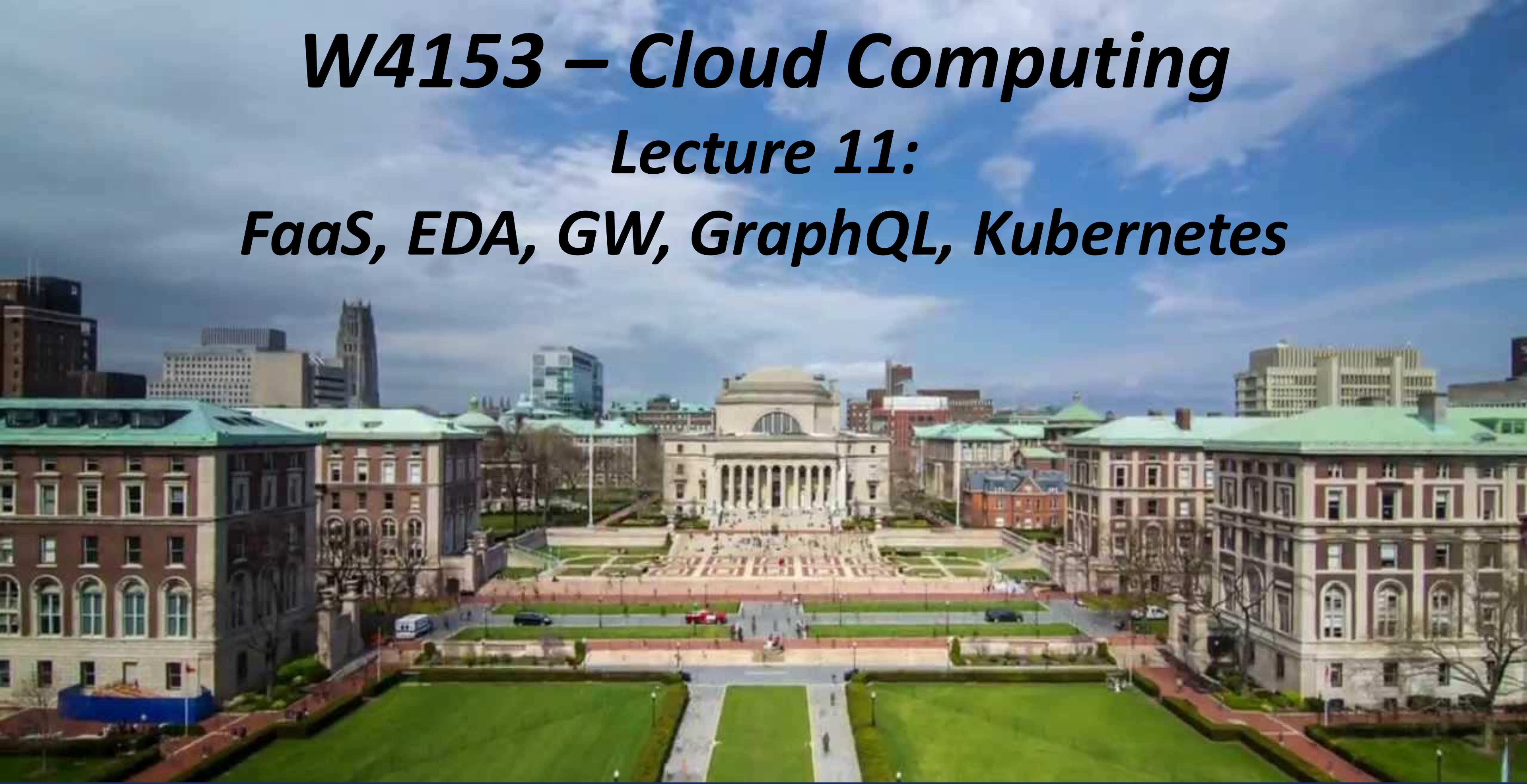


W4153 – Cloud Computing

Lecture 11:

