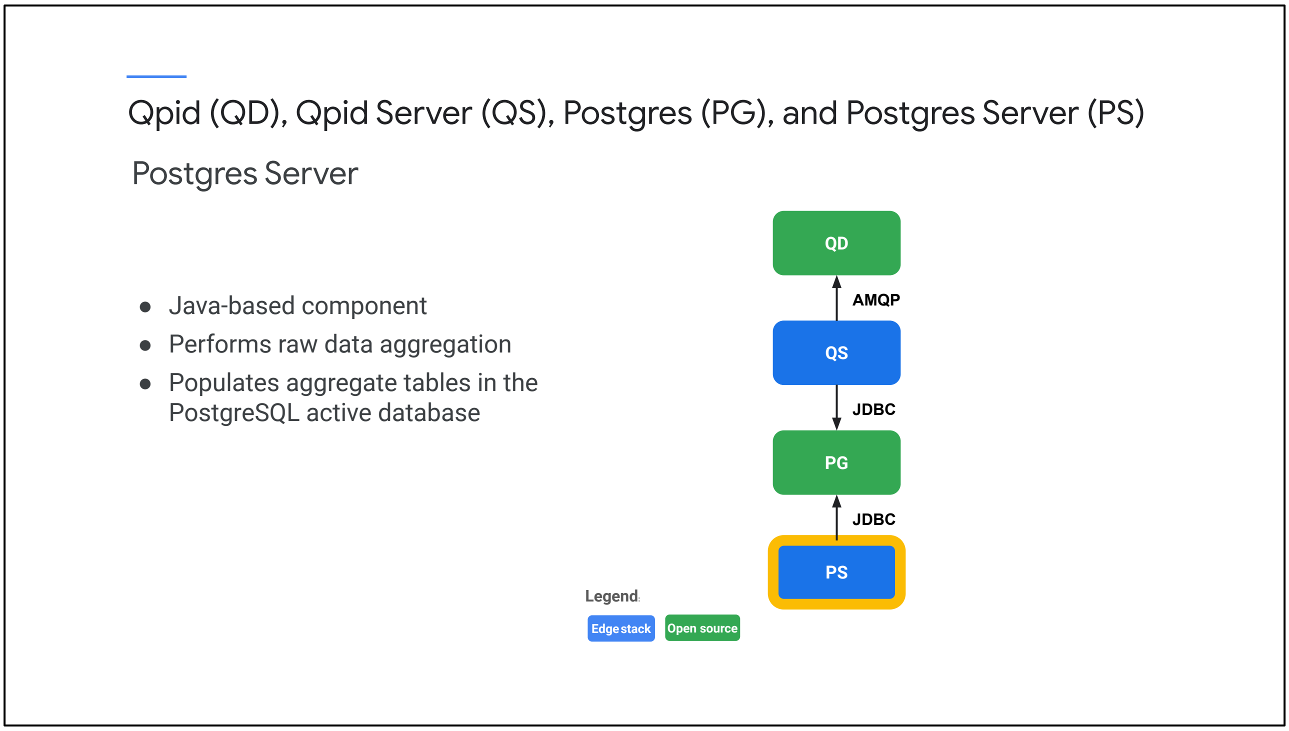
- Open-source relational database
- Supports streaming data replication from active to standby database
- Used to store Apigee Edge analytics data.



PostgreSQL is an open-source relational database management system that is used to store all Apigee analytics and monetization data in Apigee Edge.

It provides high availability thorough data replication and an active-standby mode of operation.

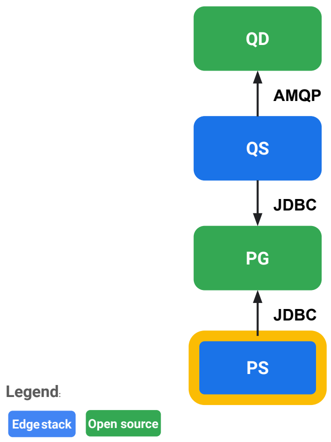
Data is streamed from active to standby servers in the form of Write-Ahead-Log files.



Qpid (QD), Qpid Server (QS), Postgres (PG), and Postgres Server (PS)

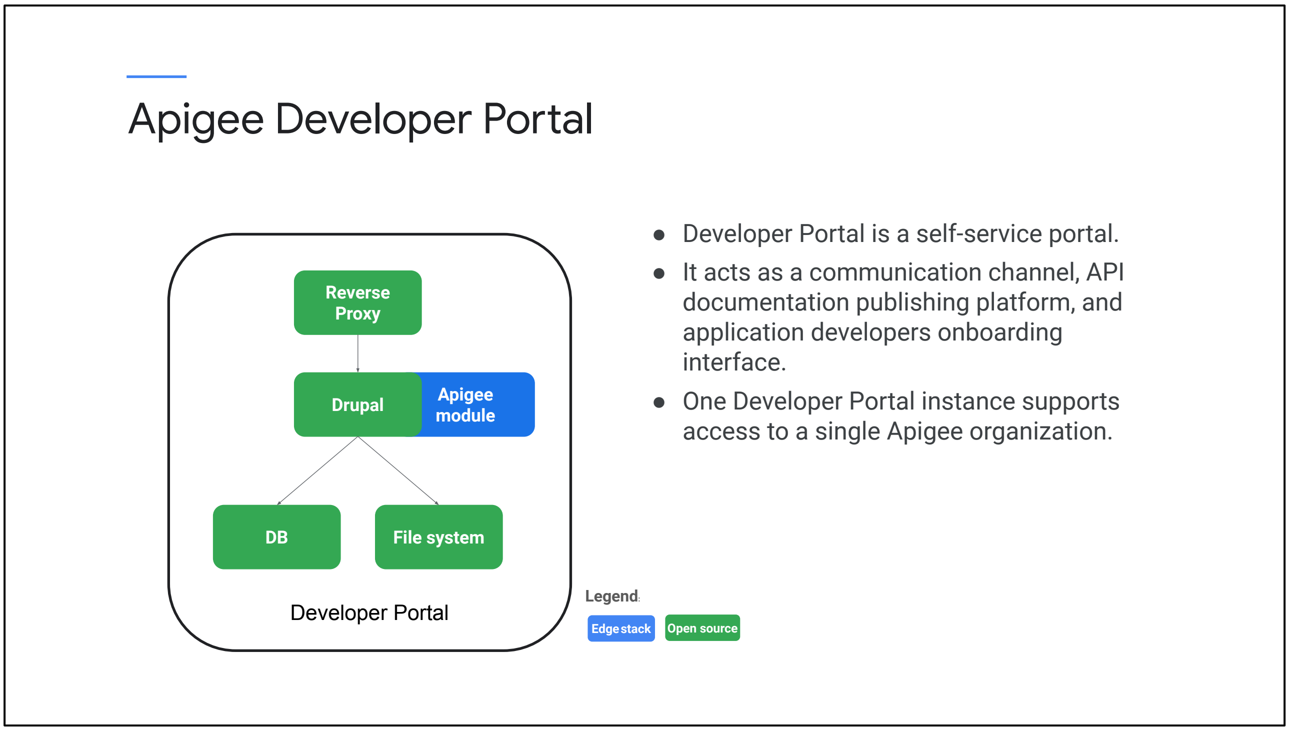
Postgres Server

- Java-based component
- Performs raw data aggregation
- Populates aggregate tables in the PostgreSQL active database

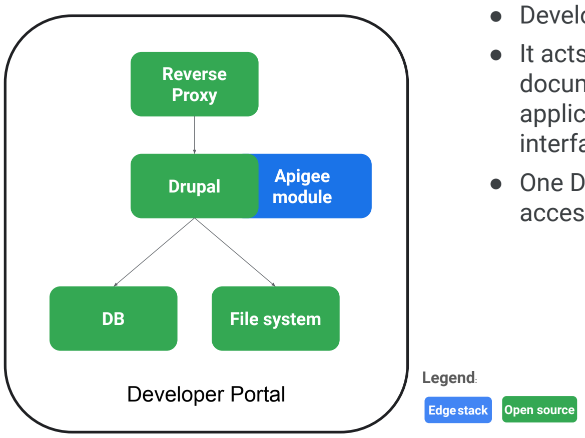


Postgres Server is an Apigee Edge java based component that performs data aggregation of analytics data in the PostgreSQL active database.

It creates default views of aggregated data tables for use in the Enterprise UI.



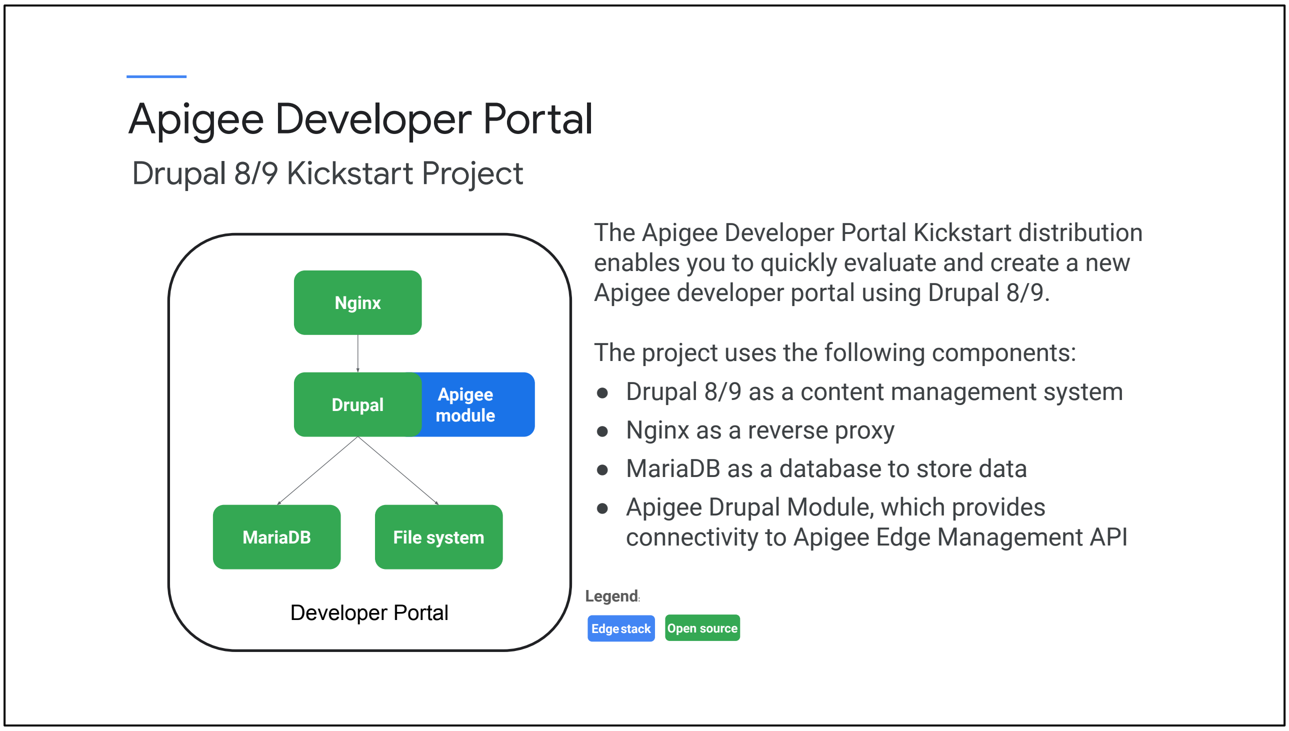
Apigee Developer Portal



- Developer Portal is a self-service portal.
- It acts as a communication channel, API documentation publishing platform, and application developers onboarding interface.
- One Developer Portal instance supports access to a single Apigee organization.

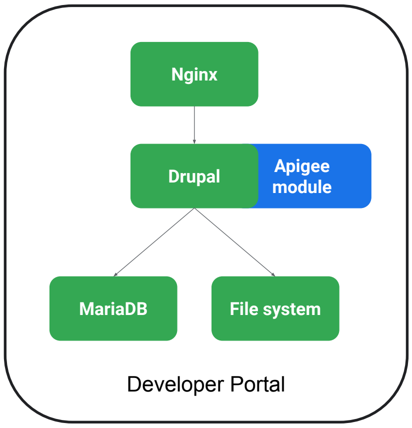
The Apigee Developer portal is based on the Drupal content management system. It is a self-service portal for use by application developers who register to use your APIs.

One developer portal connects to one Apigee organization.



Apigee Developer Portal

Drupal 8/9 Kickstart Project



The Apigee Developer Portal Kickstart distribution enables you to quickly evaluate and create a new Apigee developer portal using Drupal 8/9.

The project uses the following components:

- Drupal 8/9 as a content management system
- Nginx as a reverse proxy
- MariaDB as a database to store data
- Apigee Drupal Module, which provides connectivity to Apigee Edge Management API

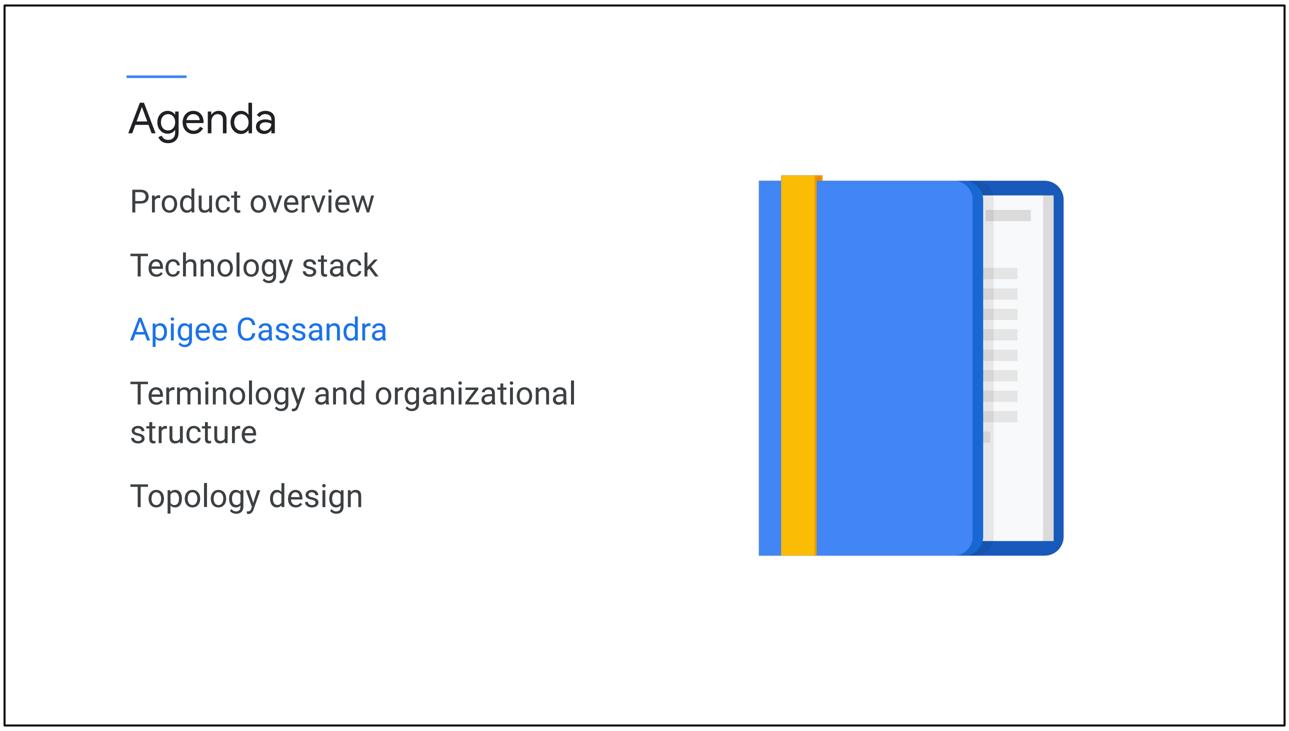
Legend:

🟦 Edge stack 🟩 Open source

You can quickly set up the developer portal using the Apigee Developer Portal Kickstart Project.

The project is a Drupal distribution that enables you to evaluate the portal using Drupal 8 or 9, and provides all the requirements and installation procedures.

The project uses MariaDB, Nginx, Drupal 8 or 9 and the Apigee Drupal modules.



Agenda

Product overview

Technology stack

Apigee Cassandra

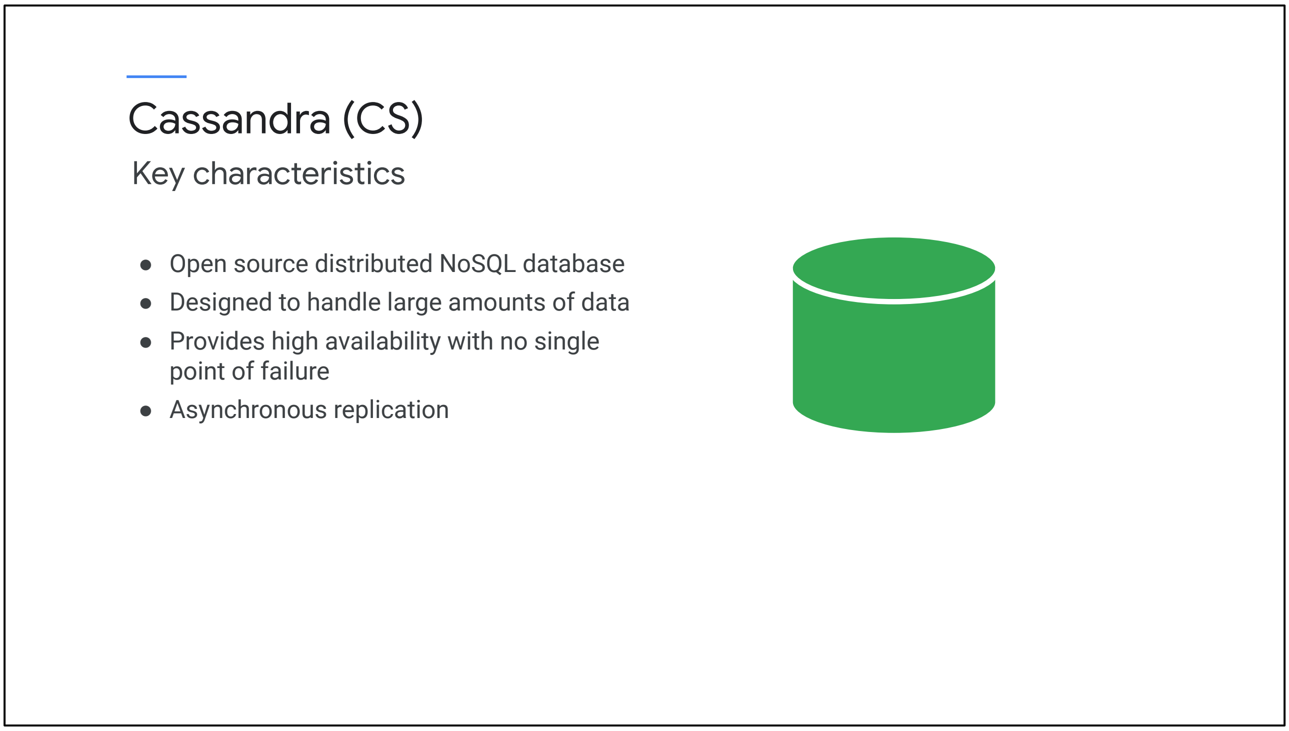
Terminology and organizational structure

Topology design

As mentioned earlier, Apigee Edge uses Cassandra as its runtime data store.

Cassandra is part of the critical path because it is used by API policies to store data during runtime execution.

This lesson teaches you about some key Cassandra concepts, and how they are used by the Apigee private cloud platform.



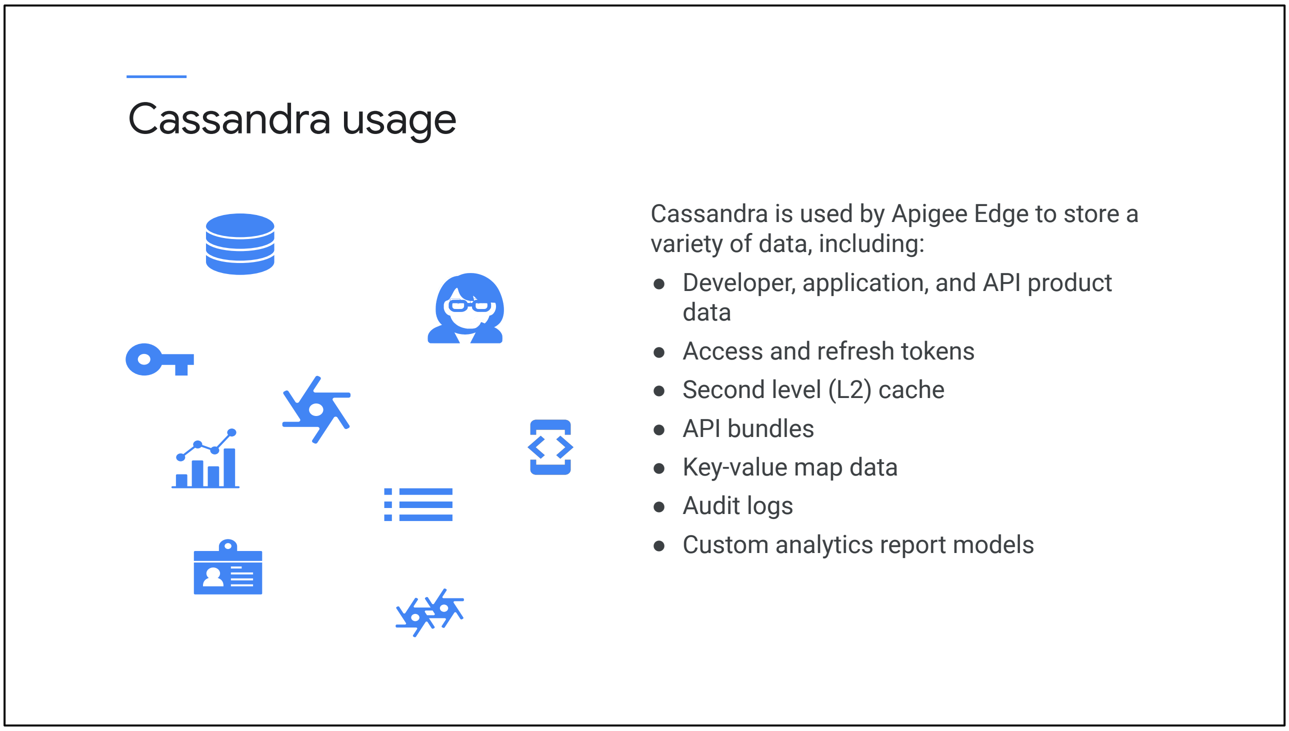
Cassandra (CS)

Key characteristics

- Open source distributed NoSQL database
- Designed to handle large amounts of data
- Provides high availability with no single point of failure
- Asynchronous replication

Apache Cassandra is a distributed NoSQL database with no single point of failure.

It is designed to handle large amounts of data that is replicated asynchronously.