FaaS, EDA, GW, GraphQL, Kubernetes

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

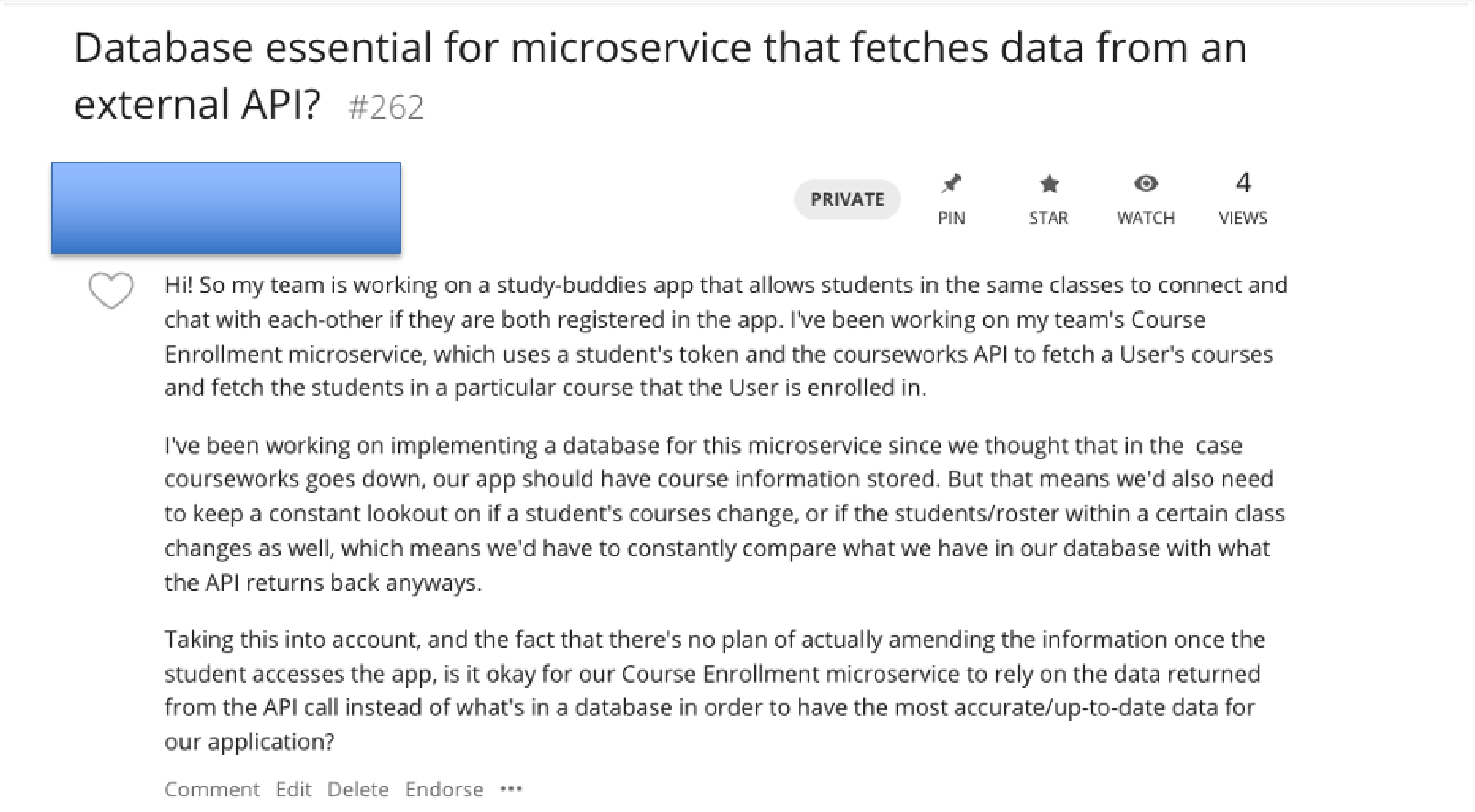
Contents

Contents

- Interesting Ed question
- Implementing and deploying application logic and microservices — overview, reminder, options.
- Function-as-a-Service
- More advanced composition
  - Event driven architecture
  - ~~Message driven architecture~~
- API Gateway
- ~~GraphQL~~