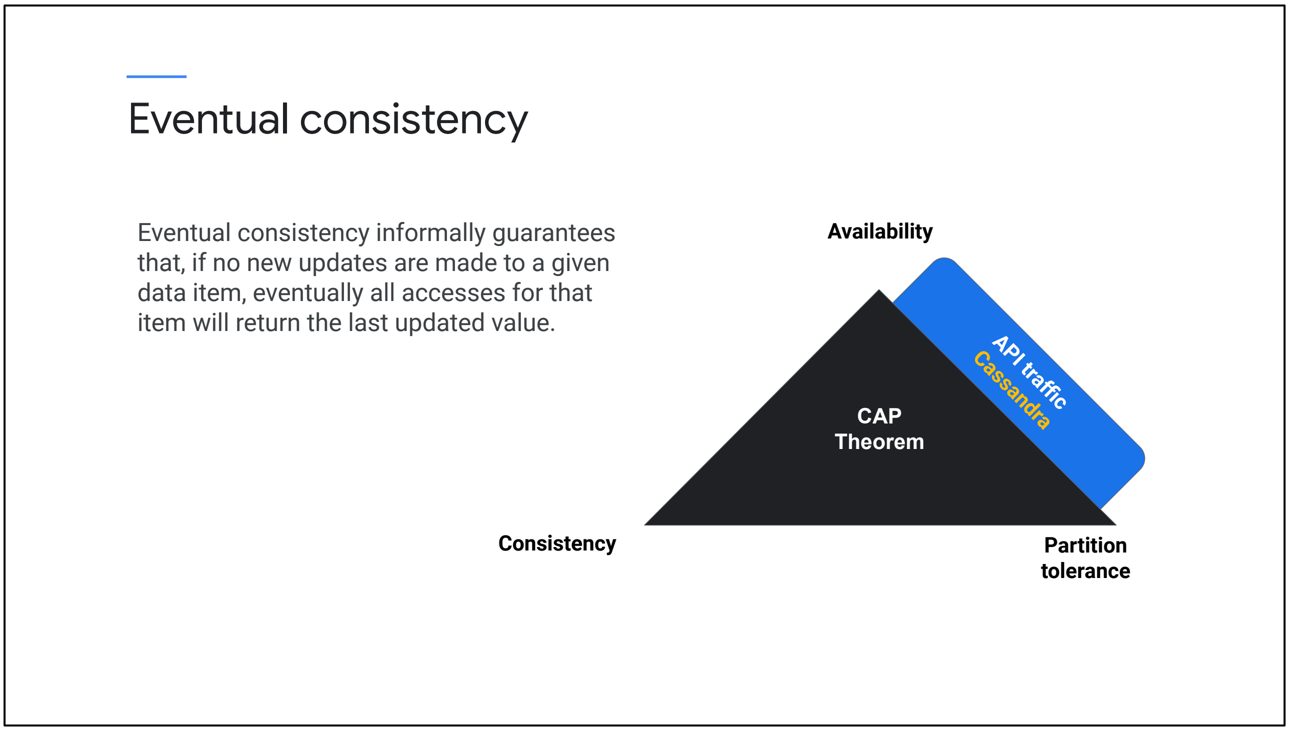
Cassandra usage

Cassandra is used by Apigee Edge to store a variety of data, including:

- Developer, application, and API product data
- Access and refresh tokens
- Second level (L2) cache
- API bundles
- Key-value map data
- Audit logs
- Custom analytics report models

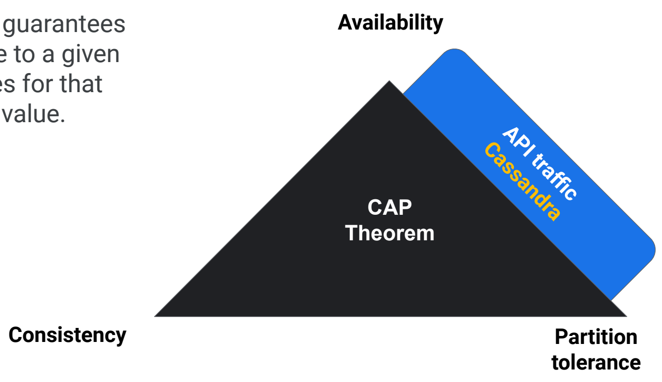
Apigee Edge stores various types of data in Cassandra including:

Developer, application and API Product data
Access and refresh tokens
Second level (L2) cache data
API proxy bundles
Key-Value map data
Audit logs and
Custom analytics report models.



Eventual consistency

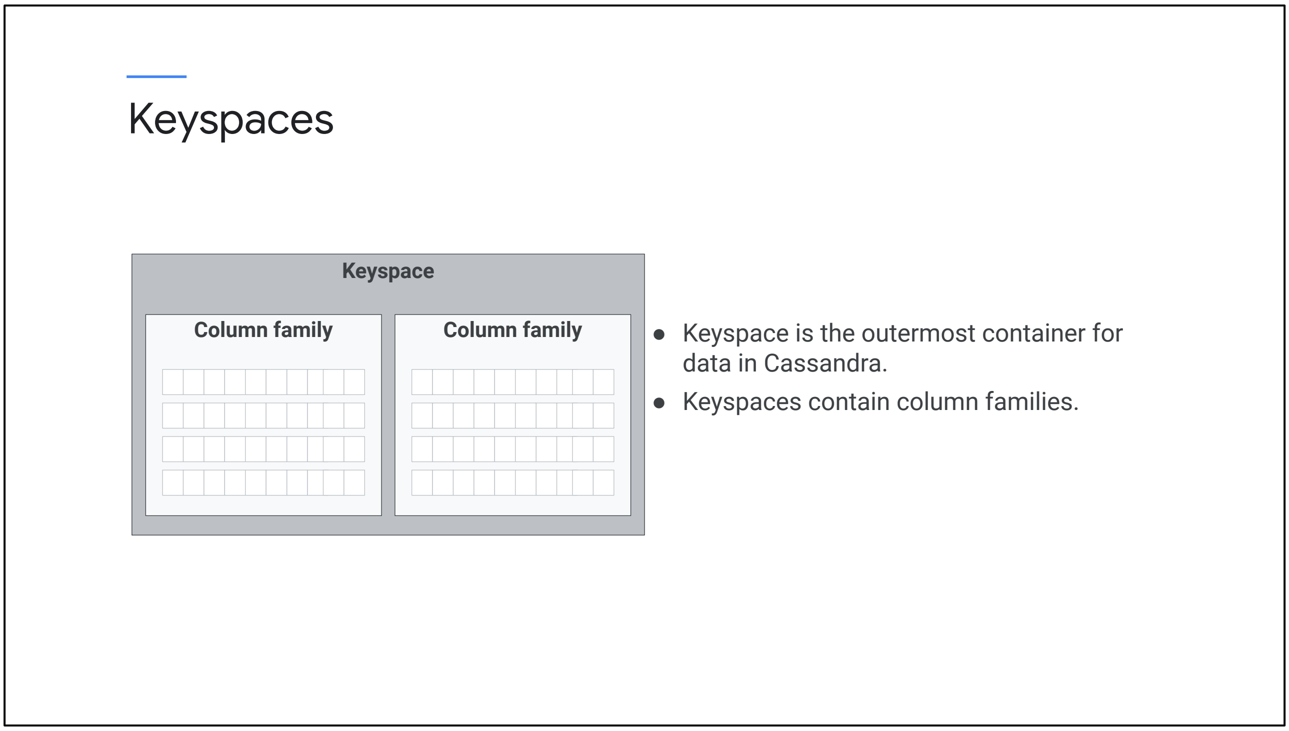
Eventual consistency informally guarantees that, if no new updates are made to a given data item, eventually all accesses for that item will return the last updated value.



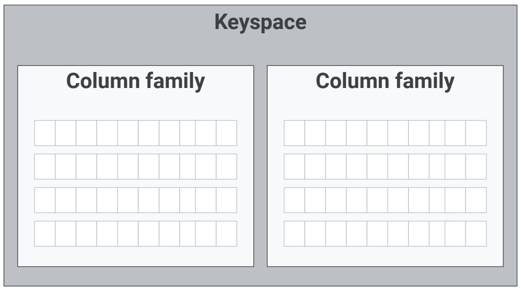
Cassandra provides high availability and functions even in split-brain types of disasters. Split brain is a term used to describe a failure condition when servers do not communicate and synchronize their data with each other.

To achieve this, it uses eventual consistency that implies that all updates reach all replicas eventually.

This means that for any given set of inputs, we will eventually know the state of the database, and any further reads will be consistent with the expected output.



Keyspaces

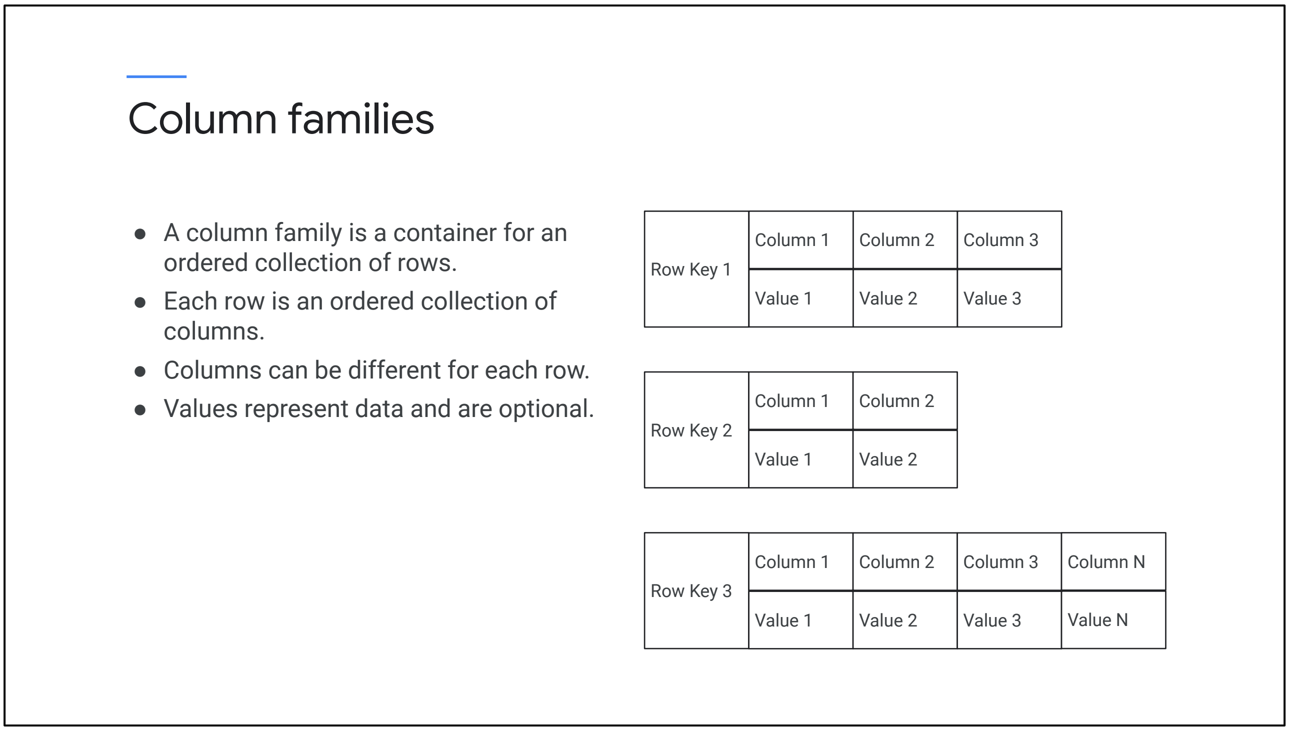


- Keyspace is the outermost container for data in Cassandra.
- Keyspaces contain column families.

Cassandra uses objects called keyspaces that controls the replication for the object it contains at each datacenter in the cluster.

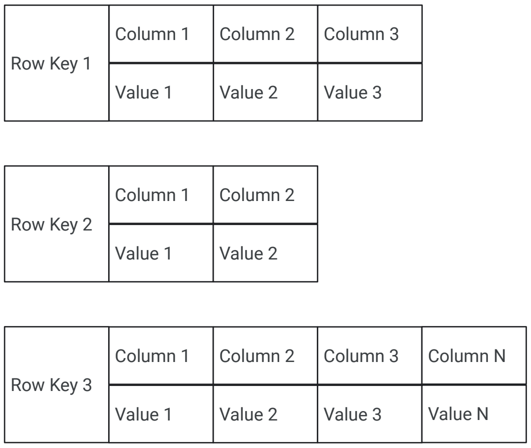
The keyspace defines how a dataset is replicated in the cluster.

A keyspace is analogous to a relational database schema and contains column families.



Column families

- A column family is a container for an ordered collection of rows.
- Each row is an ordered collection of columns.
- Columns can be different for each row.
- Values represent data and are optional.

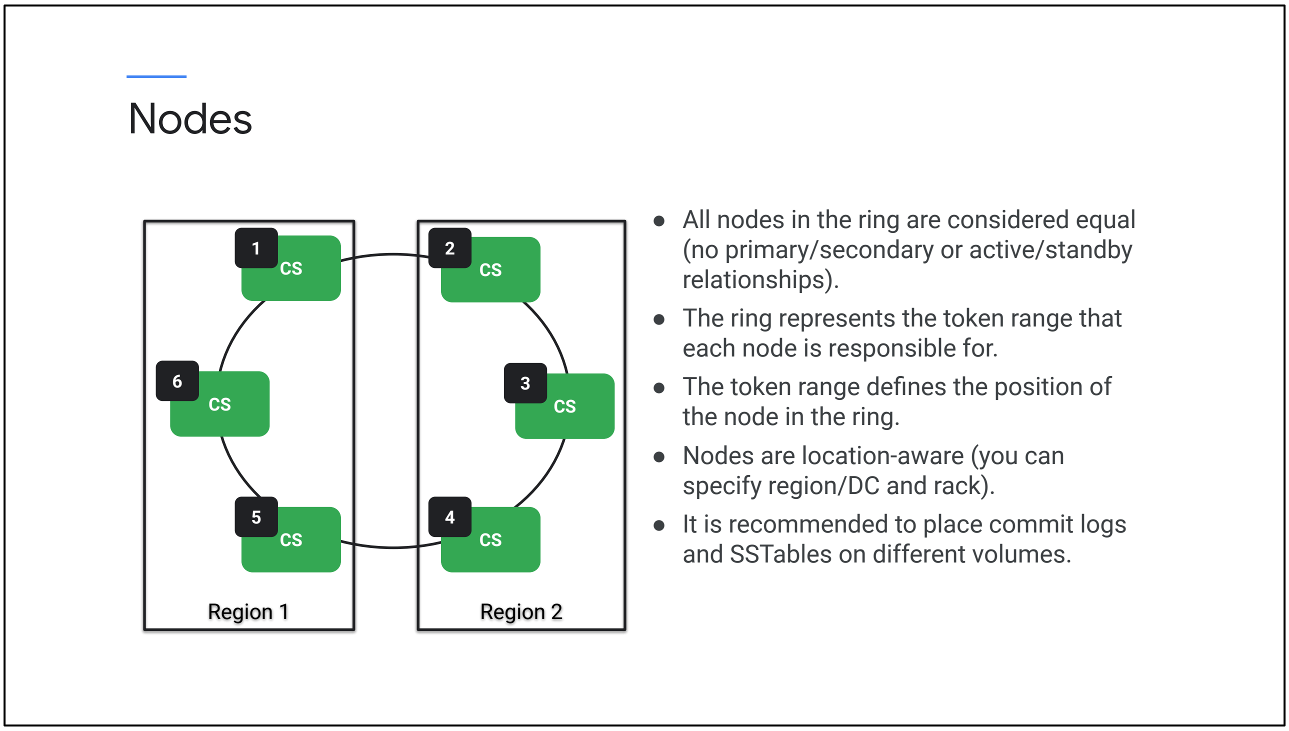


A column family contains an ordered collection of rows.

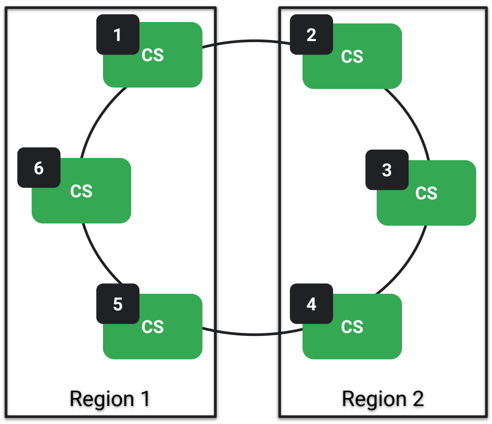
Each row is an ordered collection of columns.

You can add any column to a column family at any time.

The row key or primary key in the column family must be unique and is used to identify rows.



Nodes



- All nodes in the ring are considered equal (no primary/secondary or active/standby relationships).
- The ring represents the token range that each node is responsible for.
- The token range defines the position of the node in the ring.
- Nodes are location-aware (you can specify region/DC and rack).
- It is recommended to place commit logs and SSTables on different volumes.

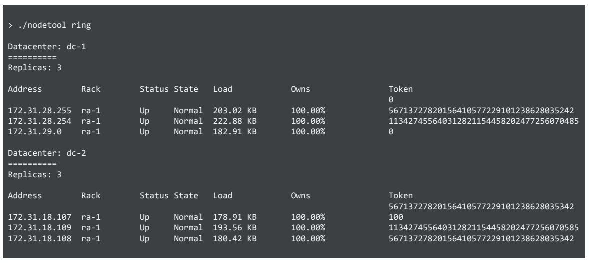
A Cassandra node is one instance of the Cassandra database.

A rack is a logical set of nodes. A data center is a logical set of racks. A ring is the complete set of nodes which map to a single token ring.

Data is stored on the Cassandra node.

Tokens and ranges

- Each node in the cluster is assigned and responsible for a token range.
- A token range is defined by a partitioner.
- Maximum range size is 2^127-1.

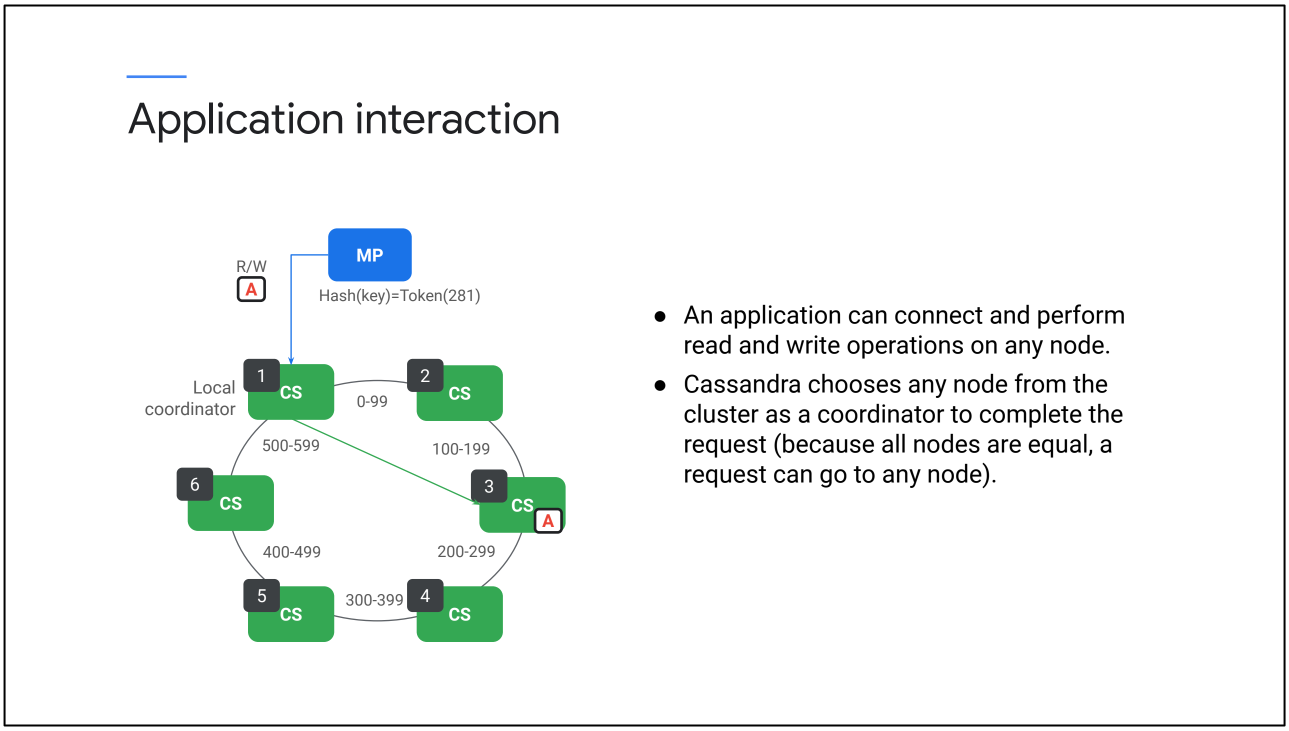


```
> ./nodetool ring

Datacenter: dc-1
==========
Replicas: 3

Address         Rack        Status State   Load            Owns                Token
                                                                               0
172.31.28.255   ra-1        Up     Normal  203.02 KB       100.00%             56713727820156410577229101238628035242
172.31.28.254   ra-1        Up     Normal  222.88 KB       100.00%             113427455640312821154458202477256070485
172.31.29.0     ra-1        Up     Normal  182.91 KB       100.00%             0

Datacenter: dc-2
==========
Replicas: 3

Address         Rack        Status State   Load            Owns                Token
                                                                               56713727820156410577229101238628035342
172.31.18.107   ra-1        Up     Normal  178.91 KB       100.00%             100
172.31.18.109   ra-1        Up     Normal  193.56 KB       100.00%             113427455640312821154458202477256070585
172.31.18.108   ra-1        Up     Normal  180.42 KB       100.00%             56713727820156410577229101238628035342
```

Cassandra distributes data based on tokens. A token is the hashed value of the primary key.

Each Cassandra node is assigned a token range. When you add data to Cassandra, it calculates the token and uses that to figure out on which node to store the new data.

The algorithms that calculate the token are designed in such a way that data is distributed evenly between the nodes.


MP
R/W
A
Hash(key)=Token(281)
Local coordinator
1
CS
2
CS
0-99
500-599
100-199
6
CS
3
CS
A
400-499
200-299
5
CS
300-399
4
CS

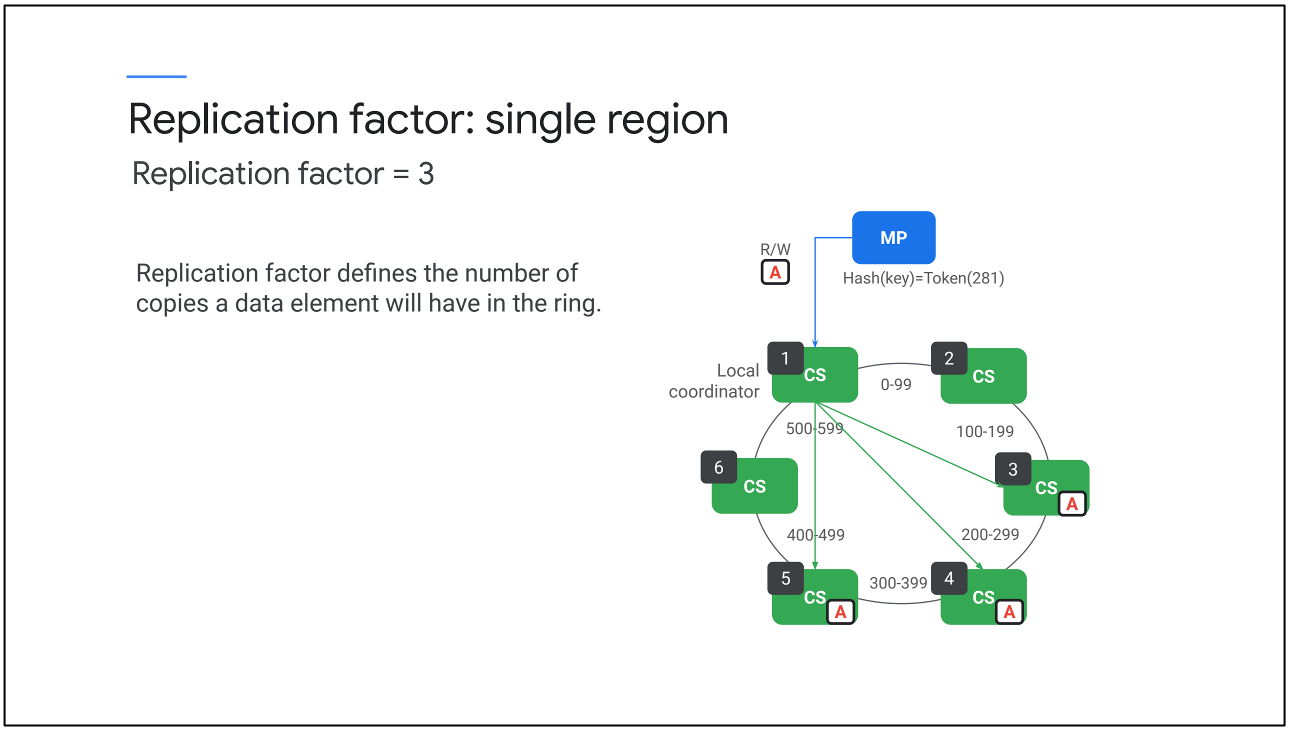
Application interaction

- An application can connect and perform read and write operations on any node.
- Cassandra chooses any node from the cluster as a coordinator to complete the request (because all nodes are equal, a request can go to any node).

The node that is chosen from the cluster to receive a particular read or write request is called a coordinator.

Any node can coordinate any request and each client request may be coordinated by a different node.

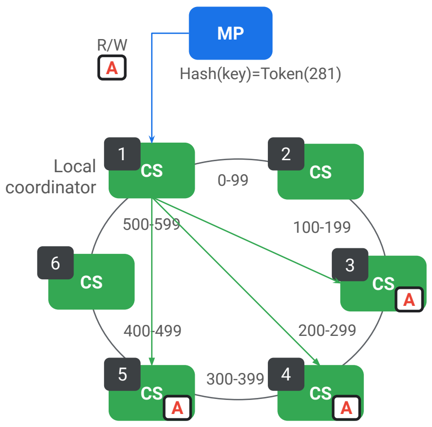
The connection strategy is determined by the database drivers when the application passes the request to the Cassandra cluster (round robin is the most commonly used).



Replication factor: single region

Replication factor = 3

Replication factor defines the number of copies a data element will have in the ring.



Cassandra stores data replicas on multiple nodes to ensure reliability and fault tolerance.

The total number of replicas for a keyspace across a Cassandra cluster is referred to as the *replication factor.*

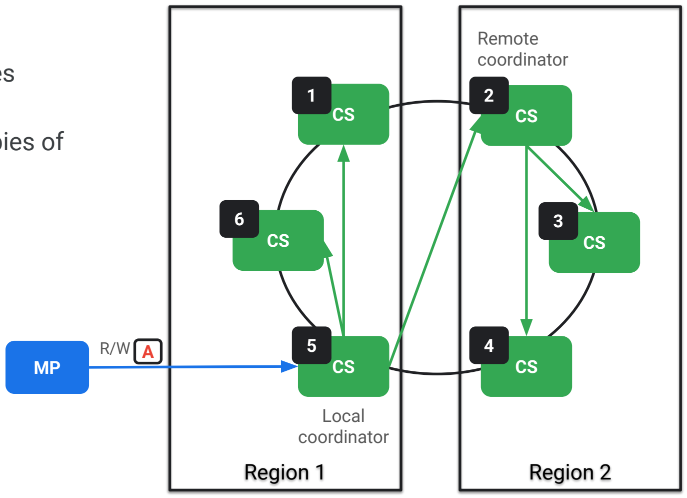
The replication factor and replication strategy is defined at the keyspace level.

Apigee Edge uses a replication factor of three for most keyspaces.

Replication factor: multi-region

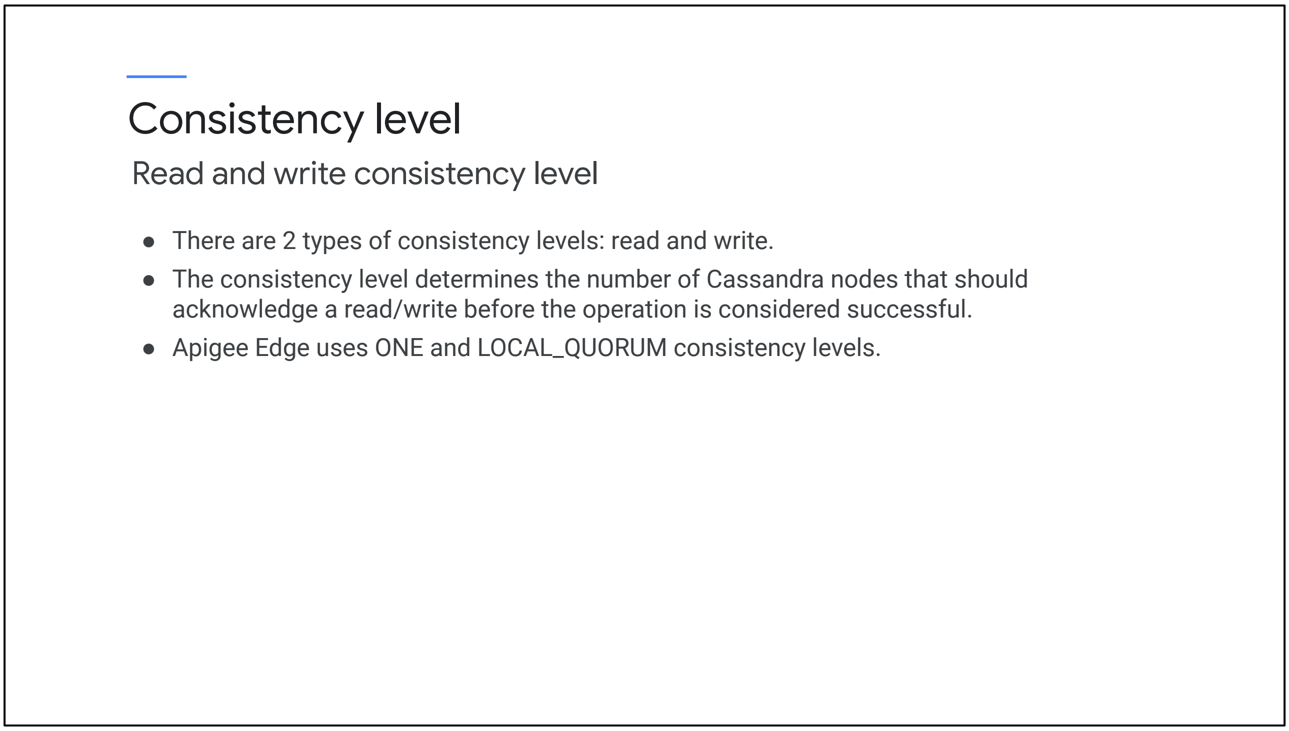
Replication factor = 3

- Replication factor applies to nodes across all DCs (regions).
- Replication factor 3 creates 3 copies of the data per region.



Note that the replication factor is applied per data center. This means that you will have six copies of data if your setup has two regions each with one data center, and a replication factor of 3.

To write to a remote data center, the coordinator node in the local data center chooses a coordinator in the remote data center and makes the request to that node.



Consistency level

Read and write consistency level

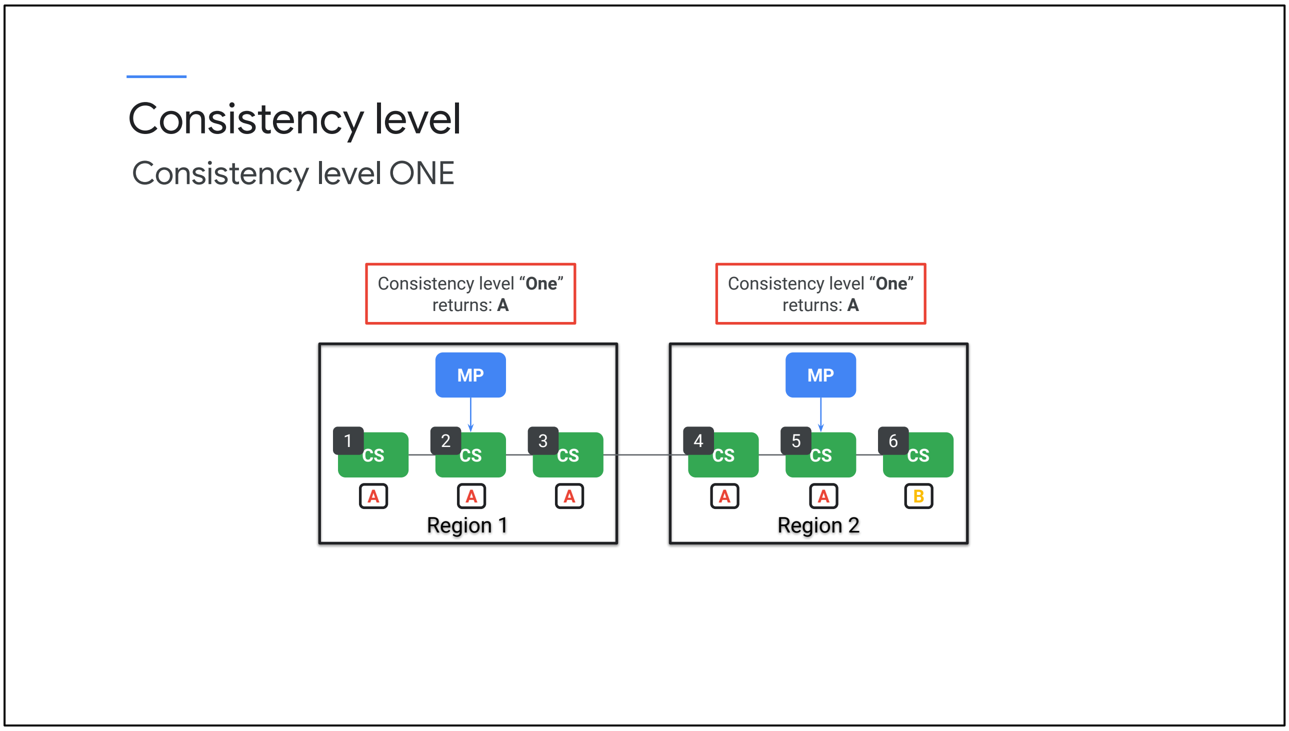
- There are 2 types of consistency levels: read and write.
- The consistency level determines the number of Cassandra nodes that should acknowledge a read/write before the operation is considered successful.
- Apigee Edge uses ONE and LOCAL_QUORUM consistency levels.

As a write request is received by a coordinator node, an asynchronous call is made to the nodes that are responsible for the primary token range and replicas.

The column values are stored with a timestamp, which Cassandra later uses to determine the latest replica.

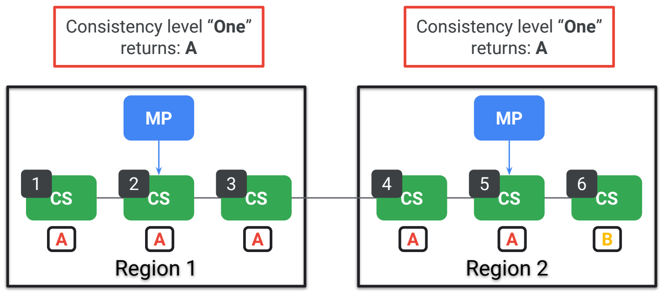
Consistency levels include ONE, LOCAL_QUORUM, and QUORUM, among others.

Apigee Edge uses ONE and LOCAL_QUORUM consistency levels.



Consistency level

Consistency level ONE


Consistency level "One" returns: A
Consistency level "One" returns: A
MP
MP
1 CS
2 CS
3 CS
4 CS
5 CS
6 CS
A
A
A
A
A
B
Region 1
Region 2

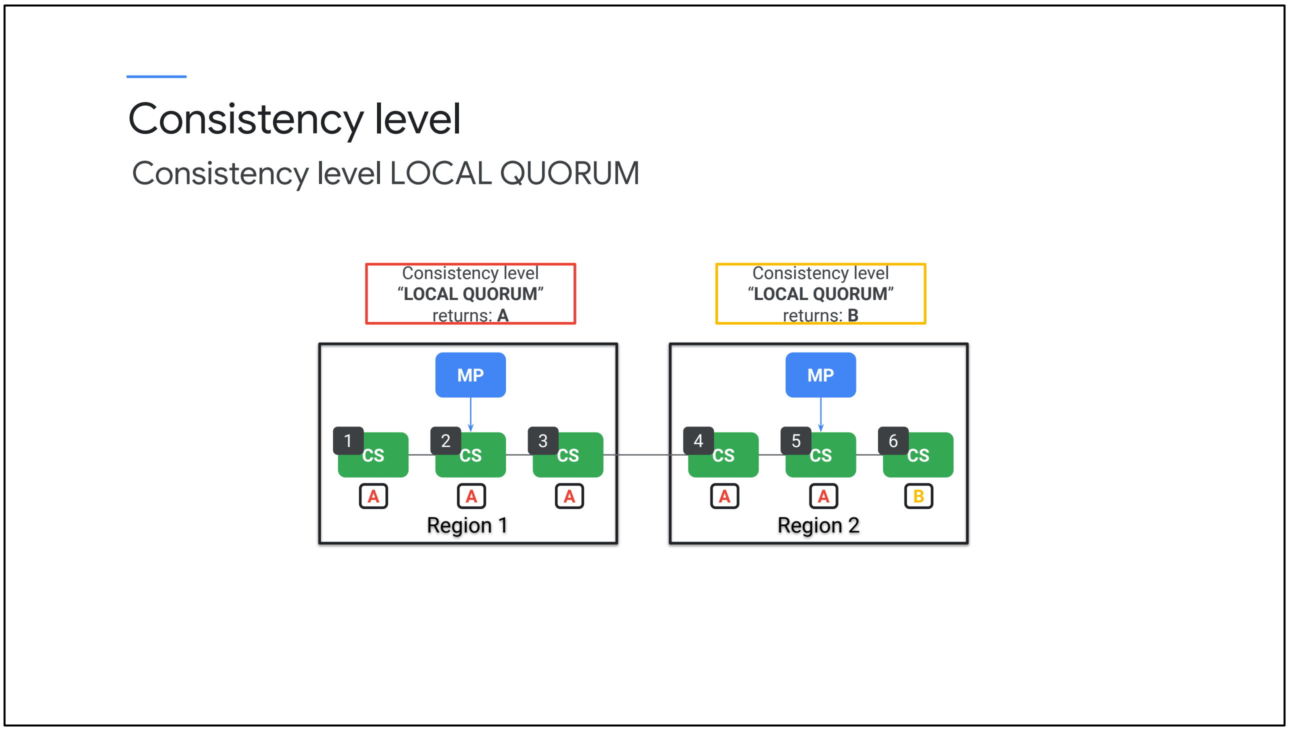
The Cassandra consistency level is defined as the minimum number of Cassandra nodes that must acknowledge a read or write operation before the operation can be considered successful.

Due to the eventually consistent data model of Cassandra, it is possible to have a key with an original value of "A" and an updated value of "B" at the same time on different nodes in the Cassandra ring.

When a read request with a consistency level of ONE is received indicating that at least one node should respond, you may not read the latest updated value of the data stored.

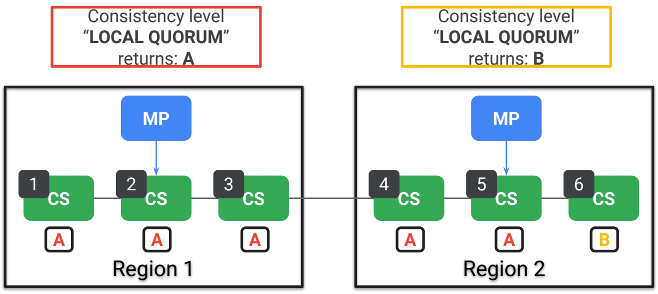
In the example, node 2 in Region 1 or node 5 in region 2 may reply to the client with the value "A", despite the fact that node 6 in region 2 has newer data.

Because the consistency level is ONE, nodes 2 and 5 will not seek confirmation from the other nodes in their respective regions.



Consistency level

Consistency level LOCAL QUORUM

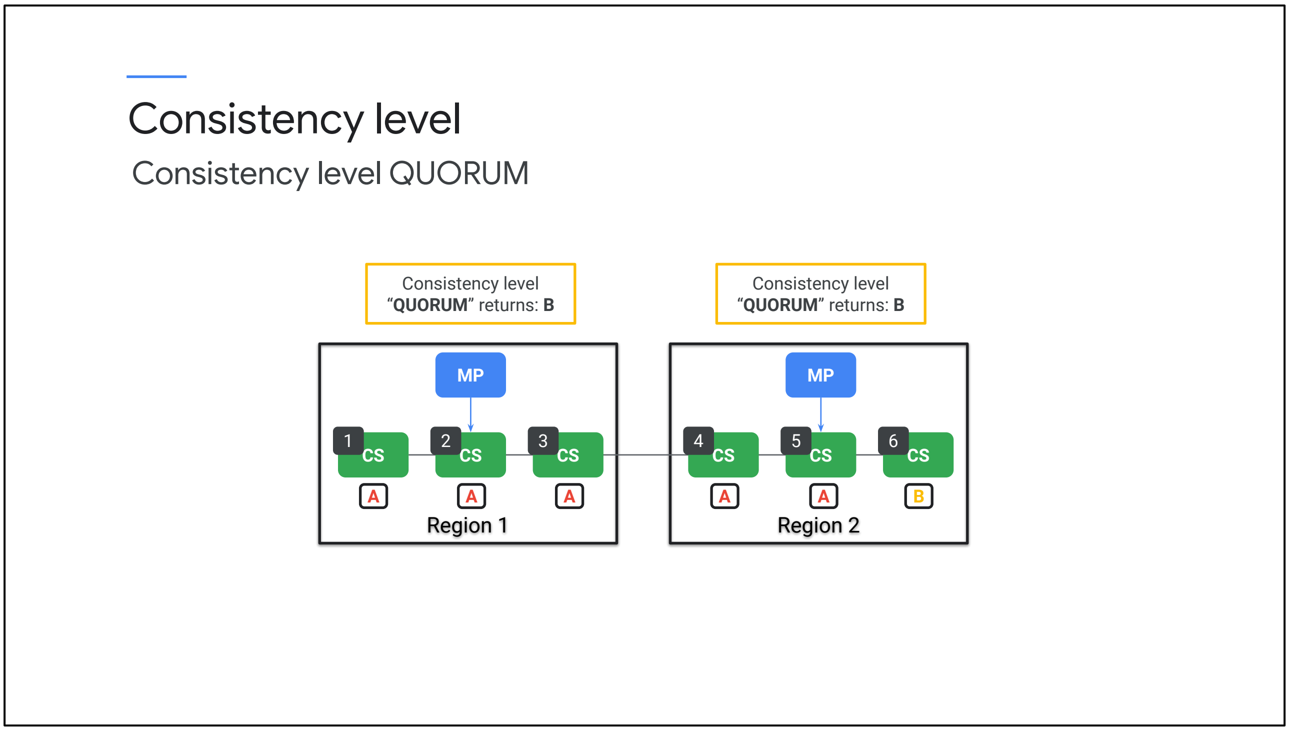


In the next example we have a read request with a consistency level of LOCAL_QUORUM.

This means that the node which receives the request, must receive confirmation from the majority of the nodes in its region that the data it provides is the newest data.

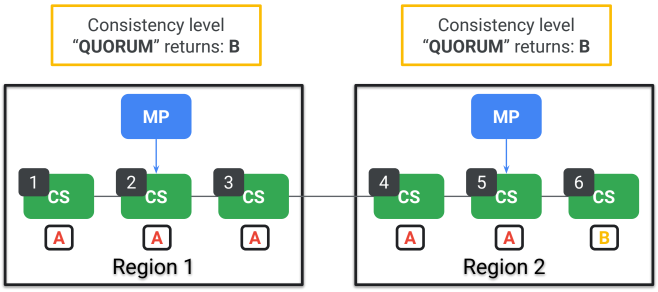
If a different node in the region has newer data, this data will be relayed to the client through the node that received the request.

A consistency level of LOCAL_QUORUM ensures you will get the newest data stored in the region. This is the highest consistency level used by Apigee Edge.



Consistency level

Consistency level QUORUM


Consistency level "QUORUM" returns: B
Consistency level "QUORUM" returns: B
MP
MP
1 CS
2 CS
3 CS
4 CS
5 CS
6 CS
A
A
A
A
A
B
Region 1
Region 2

If you want to ensure data consistency from the Cassandra ring you could configure a consistency level of QUORUM for client read and write operations.

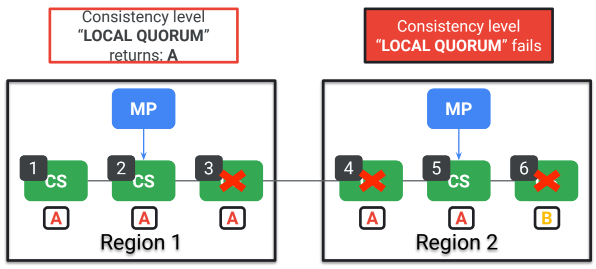
A consistency level of QUORUM indicates that more than half of all the cassandra nodes in the ring should reply to the query.

This means that you could get the newest data in the ring even if it is not stored in the region which received the read request.

Note that this is a very resource intensive consistency level especially if you have a large number of nodes in the ring.

Note that Apigee does not use this consistency level.

Cassandra availability



Cassandra availability is based on the number of available replicas given the consistency level of the database operation being executed.

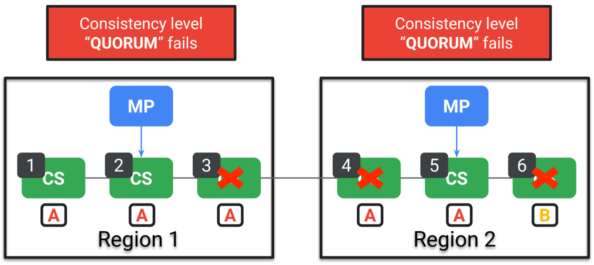
- Local Quorum = replication_factor / 2 + 1
- Local Quorum = 3/2 + 1 = 2 required

Cassandra availability is based on the number of data replicas and the required consistency level of the database operation.