Interesting Ed Question

Interesting Ed Question



Database essential for microservice that fetches data from an external API? #262

[redacted]

PRIVATE PIN STAR WATCH 4 VIEWS

Hi! So my team is working on a study-buddies app that allows students in the same classes to connect and chat with each-other if they are both registered in the app. I've been working on my team's Course Enrollment microservice, which uses a student's token and the courseworks API to fetch a User's courses and fetch the students in a particular course that the User is enrolled in.

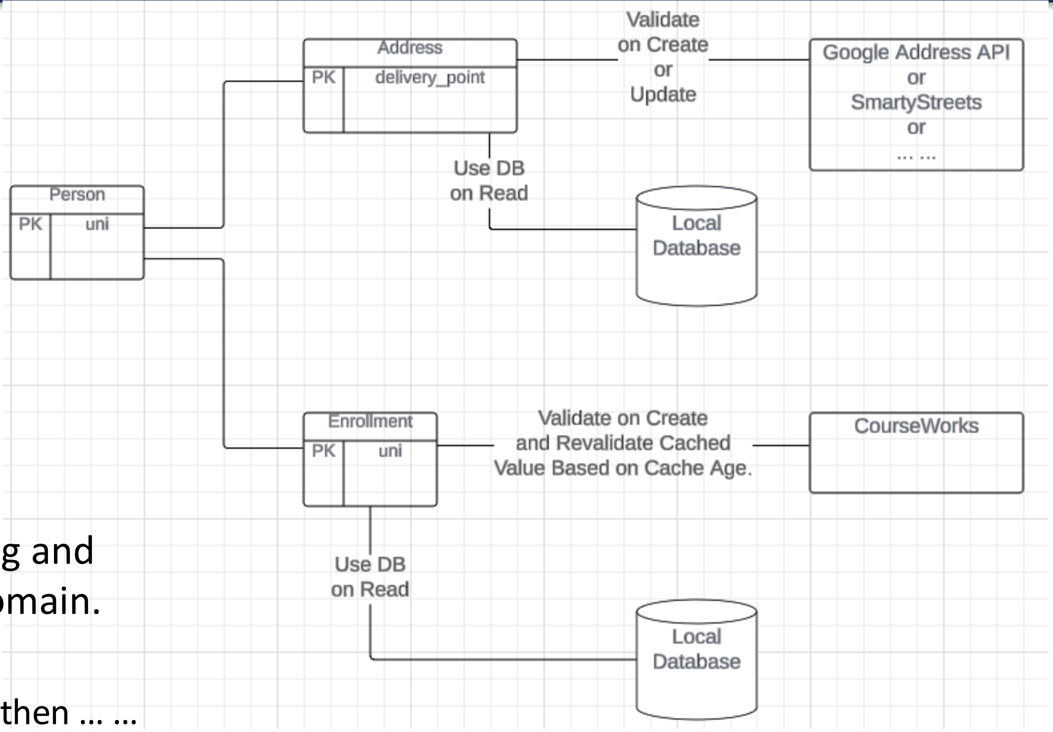
I've been working on implementing a database for this microservice since we thought that in the case courseworks goes down, our app should have course information stored. But that means we'd also need to keep a constant lookout on if a student's courses change, or if the students/roster within a certain class changes as well, which means we'd have to constantly compare what we have in our database with what the API returns back anyways.

Taking this into account, and the fact that there's no plan of actually amending the information once the student accesses the app, is it okay for our Course Enrollment microservice to rely on the data returned from the API call instead of what's in a database in order to have the most accurate/up-to-date data for our application?