Apigee uses LOCAL_QUORUM as the highest consistency level because Apigee regions are designed to be self-sufficient.

When consistency level is LOCAL_QUORUM, more than 50% of the data replicas in one data center should be available for the transaction to succeed.

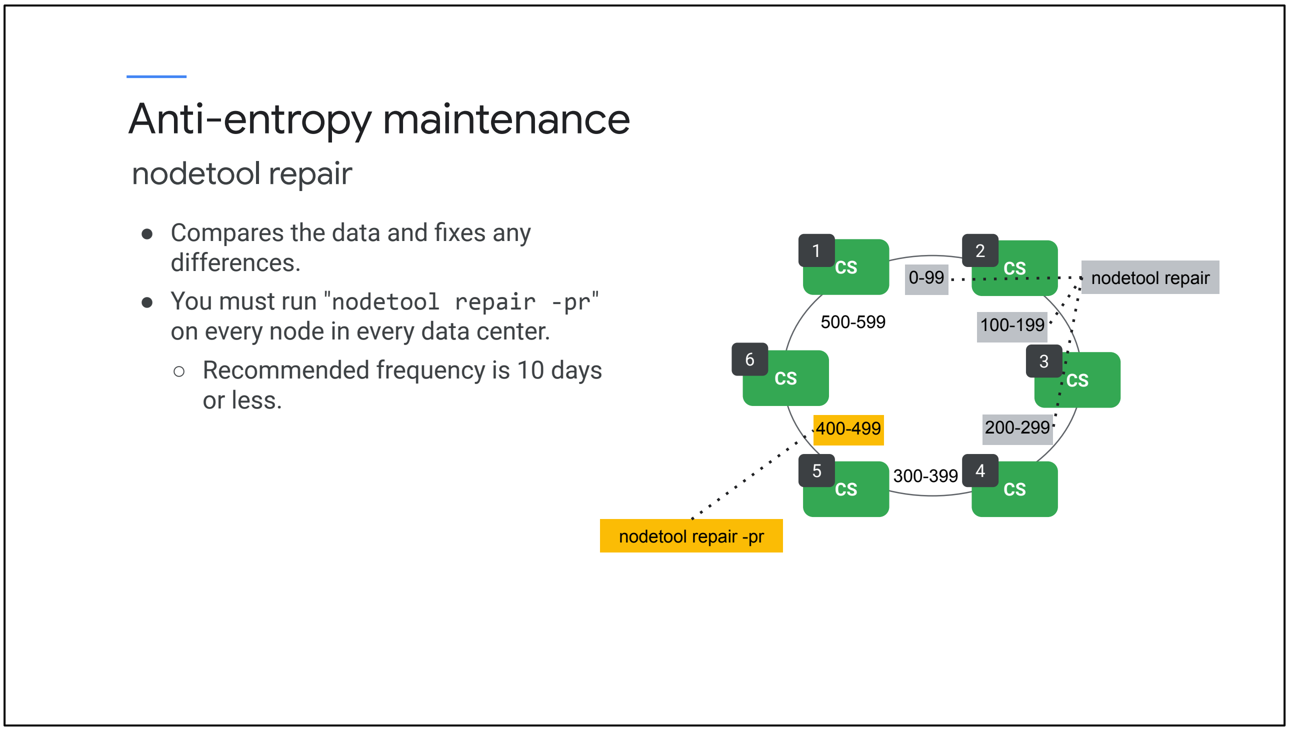
Cassandra availability


Consistency level "QUORUM" fails
Consistency level "QUORUM" fails
MP
MP
1 CS
2 CS
3
4
5 CS
6
A
A
A
A
A
B
Region 1
Region 2

Cassandra availability is based on the number of available replicas given the consistency level of the query operation being executed.

- Quorum = sum_of_all_replication_factors / 2 + 1
- Quorum = (3+3)/2 + 1 = 4 required

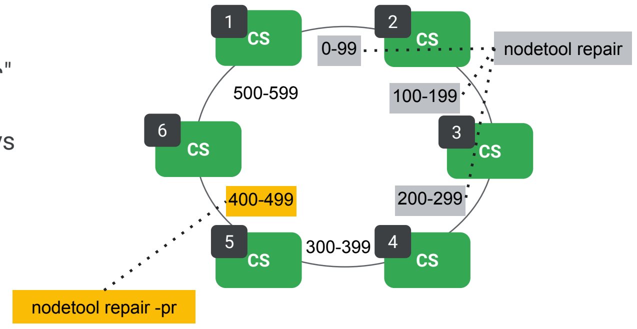
If the consistency level used is QUORUM, then more than 50% of the data replicas in the ring should be available for the transaction to succeed.



Anti-entropy maintenance

nodetool repair

- Compares the data and fixes any differences.
- You must run "`nodetool repair -pr`" on every node in every data center.
    - Recommended frequency is 10 days or less.



The nodetool utility is a command line interface that helps you manage the Cassandra cluster.

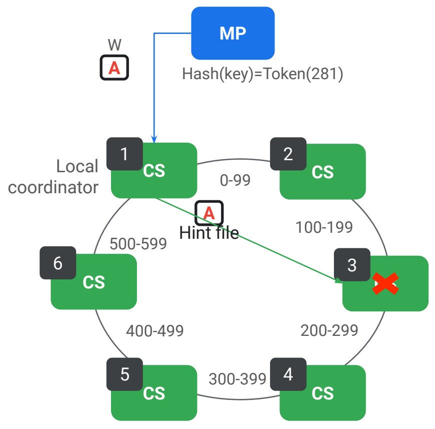
The “nodetool repair” command repairs one or more nodes in a cluster.

The utility splits the token range into subranges based on groups of nodes which share data, then compares and fixes the data in these ranges.

When used with the **pr** option, nodetool repairs the data which is owned by the node on which the command is run.

Hint files

Hint files can be created in case nodes become unresponsive, or a write failure occurs.

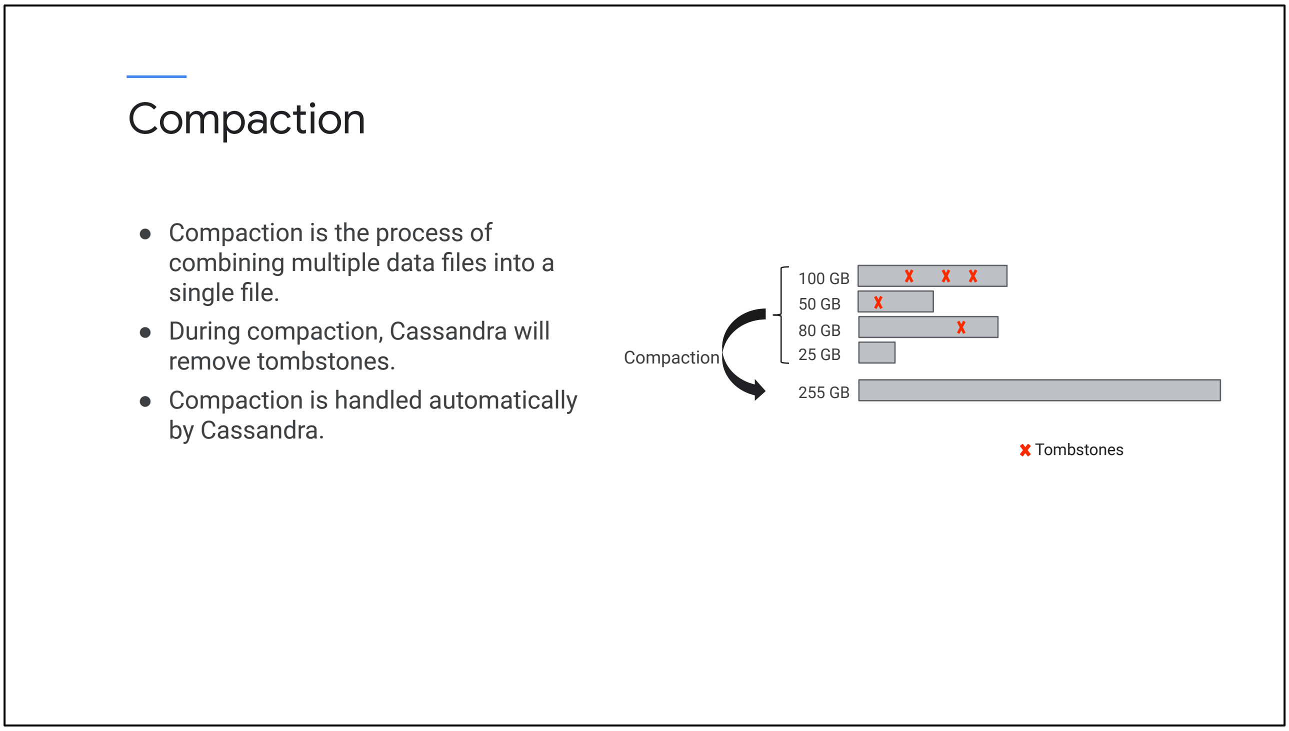


Hinting is a data repair technique that is used when replica nodes are unavailable or a write failure occurs.

After the node is marked as unavailable, missed writes are stored by the coordinator in hint files for a period of time.

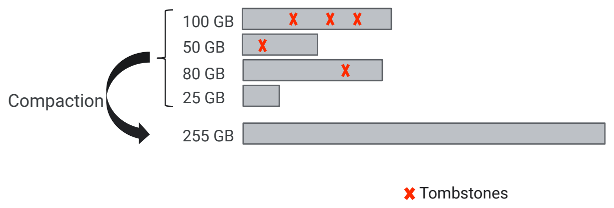
Coordinators replay hints quickly after the unavailable nodes return to the ring.

This process of applying hints is known as hinted handoff and can be enabled in the cassandra.yaml file.



Compaction

- Compaction is the process of combining multiple data files into a single file.
- During compaction, Cassandra will remove tombstones.
- Compaction is handled automatically by Cassandra.

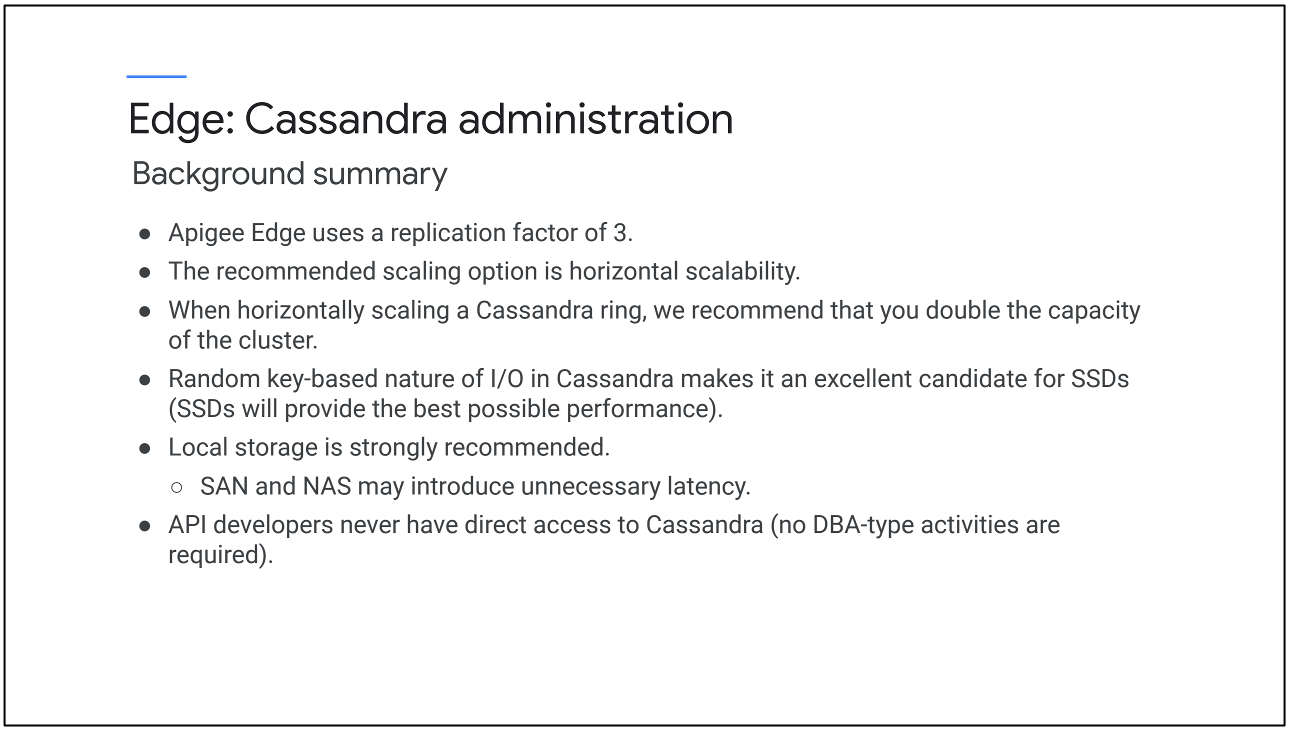


Compaction is the process of combining multiple Cassandra data files into a single file.

During compaction, old files are not deleted until all the data has being moved to new consolidated files.

Usually compaction is an automated process and is triggered based on the number of files or the size of the files.

Tombstones represent data that is logically marked as deleted in the old files, and are removed during compaction.



Edge: Cassandra administration

Background summary

- Apigee Edge uses a replication factor of 3.
- The recommended scaling option is horizontal scalability.
- When horizontally scaling a Cassandra ring, we recommend that you double the capacity of the cluster.
- Random key-based nature of I/O in Cassandra makes it an excellent candidate for SSDs (SSDs will provide the best possible performance).
- Local storage is strongly recommended.
  - SAN and NAS may introduce unnecessary latency.
- API developers never have direct access to Cassandra (no DBA-type activities are required).

While administering your Cassandra installation for Apigee private cloud, consider the following:

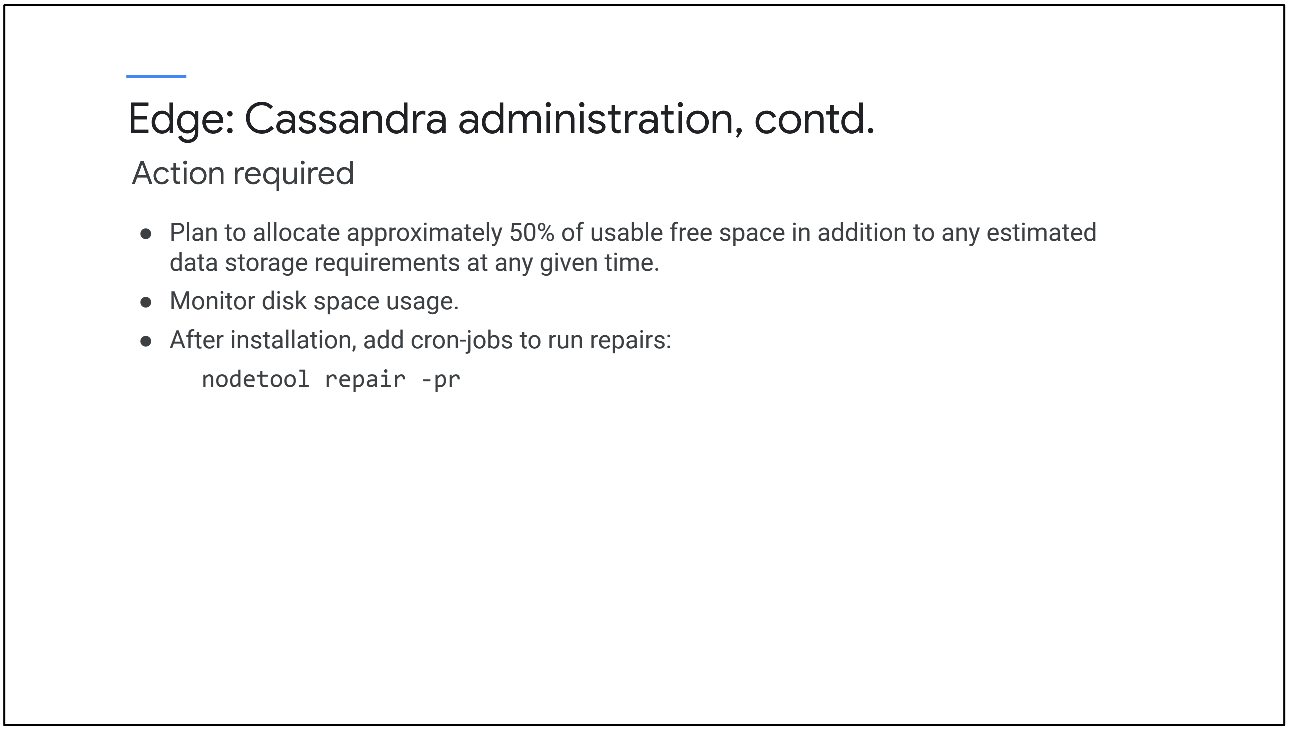
Apigee Edge uses a replication factor of 3, with horizontal scalability as the recommended scaling option.

It is recommended to double the capacity of the cluster when horizontally scaling a Cassandra ring.

The random key based nature of I/O in Cassandra makes solid-state disks an excellent choice as the storage medium.

Using local SSDs will give you the best possible performance.

Also, API Developers never have direct access to the Cassandra database.



Edge: Cassandra administration, contd.

Action required

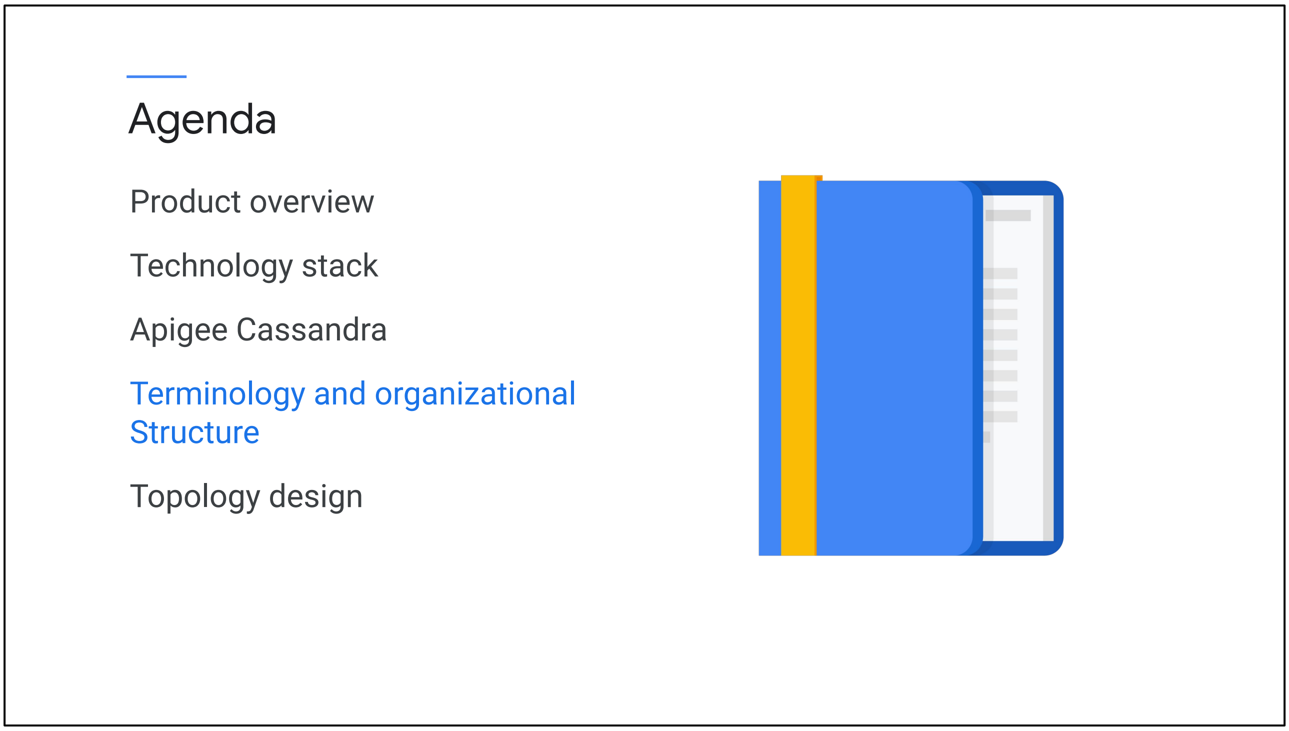
- Plan to allocate approximately 50% of usable free space in addition to any estimated data storage requirements at any given time.
- Monitor disk space usage.
- After installation, add cron-jobs to run repairs:

```
nodetool repair -pr
```

You should plan to allocate approximately 50% of usable free space in addition to any estimated data storage requirements at any given time.

It is a good practice to monitor disk space usage by considering the required and available storage space.

After the installation is completed, add cron-jobs to periodically run repairs using the nodetool utility.



Agenda

Product overview

Technology stack

Apigee Cassandra

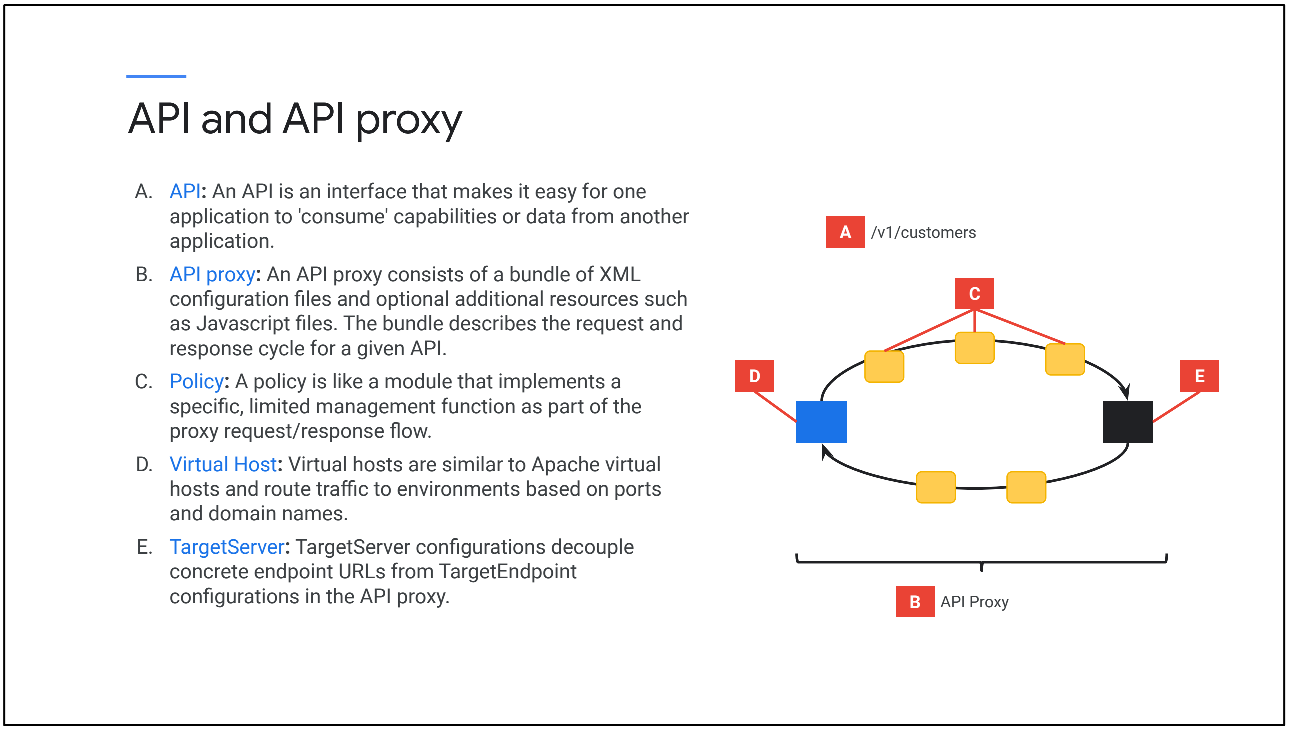
Terminology and organizational Structure

Topology design

Apigee uses specific terms to describe key capabilities and concepts.