Comment Edit Delete Endorse •••

My Examples

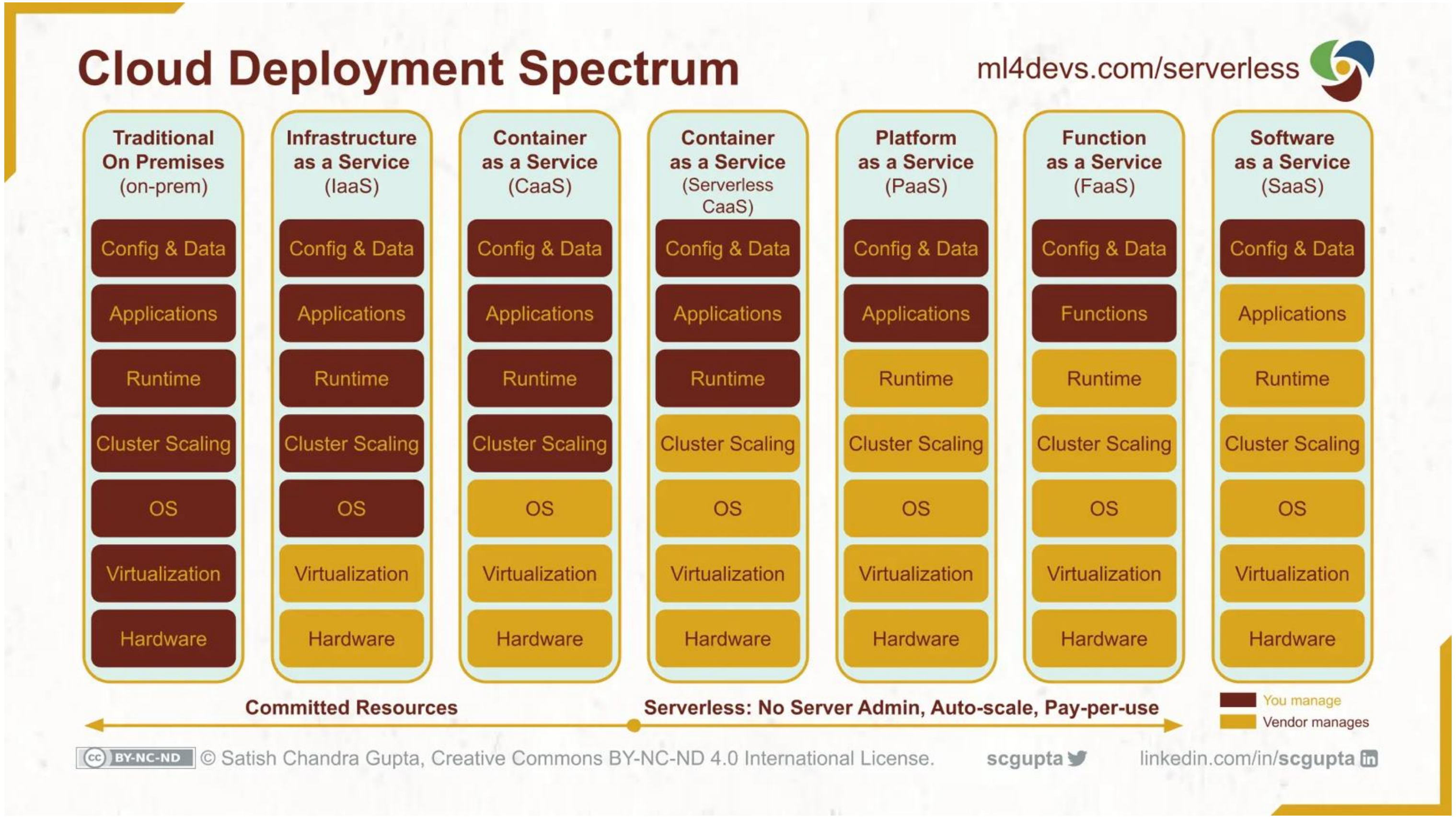
- My example application uses:
  - CourseWorks
  - An address verification API
    - SmartyStreets
    - Google
- In general, "always" access the API is not the best practice:
  - Performance, reliability, … …
  - API usage costs
- I typically implement some form of caching and verification based on knowledge of the domain.
  - Validate on an update
  - Refresh of read when cache entry is older then … …
- Some application support subscribing to change notification events.