In this lesson, we focus on building an understanding of these concepts.

As we progress through this lesson, keep in mind the responsibilities of components discussed in the technology stack lesson.



API and API proxy

A. API: An API is an interface that makes it easy for one application to 'consume' capabilities or data from another application.
B. API proxy: An API proxy consists of a bundle of XML configuration files and optional additional resources such as Javascript files. The bundle describes the request and response cycle for a given API.
C. Policy: A policy is like a module that implements a specific, limited management function as part of the proxy request/response flow.
D. Virtual Host: Virtual hosts are similar to Apache virtual hosts and route traffic to environments based on ports and domain names.
E. TargetServer: TargetServer configurations decouple concrete endpoint URLs from TargetEndpoint configurations in the API proxy.

An API proxy allows you to implement the request and response flow for a given API.

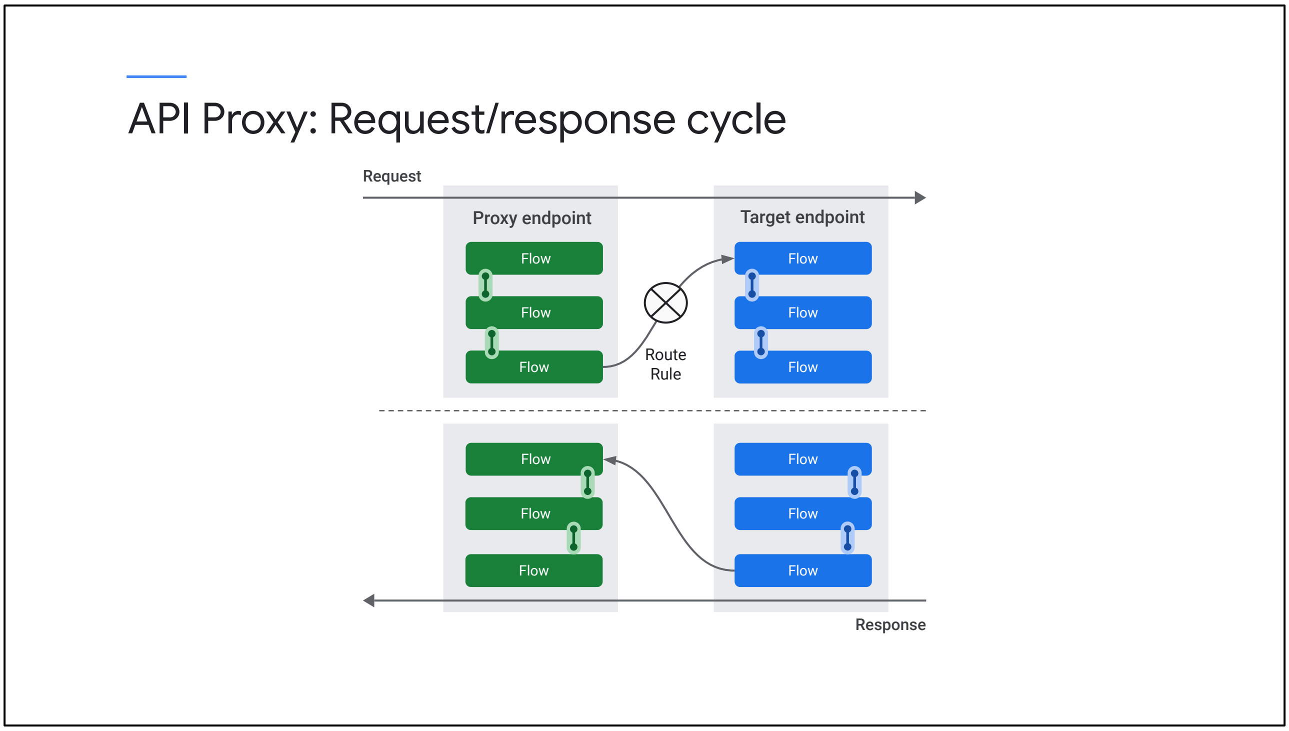
Policies are attached to a flow. A policy allows you to intercept the flow and inject specific functionality. Apigee Edge offers many out of the box and extension policies, these can be used in the request and response flows.

Conceptually, the entry point for API calls to an API proxy, is represented by a virtual host. Virtual hosts are exposed on the router, and allow you to direct requests to specific API proxies.

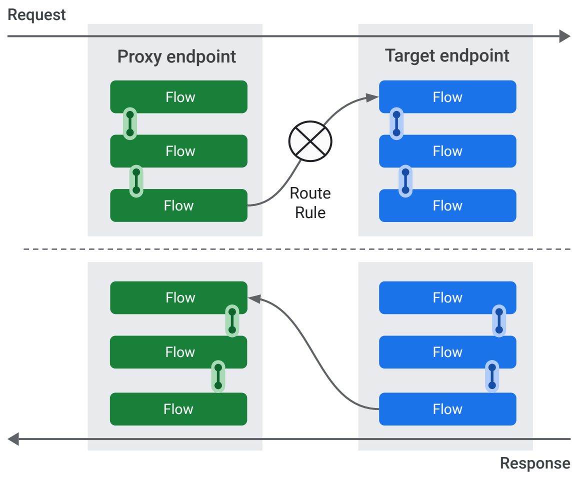
Often, an API may call a single backend system. Apigee Edge allows you to abstract backend system URLs, making it easy to move API proxies between environments without needing to modify code.

TargetServers are configuration objects that are used for this purpose.

A TargetServer allows you to define a backend resource, and associate this definition to a logical name that is then referenced by API proxies.



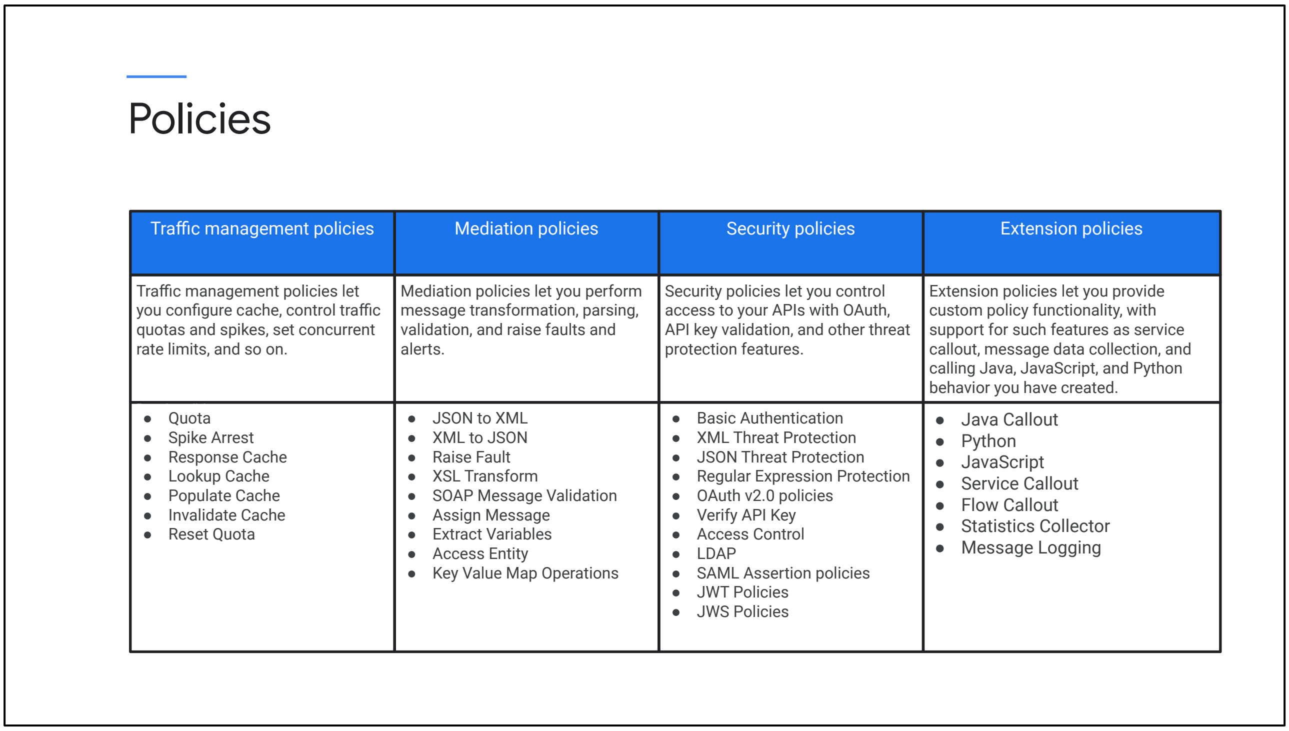
API Proxy: Request/response cycle



API proxies are comprised of a proxy endpoint and target endpoint.

Each endpoint consists of a pre-flow, conditional flows and a post-flow.

A flow consists of policies that apply your API proxy execution logic to requests to the API and to responses from the target backend.



Policies

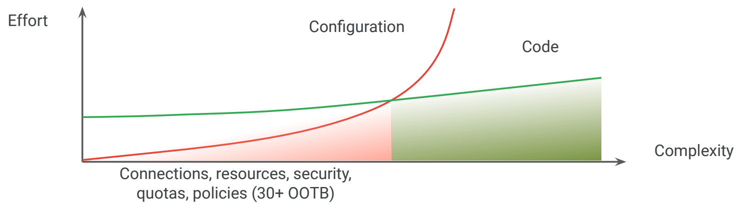
Traffic management policies	Mediation policies	Security policies	Extension policies
Traffic management policies let you configure cache, control traffic quotas and spikes, set concurrent rate limits, and so on.	Mediation policies let you perform message transformation, parsing, validation, and raise faults and alerts.	Security policies let you control access to your APIs with OAuth, API key validation, and other threat protection features.	Extension policies let you provide custom policy functionality, with support for such features as service callout, message data collection, and calling Java, JavaScript, and Python behavior you have created.
• Quota • Spike Arrest • Response Cache • Lookup Cache • Populate Cache • Invalidate Cache • Reset Quota	• JSON to XML • XML to JSON • Raise Fault • XSL Transform • SOAP Message Validation • Assign Message • Extract Variables • Access Entity • Key Value Map Operations	• Basic Authentication • XML Threat Protection • JSON Threat Protection • Regular Expression Protection • OAuth v2.0 policies • Verify API Key • Access Control • LDAP • SAML Assertion policies • JWT Policies • JWS Policies	• Java Callout • Python • JavaScript • Service Callout • Flow Callout • Statistics Collector • Message Logging

Apigee Edge offers many policies that are used in API proxies. Policies are used by API engineers during API proxy development.

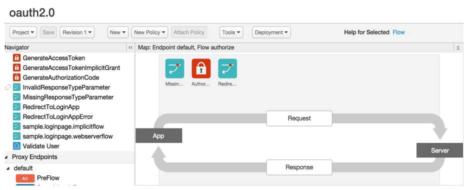
Certain policies such as cache, key value map, spike arrest and message logging may have operational requirements.

The operations team works with the API development team to implement specific capabilities in the proxy based on their requirements.

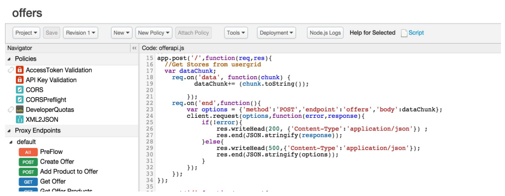
Policies and extensions



Configuration-driven proxy definition



Code-driven proxy definition



Code: offerapi.js

```
app.post('/',function(req,res){
  //Get Stores from usergrid
  var dataChunk;
    req.on('data', function(chunk) {
        dataChunk+= (chunk.toString());

      });
    req.on('end',function(){
      var options = {'method':'POST','endpoint':'offers','body':dataChunk};
      client.request(options,function(error,response){
          if(!error){
              res.writeHead(200, {'Content-Type':'application/json'}) ;
              res.end(JSON.stringify(response));
          }else{
              res.writeHead(500,{'Content-Type':'application/json'});
              res.end(JSON.stringify(options));
          }
      });
    });
});
```

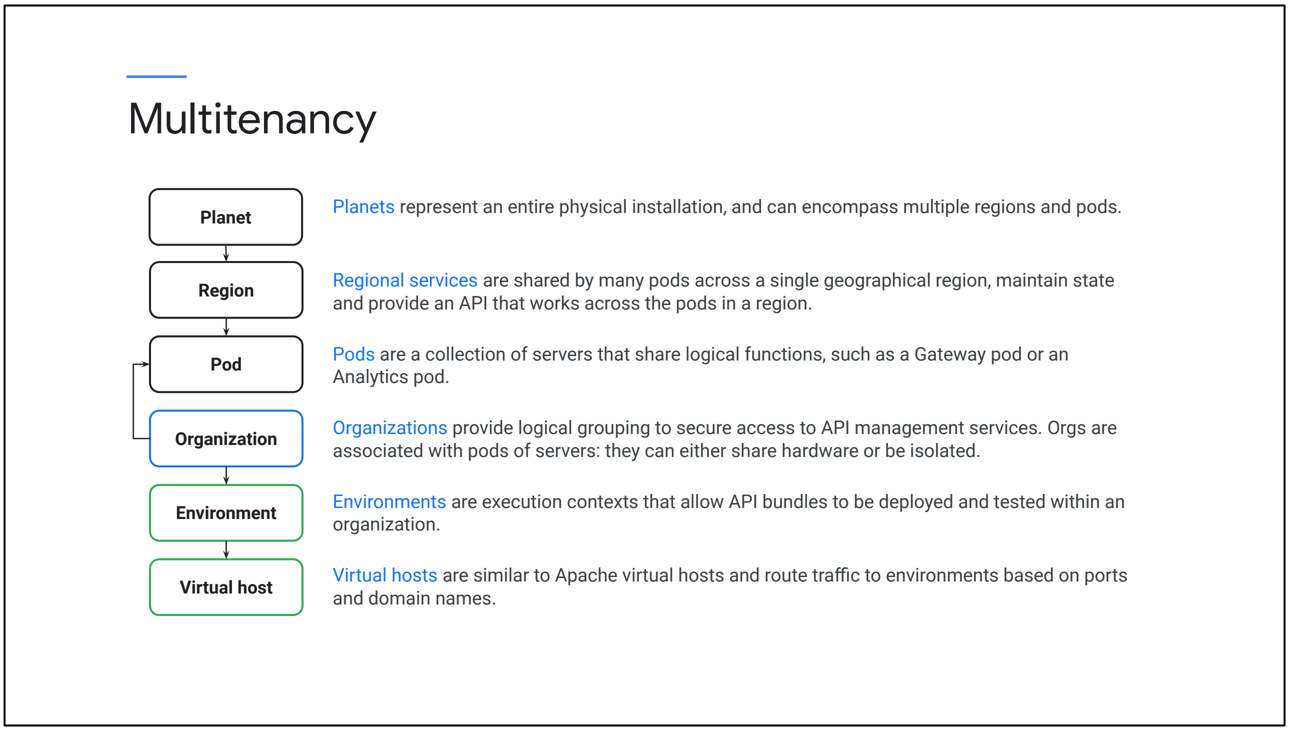
On Apigee Edge, most policies are provided as part of the product and described in XML. Developers configure these policies to modify their behavior.

Even though XML policies allow you to address a wide range of scenarios, sometimes there is a need to implement custom policies.

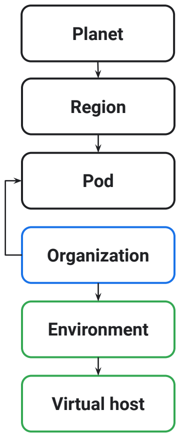
Apigee Edge provides extension policies that allow API engineers to write custom code in JavaScript, Java and Python.

Most API proxies usually contain XML policies with a small subset of extension policies that contain code.

Depending on the complexity of your proxy logic, you may not be able to solely rely on configuration based policies and may have to include extension policies.



Multitenancy



Planets represent an entire physical installation, and can encompass multiple regions and pods.

Regional services are shared by many pods across a single geographical region, maintain state and provide an API that works across the pods in a region.

Pods are a collection of servers that share logical functions, such as a Gateway pod or an Analytics pod.

Organizations provide logical grouping to secure access to API management services. Orgs are associated with pods of servers: they can either share hardware or be isolated.

Environments are execution contexts that allow API bundles to be deployed and tested within an organization.

Virtual hosts are similar to Apache virtual hosts and route traffic to environments based on ports and domain names.

In Apigee terminology, a planet refers to an Apigee Edge private cloud installation and is a collection of resources across one or more regions.

This includes all of the hardware, virtual machines or cloud instances in a given installation.

Planets are subdivided into regions, a region is typically represented by a data center or cloud region.

Pods represent a logical grouping of components by function. We use pods to group components together for configuration and management. Apigee Edge uses three kinds of pods: gateway, central and analytics.

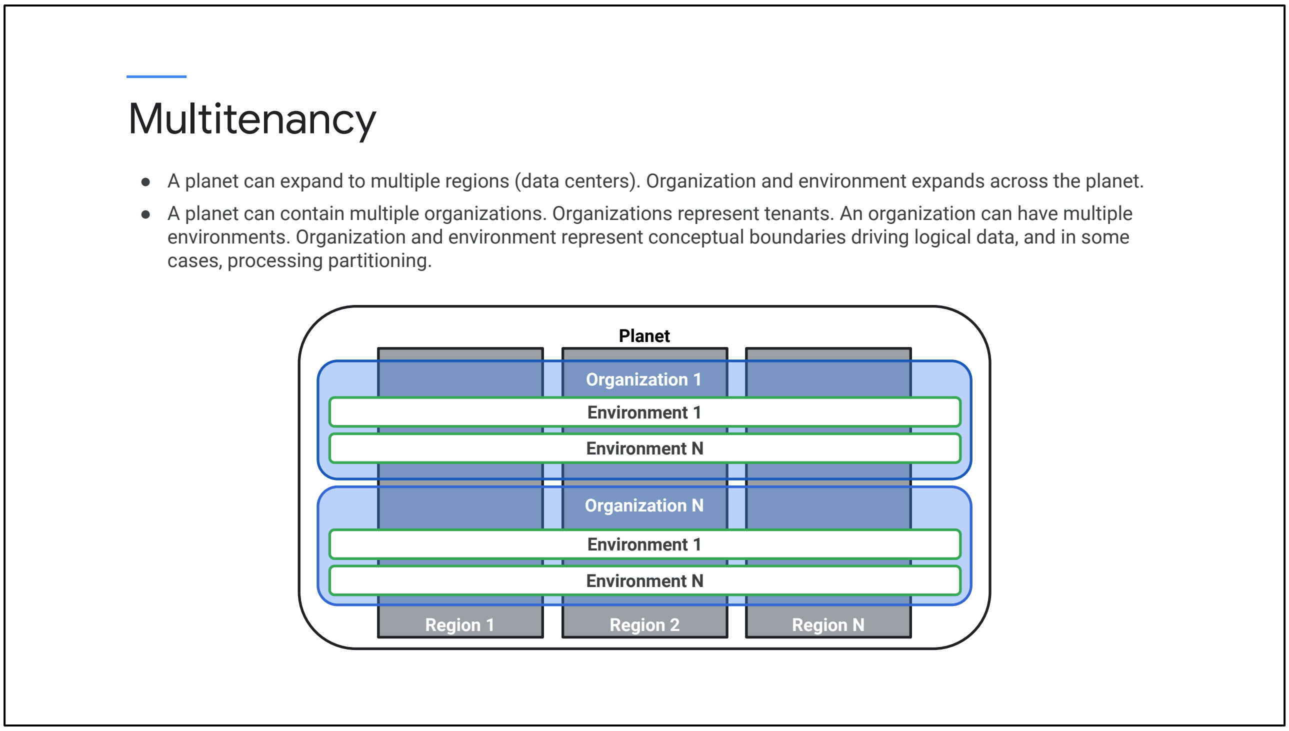
An organization is a logical boundary that enforces logical data partitioning and security, and is key to implementing multi tenancy. An organization is considered to be a tenant in Apigee Edge. This means that you can create one or more organizations on the same physical infrastructure.

Environments are execution contexts within a given organization. Typically, environments are mapped to an API SDLC. Most customers create environments named Dev, QA and so on.

An organization can contain one or more environments.

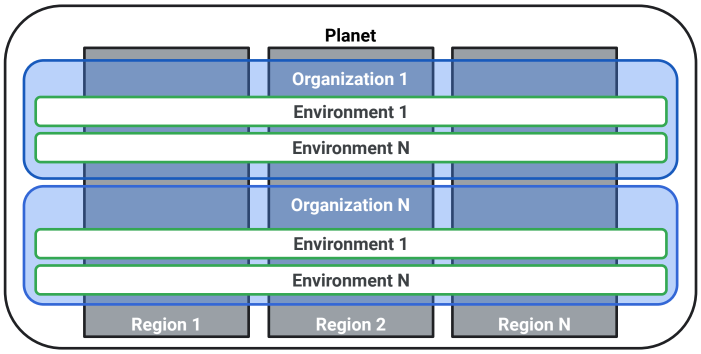
Apigee Edge analytics data is partitioned by organization and environment.

Virtual hosts route traffic to an environment. Each environment can have one or more



Multitenancy

- A planet can expand to multiple regions (data centers). Organization and environment expands across the planet.
- A planet can contain multiple organizations. Organizations represent tenants. An organization can have multiple environments. Organization and environment represent conceptual boundaries driving logical data, and in some cases, processing partitioning.



A common approach is to have separate planets for production and non-production use. A planet can span multiple data centers or regions as shown here.

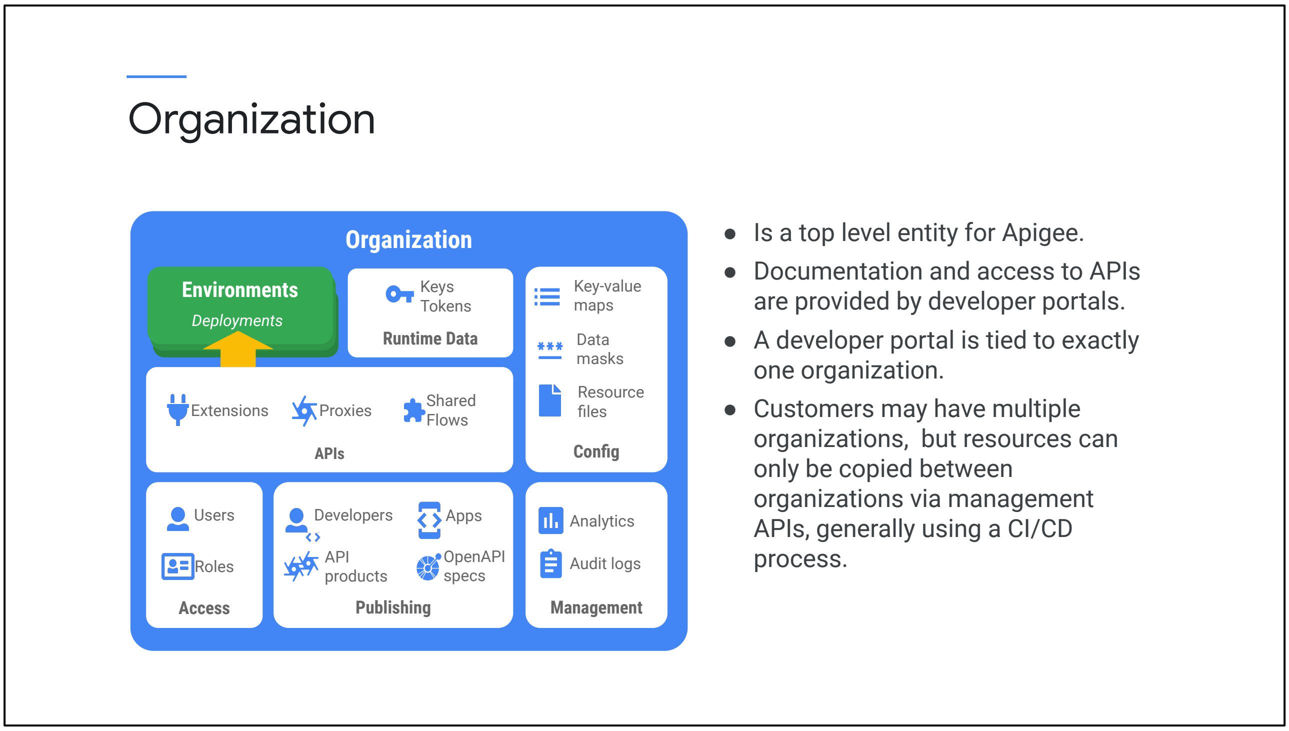
On the diagram, we illustrate an Apigee planet with multiple regions. The planet contains one or more organizations.

Each organization contains one or more environments. Organizations and environments can span the planet in all regions.

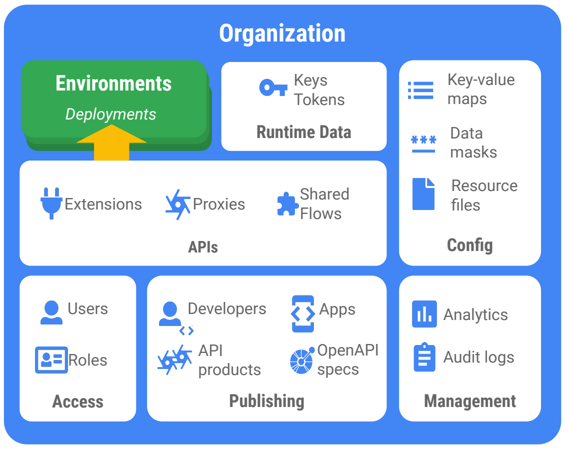
This is an important concept. The scope of organizations and environments allows you to execute actions in a single location and influence the behavior of APIs, the platform, and infrastructure components across multiple regions.

For example, deploying an API to an environment in region three, will result in the API being deployed to the other regions as well.

This allows you to manage a distributed infrastructure from a centralized point.



Organization



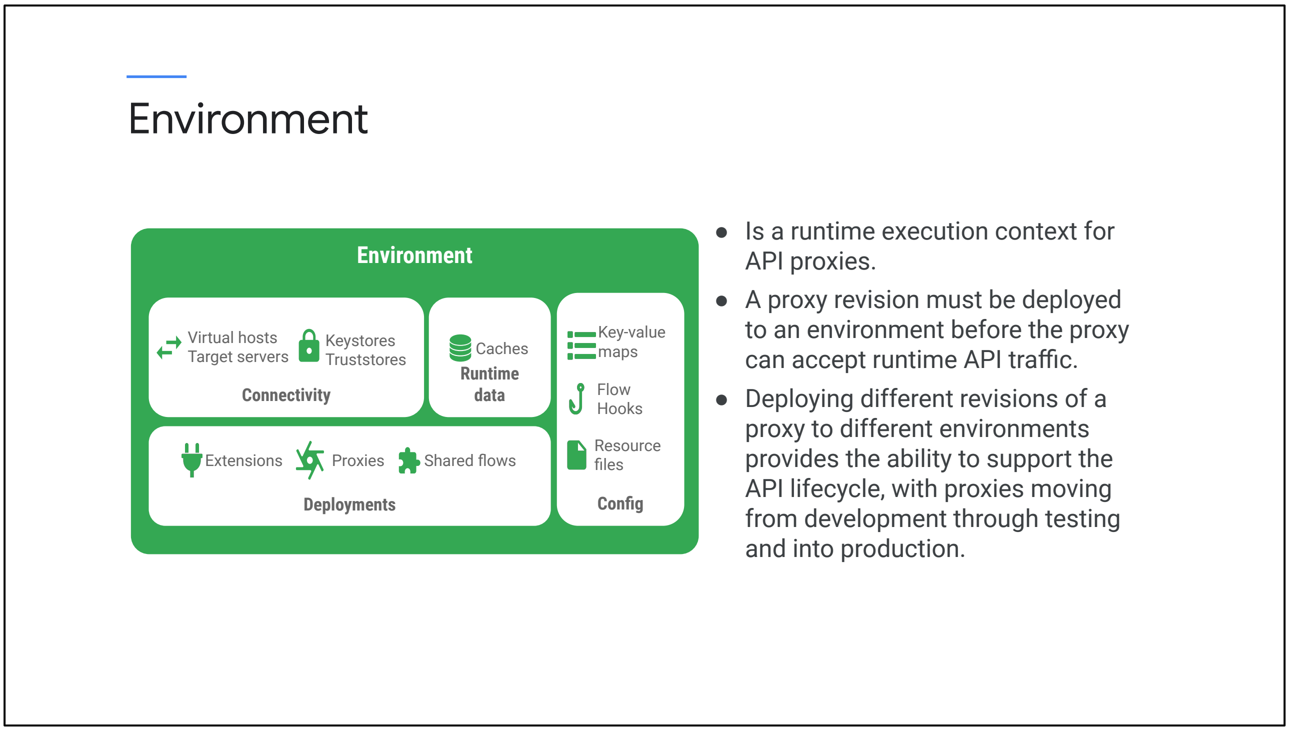
- Is a top level entity for Apigee.
- Documentation and access to APIs are provided by developer portals.
- A developer portal is tied to exactly one organization.
- Customers may have multiple organizations, but resources can only be copied between organizations via management APIs, generally using a CI/CD process.

Organizations and environments represent strong conceptual boundaries which allow for data and process partitioning.

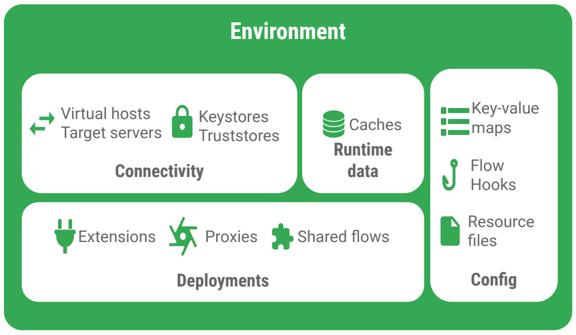
An organization is a top level entity and represents a tenant on the platform.

It also represents scope for API proxies and other related objects, which cannot be shared outside the organization.

Organization scoped objects include API keys, tokens, API proxies, users, API products, developers and applications, analytics and configuration data etc.



Environment



- Is a runtime execution context for API proxies.
- A proxy revision must be deployed to an environment before the proxy can accept runtime API traffic.
- Deploying different revisions of a proxy to different environments provides the ability to support the API lifecycle, with proxies moving from development through testing and into production.

An environment represents a runtime execution context for API proxies.

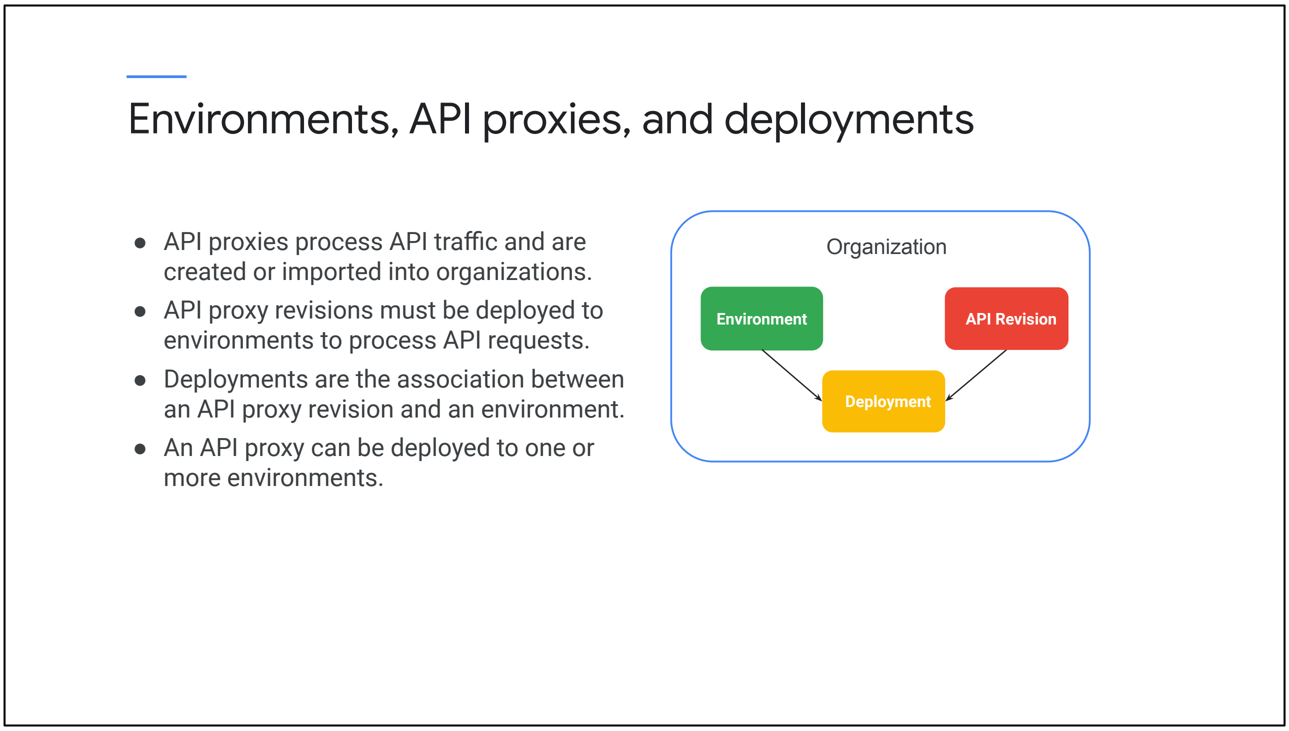
It has access to all organization level resources like API proxy revisions, developers, developer apps, API keys , shared flows and others.

You can also create environment scoped resources which are only available to API proxies deployed to that environment.

You typically create one production organization and one non-production organization, with one or more environments. This can however vary depending on your requirements.

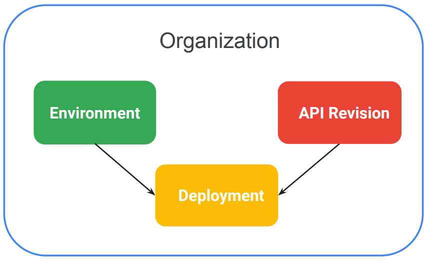
With multiple organizations, API consumers that need to access the same API deployed to environments in each organization will require different API keys.

Multiple environments can serve to implement the software development lifecycle for API proxies.



Environments, API proxies, and deployments

- API proxies process API traffic and are created or imported into organizations.
- API proxy revisions must be deployed to environments to process API requests.
- Deployments are the association between an API proxy revision and an environment.
- An API proxy can be deployed to one or more environments.

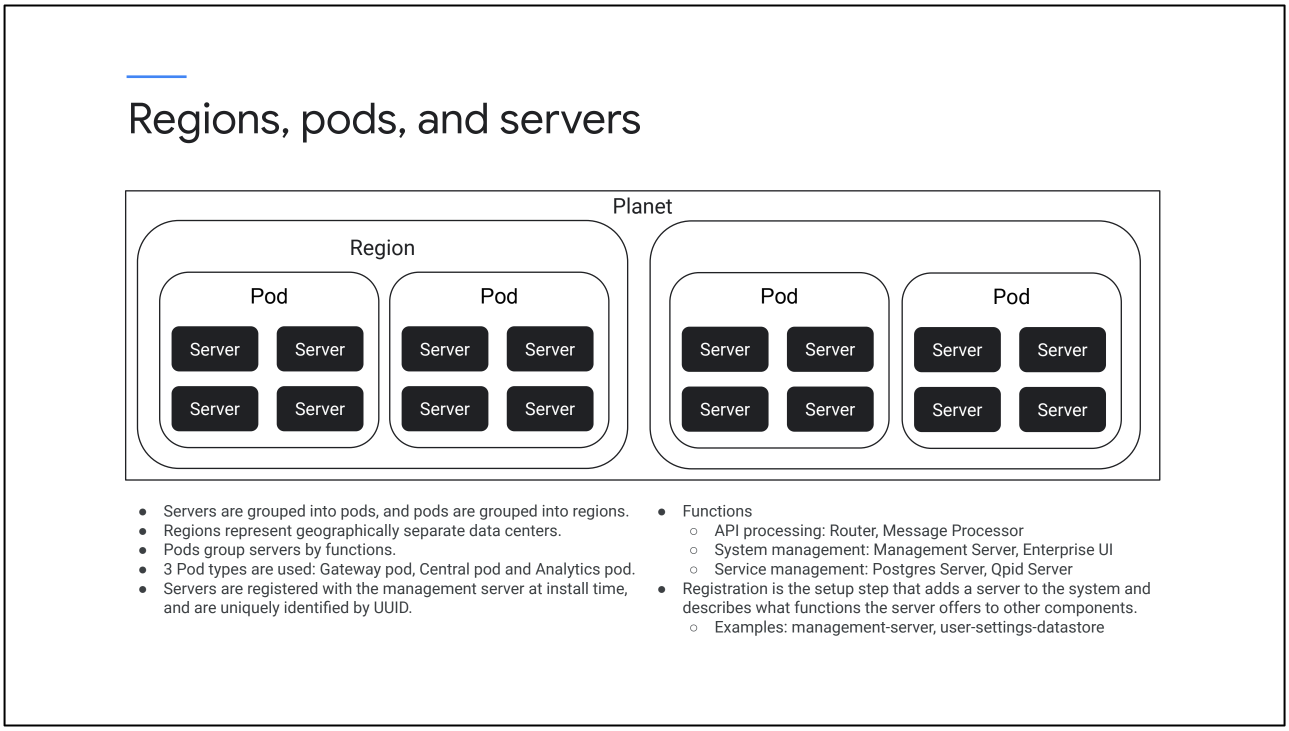


Given the logical nature of environments, an API deployment is no more than a logical association between an API revision and the specified environment.

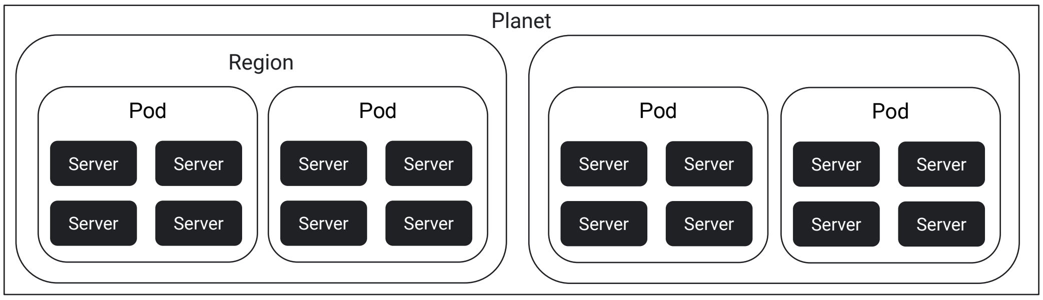
On Apigee Edge, APIs belong to organizations.

Deploying APIs to an environment within the same organization does not require any movement of code artifacts.

We explore API deployment in detail in later modules.



Regions, pods, and servers



- Servers are grouped into pods, and pods are grouped into regions.
- Regions represent geographically separate data centers.
- Pods group servers by functions.
- 3 Pod types are used: Gateway pod, Central pod and Analytics pod.
- Servers are registered with the management server at install time, and are uniquely identified by UUID.
- Functions
  - API processing: Router, Message Processor
  - System management: Management Server, Enterprise UI
  - Service management: Postgres Server, Qpid Server
- Registration is the setup step that adds a server to the system and describes what functions the server offers to other components.
  - Examples: management-server, user-settings-datastore

This diagram illustrates the concepts of regions, pods and servers, and their relationships to each other.

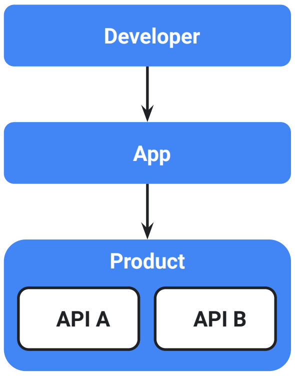
On Apigee Edge, it is possible to add and remove logical and physical capacity from the system.

During installation, each server is registered with its corresponding pod.

Associations between servers and pods are managed using unique identifiers or UUIDs.

During installation, a UUID is generated for each server that is then used to register the server to the pod.

Products, apps, and developers



- Products bundle together API proxies to offer them to developers.
- Developers are external API users who interact with your APIs by developing applications that use them.
- Developers may be grouped into companies.
- Apps are associations between developers and products.

An API product bundles resources, such as API proxies, in order to provide a specific level of access and functionality for client application developers.

And API product typically specifies a list of API proxies along with access limits, the API key approval method, and other configurations that should be in place for all of the bundle proxies.

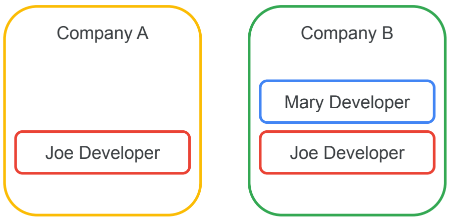
API products are a key part of API management.

Elements such as quota, and other characteristics of the API behavior, are defined by API products.

API products allow delegation of API management to API product owners.

Changes to products can be implemented without the intervention of API engineers or operations teams.

Companies



- A company is a collection of developers managed as a single entity.
- A developer is a single entity, uniquely identified by an email address.
- Developers maintain profile information, including a list of companies that they belong to. Developers can belong to more than one company.
- Companies were originally tied to monetized orgs for the purpose of managing billing and quota at the company level. You can now also use them for non-monetized orgs to group developers, apps, and keys.

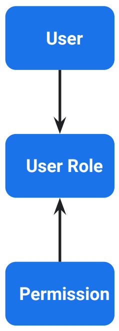
A developer is a single entity, uniquely identified by an email address.

A company is a collection of developers managed as a single entity. A developer can belong to more than one company.

You can optionally group multiple developers into companies based on business unit, product line, division, or other corporate entity.

Users and roles

- Users are accounts with some level of access on Edge and are:
  - Uniquely identified by email address
  - Used to log in to the user interface and issue calls to the management API
- Roles connect users to permissions in the system.
- Both users and roles can be created at the global or organizational levels:
  - Global users can be members of global and organizational roles.
  - Organizational users can only be members of organizational roles.



https://docs.apigee.com/private-cloud/v4.50.00/managing-users-roles-and-permissions

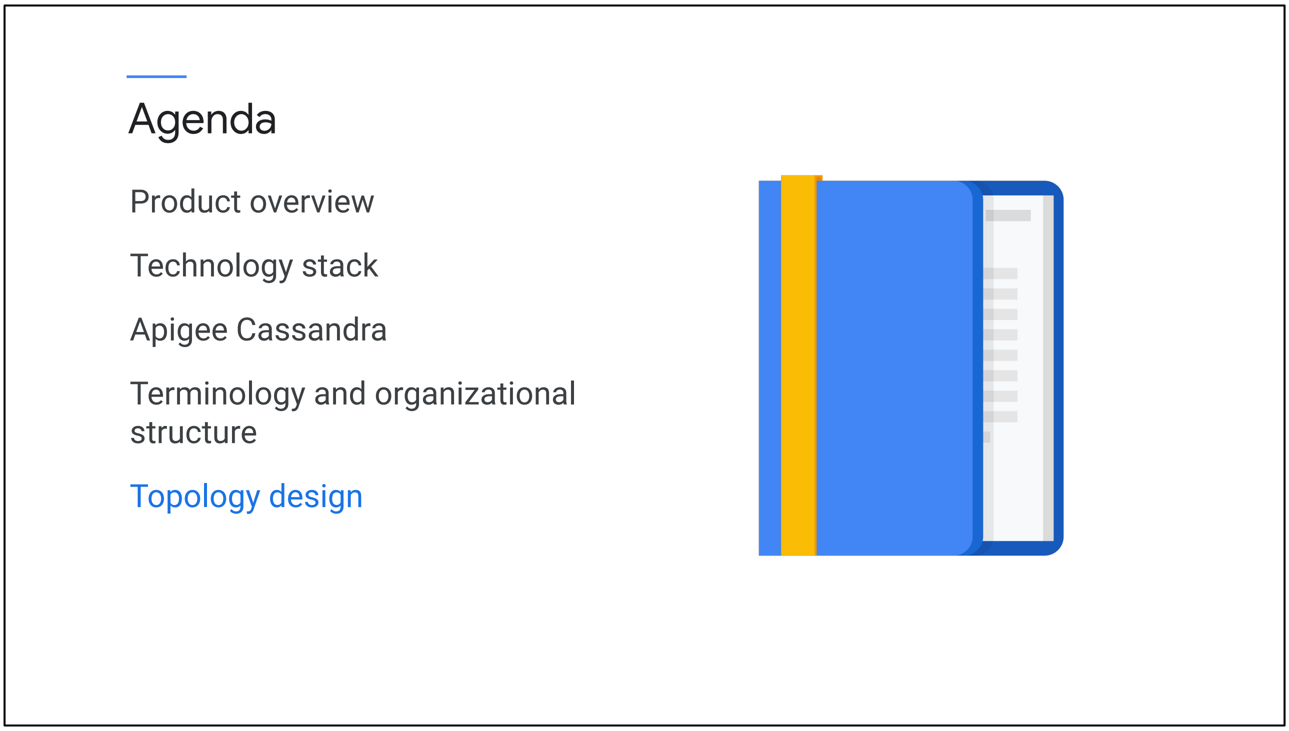
Apigee Edge for private cloud provides fine grained role based access control.

Users, roles and permissions are created and stored in Openldap.

Management of users and roles can be done through the Apigee UI and management API.

During installation, a collection of default roles is created. You can also create custom roles, tailoring them to your specific security and management requirements.

For more information on this topic, refer to the Apigee private cloud documentation.



Agenda

Product overview

Technology stack

Apigee Cassandra

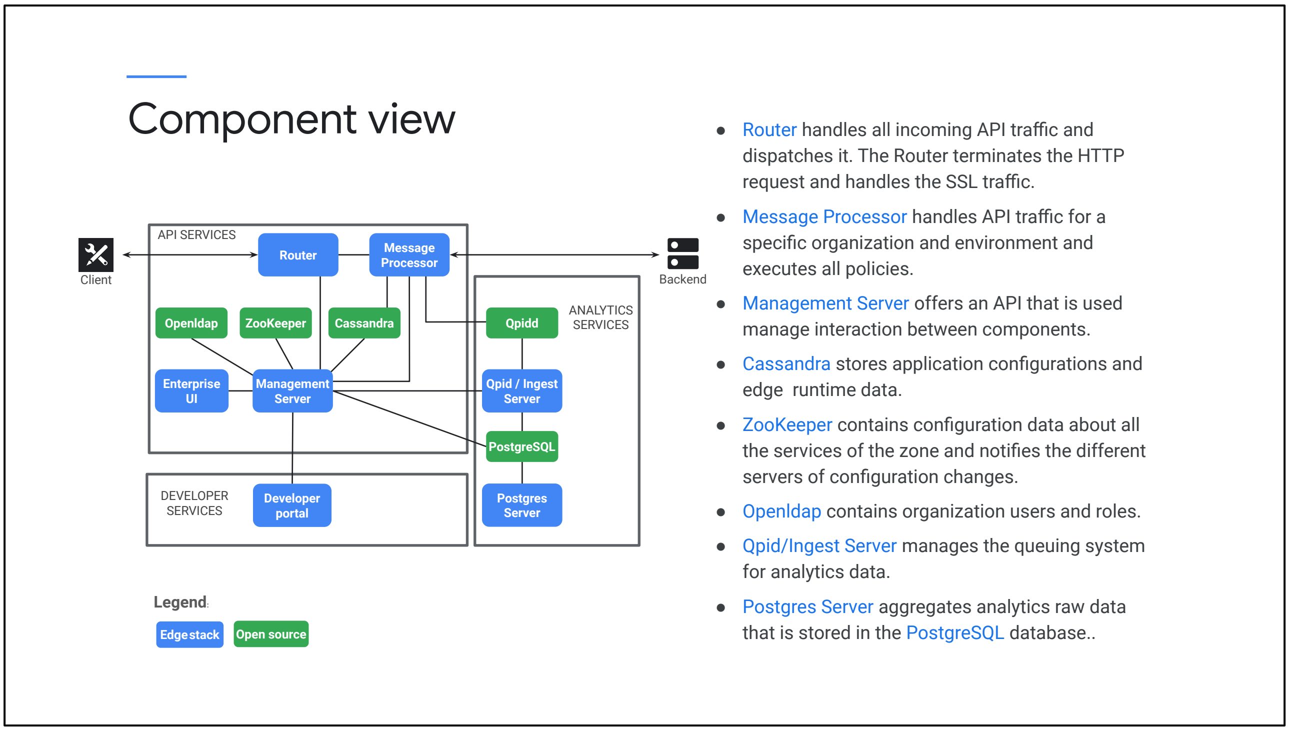
Terminology and organizational structure

Topology design

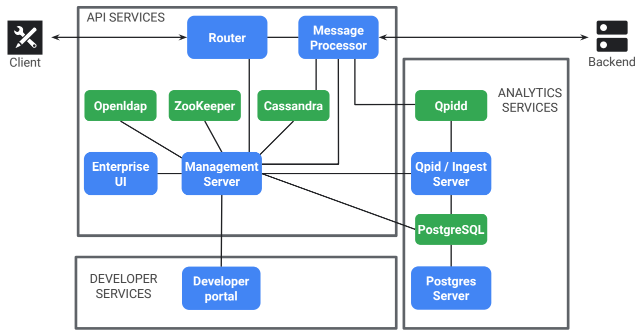
Welcome to the topology design lesson.

In this lesson, we explore key concepts on the design layout of Apigee components across your physical or virtual infrastructure.

Let's get started.



Component view



Legend:

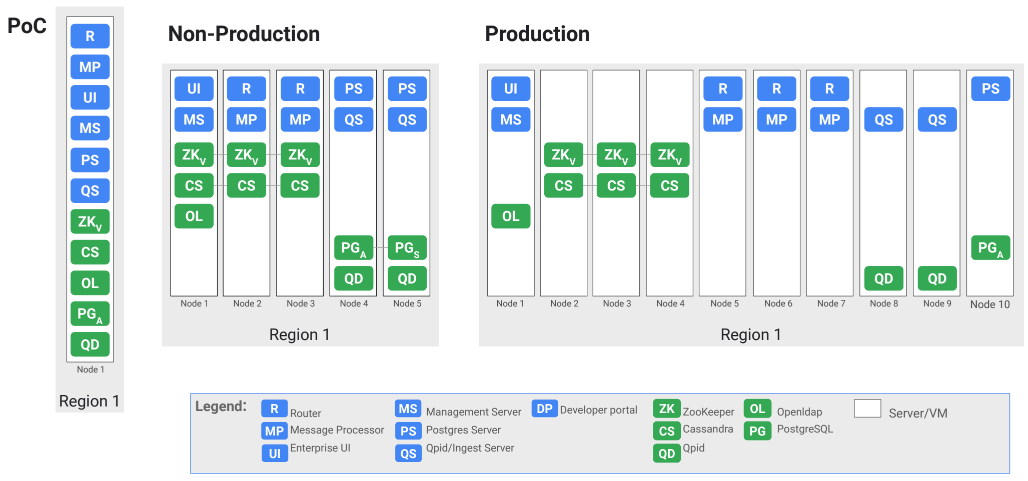
Edge stack | Open source

- Router handles all incoming API traffic and dispatches it. The Router terminates the HTTP request and handles the SSL traffic.
- Message Processor handles API traffic for a specific organization and environment and executes all policies.
- Management Server offers an API that is used manage interaction between components.
- Cassandra stores application configurations and edge runtime data.
- ZooKeeper contains configuration data about all the services of the zone and notifies the different servers of configuration changes.
- Openldap contains organization users and roles.
- Qpid/Ingest Server manages the queuing system for analytics data.
- Postgres Server aggregates analytics raw data that is stored in the PostgreSQL database..

Before we discuss topology design, it is important to understand the various Apigee private cloud components, their roles and their relationship to each other.

You can pause the video to review this slide for a recap of the information presented earlier.

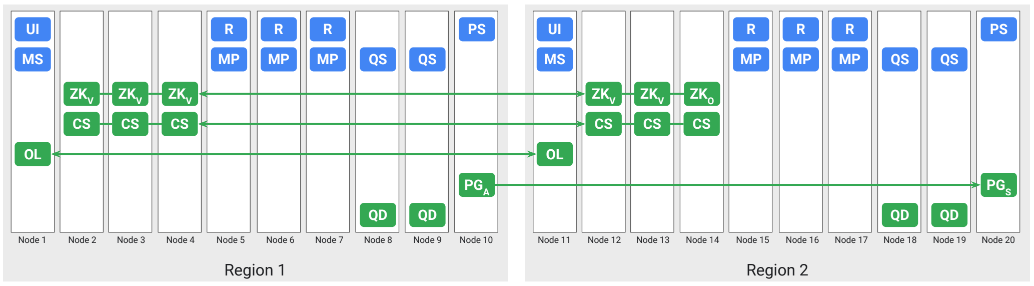
Topology examples



Apigee Edge for private cloud can be deployed on a single physical or virtual machine with all components on it, or to multiple machines or VMs in more than one region with an active/active configuration.

The platform has been designed to be horizontally scalable and multi-tenant, and is capable of running on both virtualized and physical hardware.

Architecture characteristics

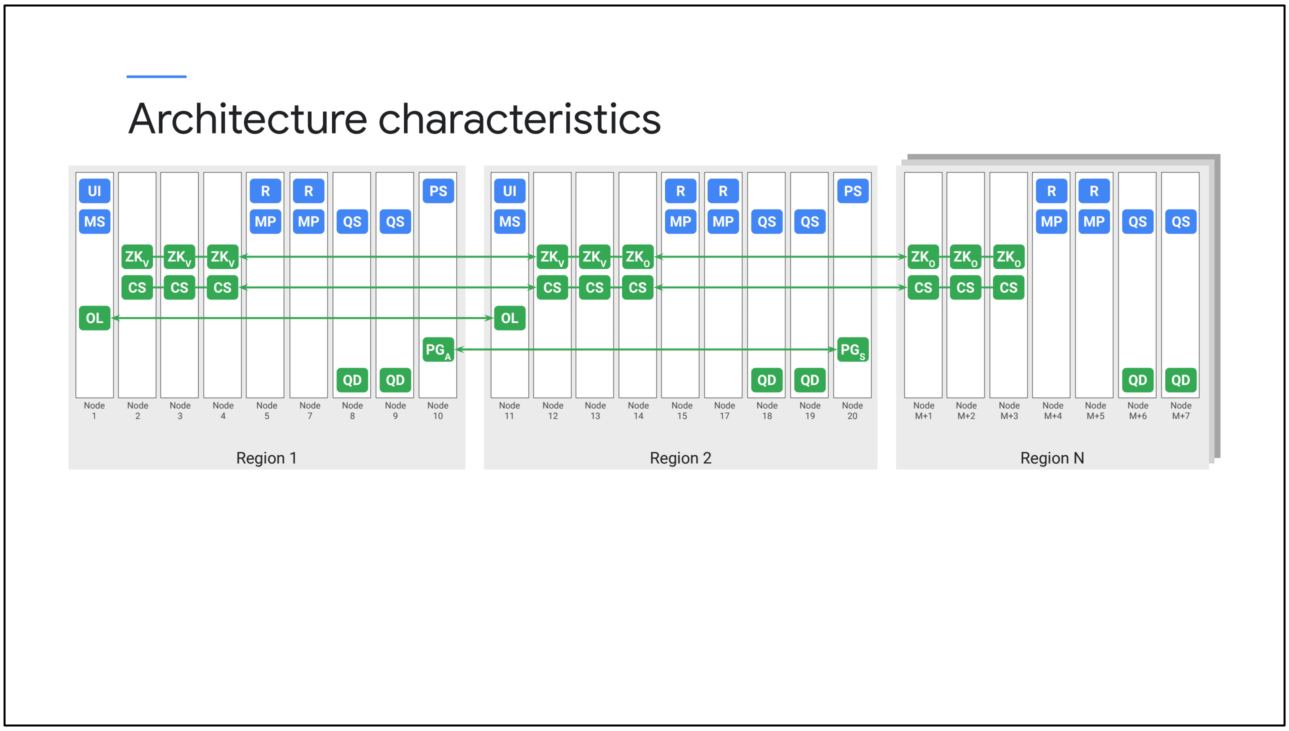


- Multi-tenant by design
- Horizontally scalable
- Ability to install components on the same node
- Asynchronous analytics data capture
- Centralized configuration for distributed components
- Management via APIs and UI console

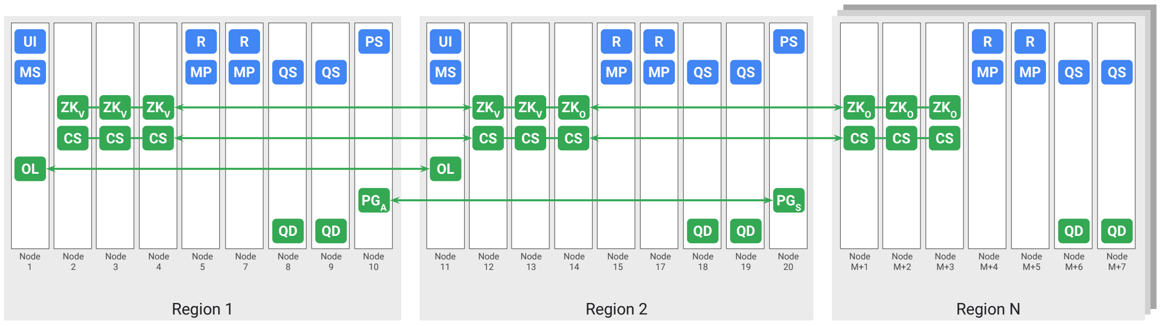
There are two regions in this diagram. Both regions can be managed by any one of the Management Servers and UI consoles.

Most Apigee Edge Components are horizontally scalable. Components can be installed and configured to run on a single node using an all-in-one configuration, or on multiple nodes in more than one data center, in an active/active, globally distributed setup. Some components can be installed on the same server to better utilize its resources.

By design, Apigee Edge is active active. That means there is continuous data replication across data stores that are located in all regions.



Architecture characteristics



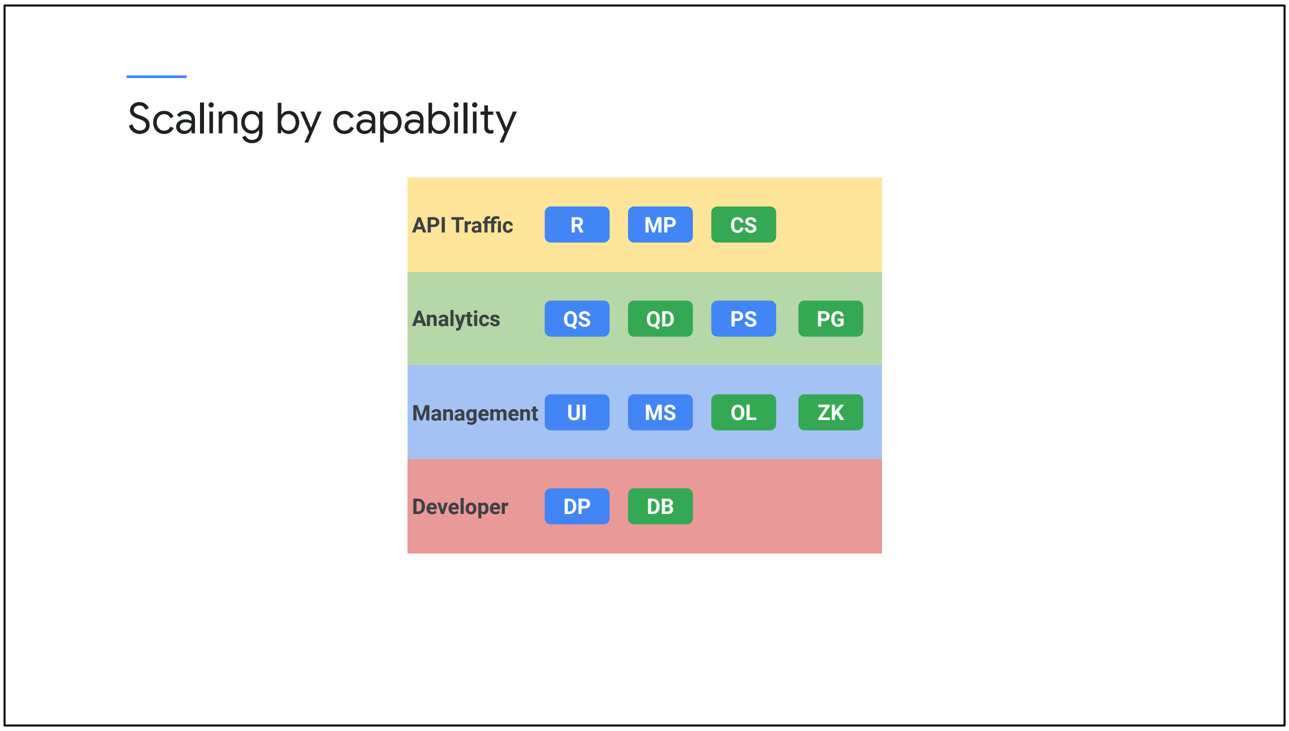
The Apigee Edge planet footprint is driven by requirements such as transaction volumes, availability and reliability as well as other requirements.

Note that the router, the message processor and Cassandra components are part of the critical path for runtime API traffic.

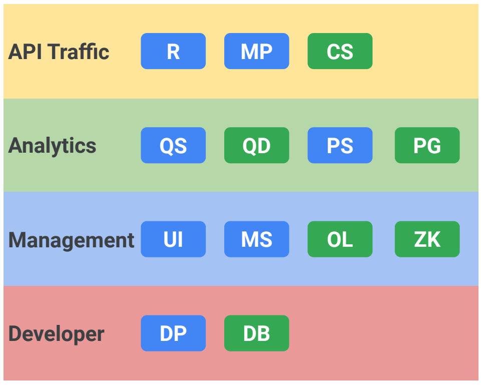
The management and analytics components are not required in every region.

For high availability, the ZooKeeper and Qpid components should be available in all regions.

Analytics data replication is managed using an active/standby configuration of PostgreSQL.



Scaling by capability



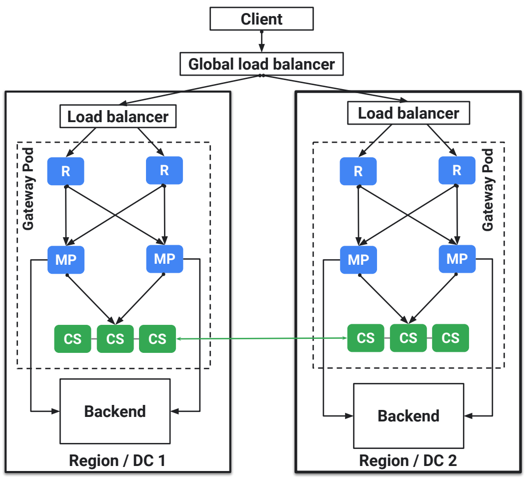
Given the responsibility and capabilities offered by each component, their scalability requirements may vary.

In most scenarios, scaling to accommodate higher API volume may impact only components that serve live API traffic.

Analytics data components may have to be scaled in response to increased API traffic or the analytics data retention policies of your organization.

Other components may scale based on the availability requirements of the capabilities offered by those components.

API traffic data flow



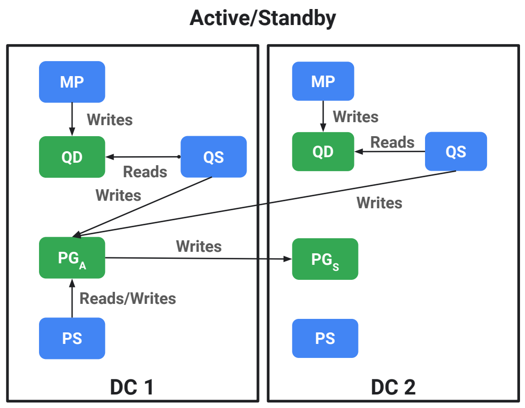
API traffic is routed to one or both regions based on customer requirements, and is usually handled by a global load balancer or DNS resolution.

Routers send requests to Message Processors within the same Gateway pod and serve the environments that are associated to the corresponding virtual hosts.

If needed, Message Processors perform read/write operations to Cassandra within the same pod/region.

Communication between Message Processors and backend systems is driven by API Proxy implementation and configuration.

Analytics data flow

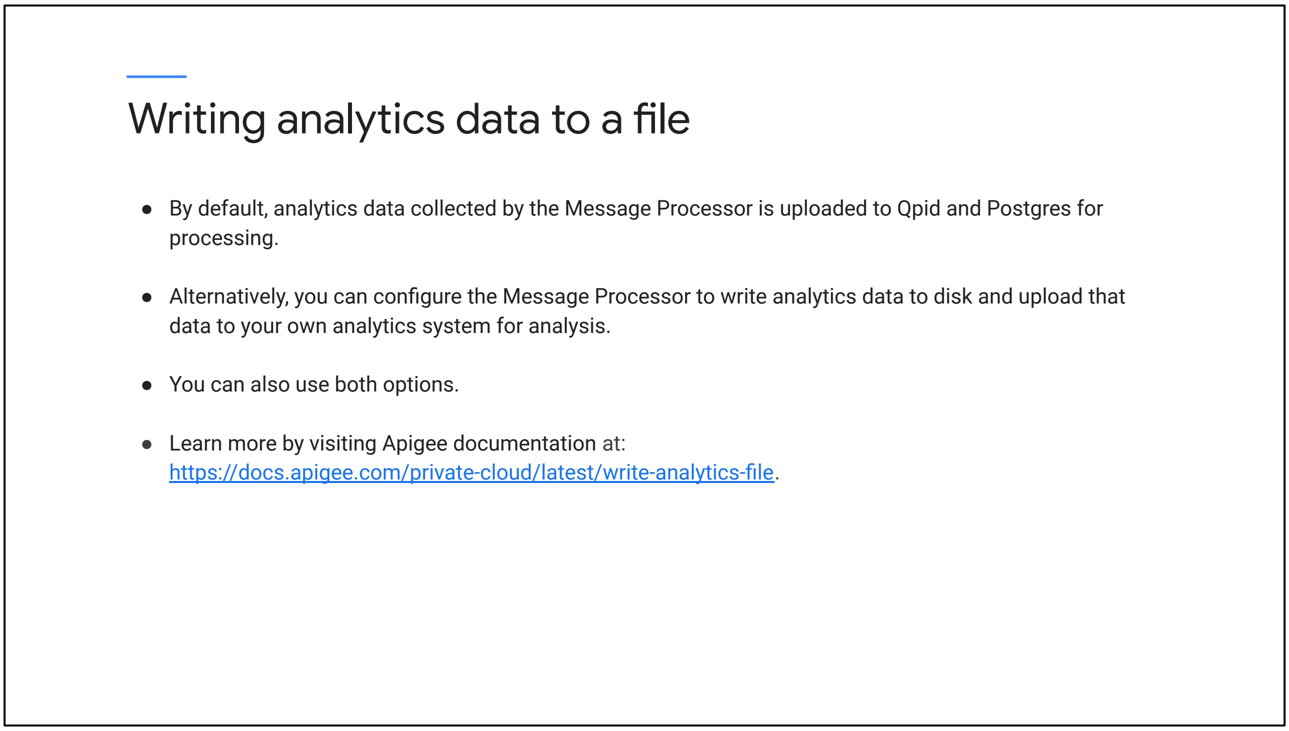


Analytics data is generated by Message Processors and asynchronously send to the Qpid component.

The Qpid server process consumes raw analytics data from Qpid and stores it in the active PostgreSQL database node.

The Postgres Server aggregates the analytics data in the active PostgreSQL database and makes it available for use in analytics reports.

Analytics data is partitioned by Organization and Environment.



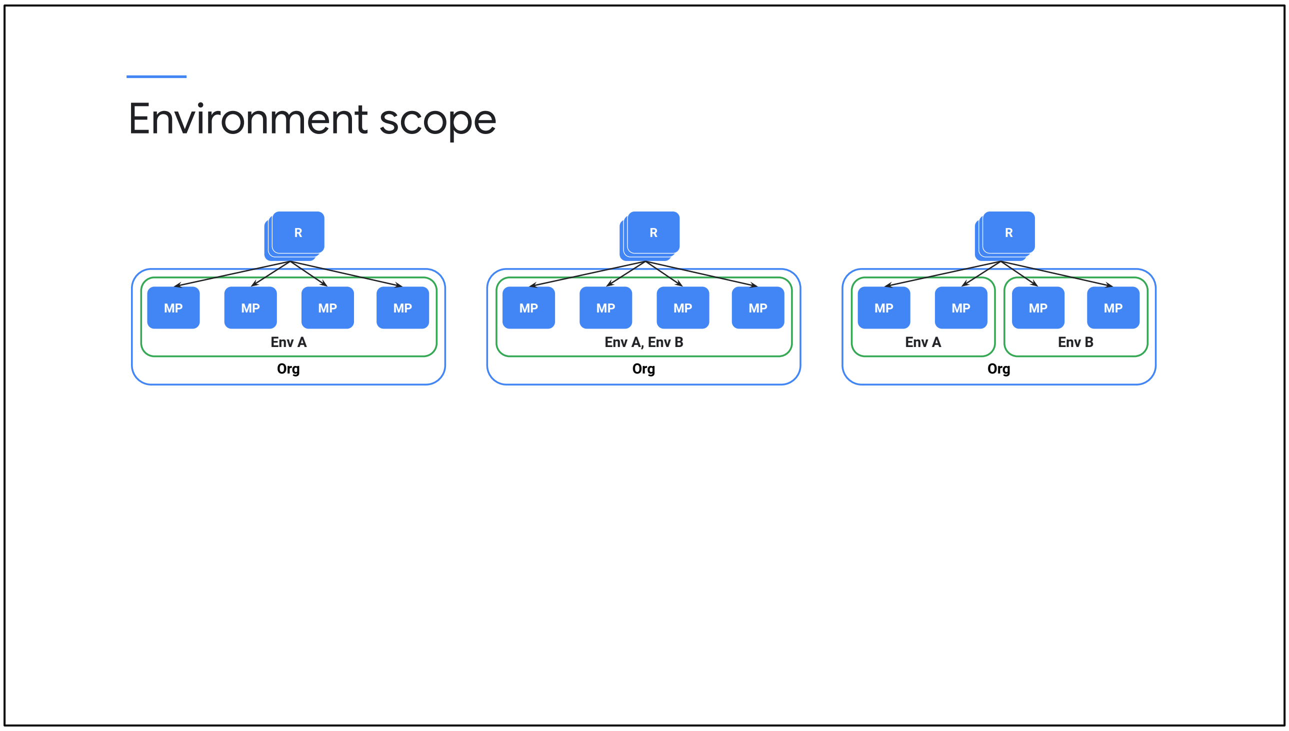
Writing analytics data to a file

- By default, analytics data collected by the Message Processor is uploaded to Qpid and Postgres for processing.
- Alternatively, you can configure the Message Processor to write analytics data to disk and upload that data to your own analytics system for analysis.
- You can also use both options.
- Learn more by visiting Apigee documentation at: https://docs.apigee.com/private-cloud/latest/write-analytics-file.

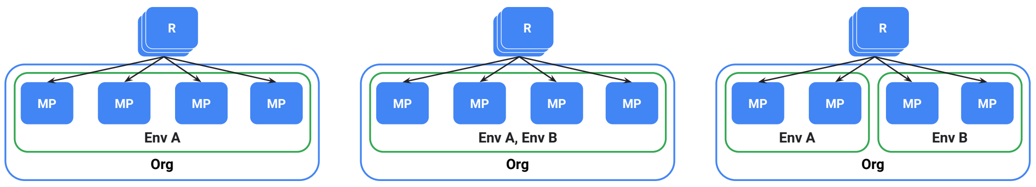
By default, analytics data that is collected by the Message Processor is uploaded to Qpid and PostgreSQL for processing. You can then view the analytics data in the Edge UI.

Alternatively, you can configure the Message Processor to write analytics data to disk. You can then upload that data to your own analytics system for analysis. For example, you could upload the data to BigQuery. You can then take advantage of the powerful query and machine learning capabilities offered by BigQuery and TensorFlow to perform your own data analysis.

You can also use both options to upload the analytics data to Qpid and PostgreSQL and also save the data to disk.



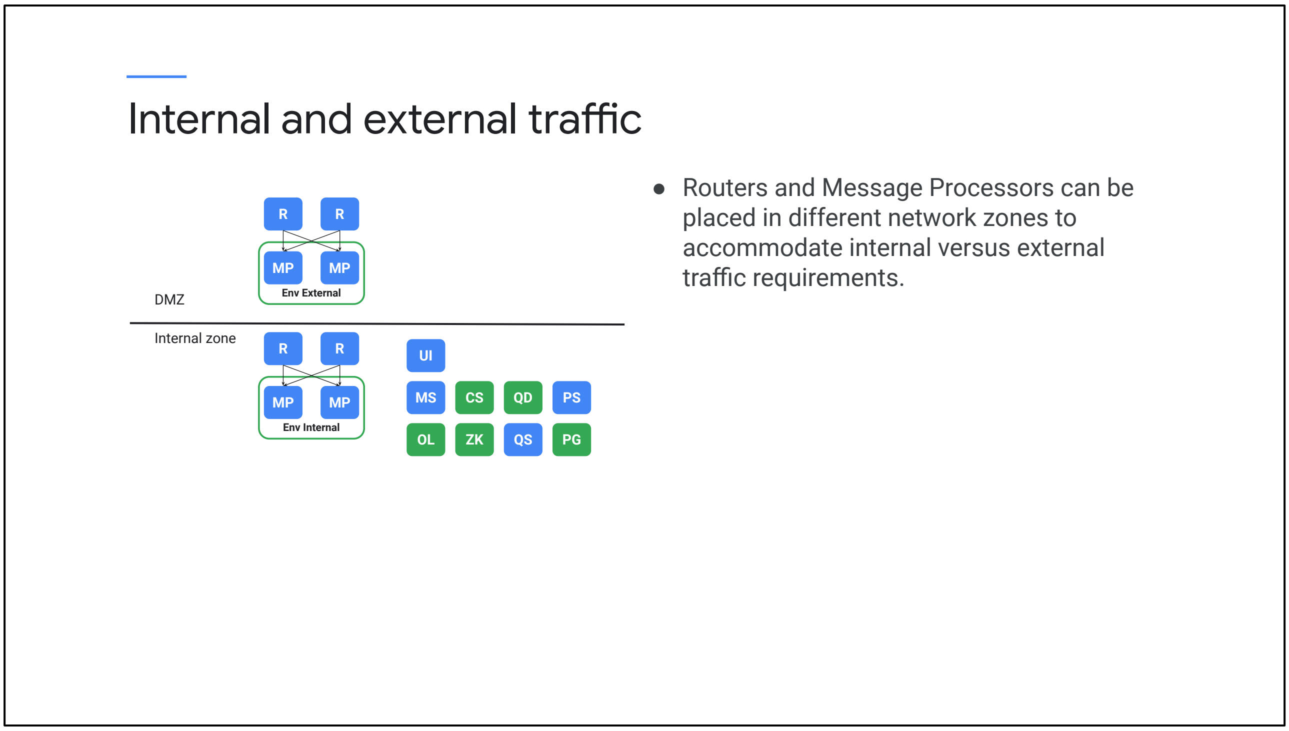
Environment scope



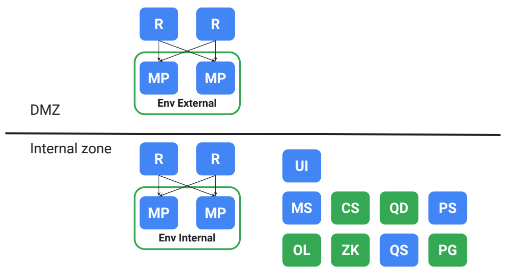
An environment defines a runtime execution context and scope for your API proxies.

Message Processors are associated to environments.

Routers are multi-tenant aware and are capable of sending API traffic to any message processor that is associated to the environment.



Internal and external traffic



- Routers and Message Processors can be placed in different network zones to accommodate internal versus external traffic requirements.

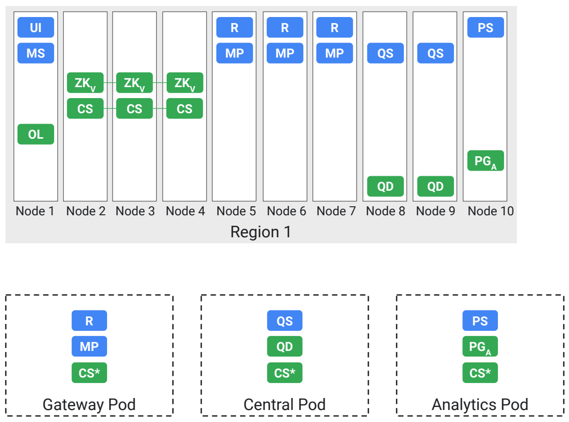
External and internal environments can be part of the same organization.

A single planet containing the Cassandra, ZooKeeper and other components can be shared and allocated to both the external and internal environments.

This allows partial physical separation of runtime execution, and logical data partitioning that is driven by the use of environments.

Pods

Pods are a logical grouping of components by function.



- Pods are managed using the management API.

The Apigee for private cloud installation process create pods and registers Edge components in the pods.

A Gateway pod contains Routers and Message Processors. A Central pod contains Qpid servers. An Analytics pod contains Postgres servers.

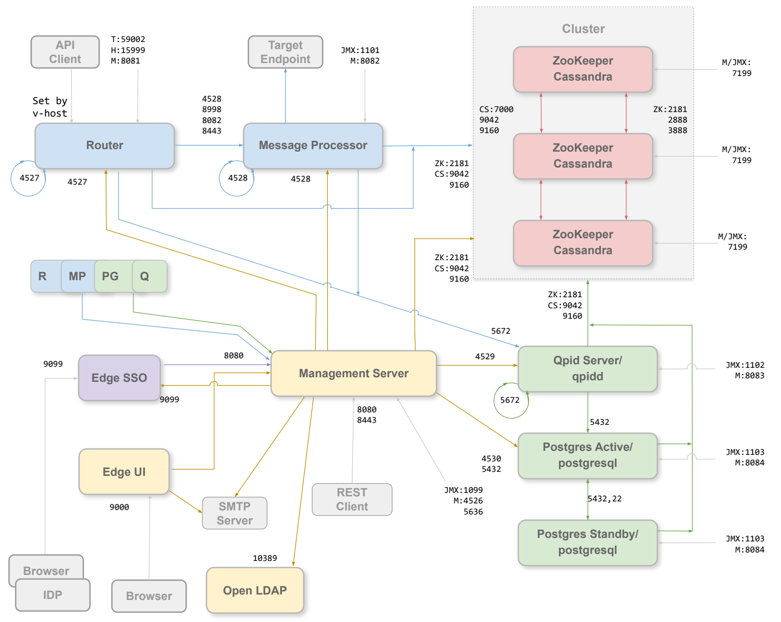
Each pod contains references to Cassandra keyspaces associated to the functions of the components registered in it.

To expand or reduce capacity you can horizontally scale the pods in the installation.

When pods are scaled, the components are registered or deregistered from the pods.

Components connectivity

Single region view



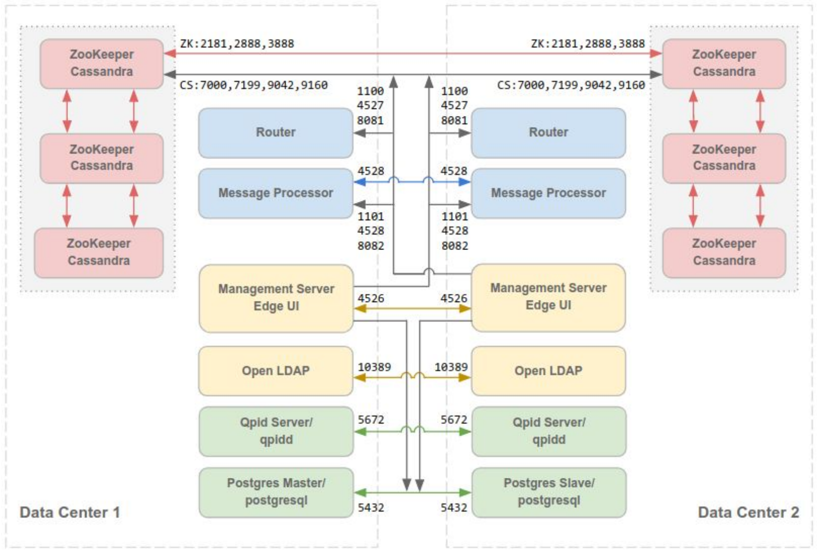
Here is a single region view of the connections between the various Apigee Edge components.

Note that the need to manage the firewall goes beyond just the virtual hosts.

Both virtual and physical host firewalls must allow traffic to various ports that are required by the components to communicate with each other.

Components connectivity

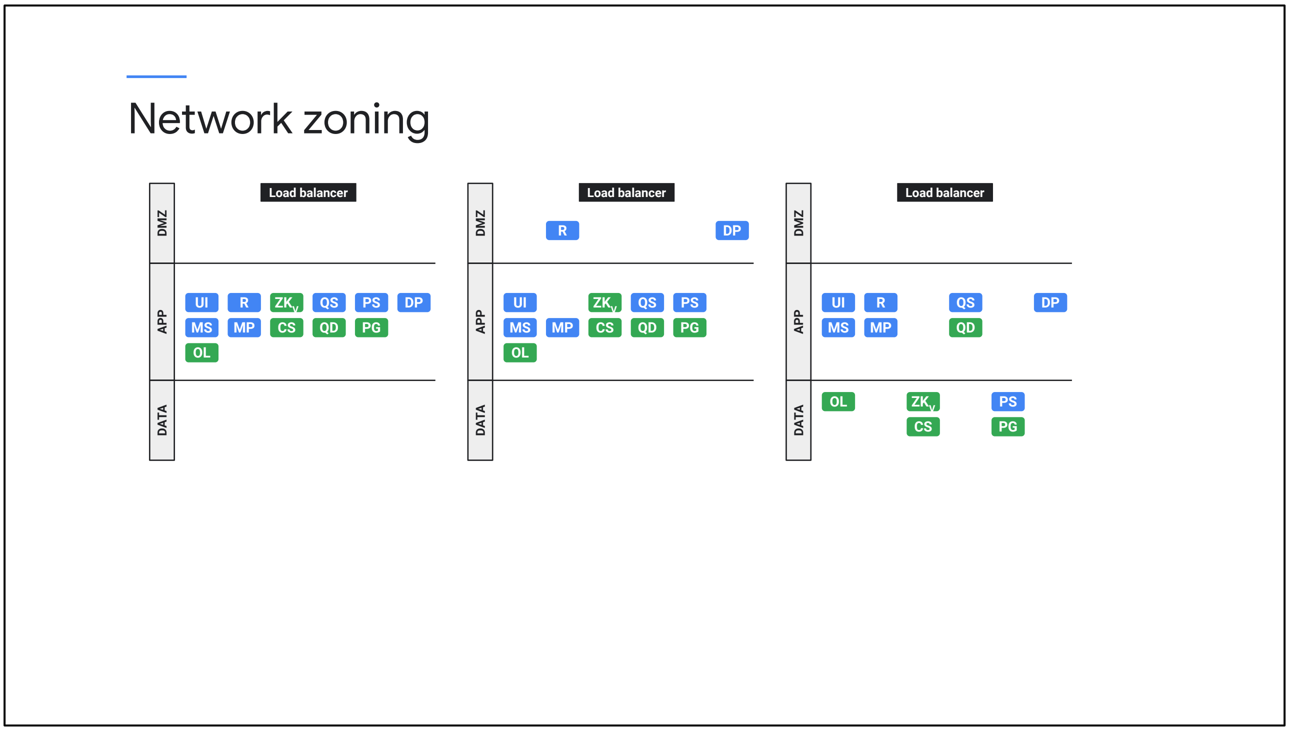
Multi-region view



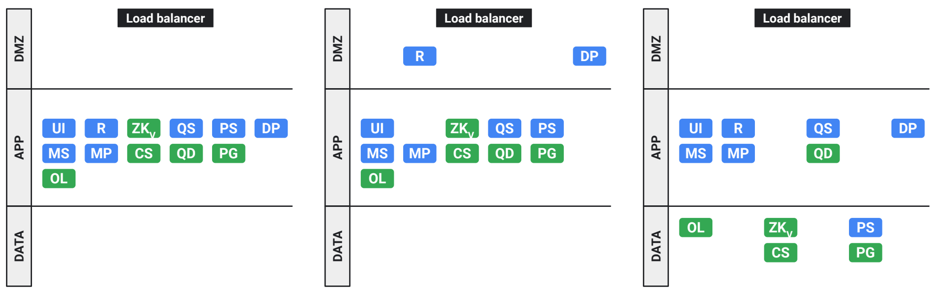
This is a multi-region view of the connections between components across two data centers.

If you install Apigee in multiple regions you have to open additional network ports for cross region connectivity between Apigee components.

Please pause the video to examine the specific ports required for component communication.



Network zoning

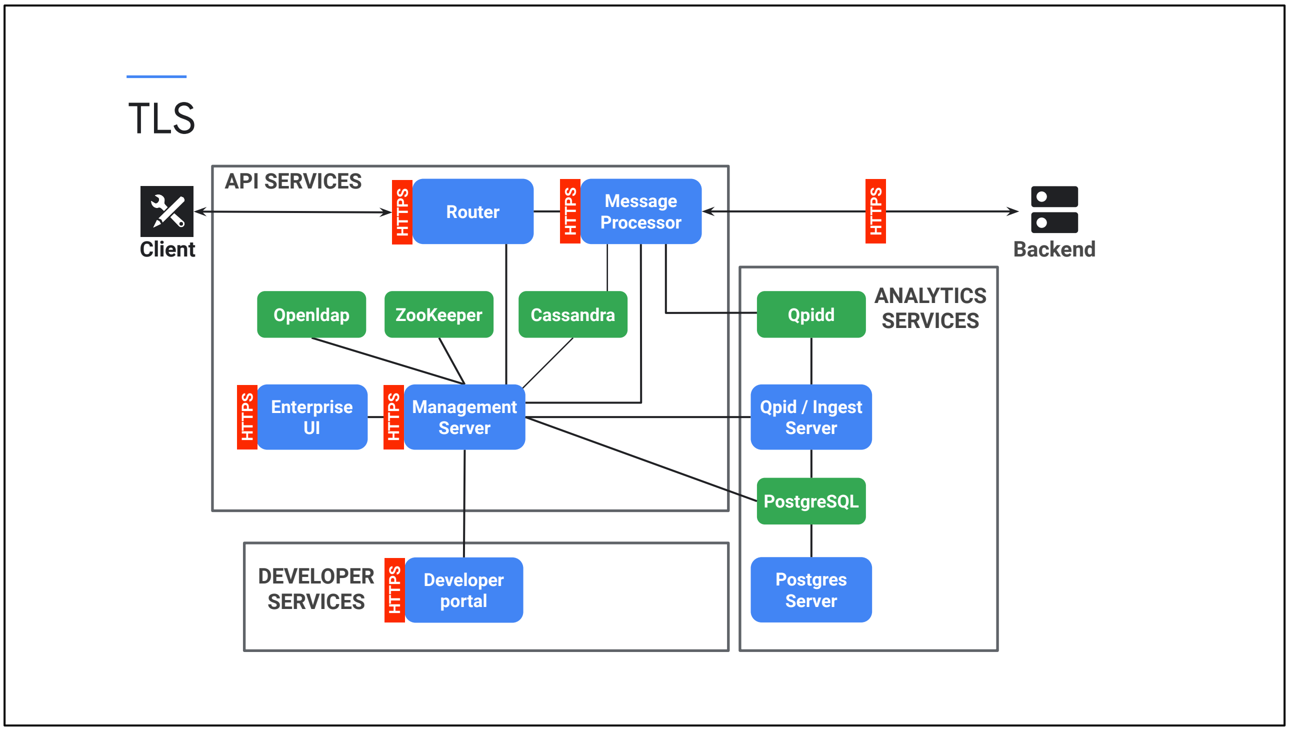


The Apigee Edge private cloud architecture does not impose network zone requirements.

The examples shown are three different configurations that could be used to locate the Apigee Edge components from a networking perspective.

Network zoning will be driven by customer operation and security requirements.

Firewalls and security appliances between zones should support the connectivity and traffic characteristics of Edge components without adding significant latency overhead.



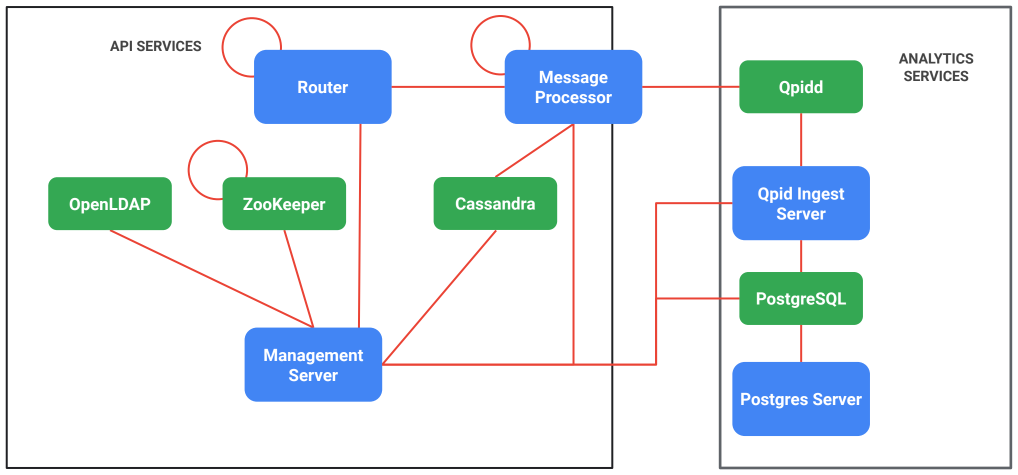
Transport layer security or TLS can be enabled on the API runtime critical path and on the interfaces that are exposed for administration and management functions.

You can also enable TLS on the developer portal.

TLS can be terminated at any of these points independently of the other services.

We discuss how to configure TLS in Apigee Edge in the security module of the management, security and upgrade course of this series.

Secure communication between components



Apigee uses a service mesh of Consul sidecars that run on each node. It establishes secure, mutually authenticated TLS connections between components.

Each Consul sidecar stores configuration data and manages a pair of services:

An egress service that intercepts outgoing messages from the host node so that it can encrypt them before sending them to their destination.

An ingress service that intercepts incoming messages to the host node so that it can decrypt them.

Review: Architecture

Hansel Miranda
Technical Curriculum Developer, Google

In this module you learned about the Apigee product and the services it provides to help customers implement their digital value chain.

You learned about the technology stack including Apigee Cassandra and the terminology and organizational structure used in Apigee Edge for private cloud.

We also discussed the architecture and topology of the platform

This knowledge represents the foundation that you need to understand the Apigee private cloud product and helps you master the remaining modules in this course.

Google Cloud