Person
PK uni
Address
PK delivery_point
Validate on Create or Update
Google Address API or SmartyStreets or … …
Use DB on Read
Local Database
Enrollment
PK uni
Validate on Create and Revalidate Cached Value Based on Cache Age.
CourseWorks
Use DB on Read
Local Database

Implementing and Deploying Microservices

Cloud Layers



Cloud Deployment Spectrum

ml4devs.com/serverless

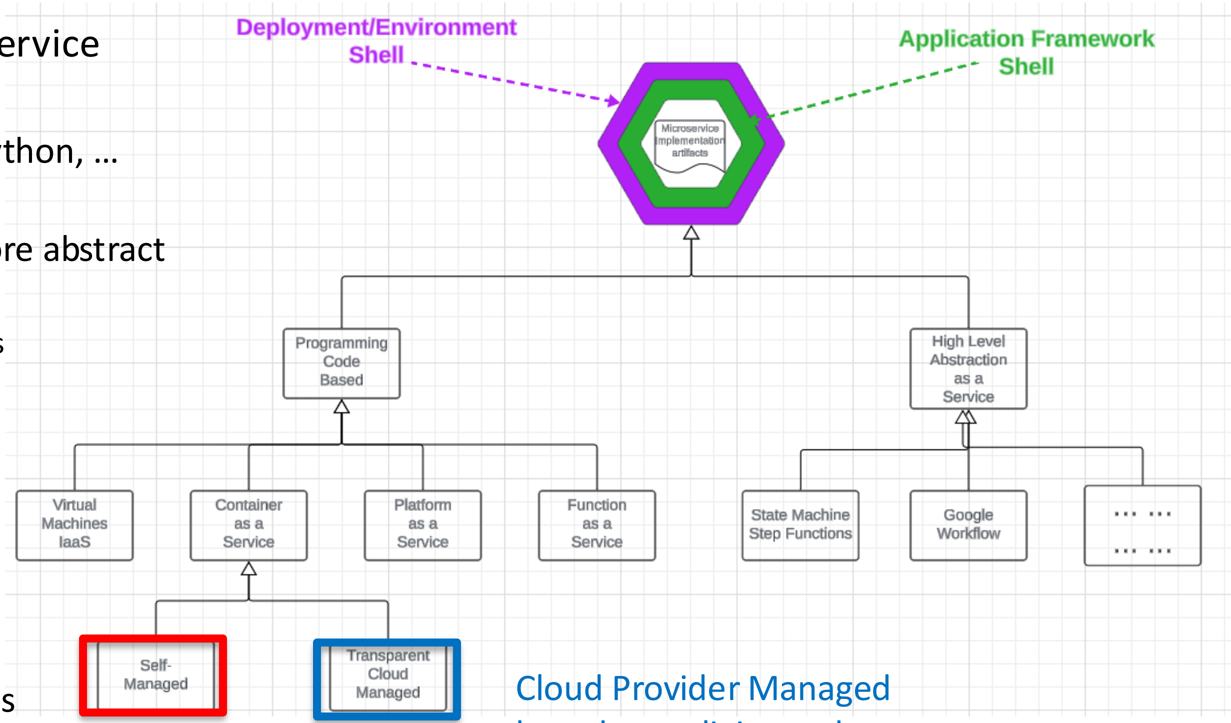
Traditional On Premises (on-prem)	Infrastructure as a Service (IaaS)	Container as a Service (CaaS)	Container as a Service (Serverless CaaS)	Platform as a Service (PaaS)	Function as a Service (FaaS)	Software as a Service (SaaS)
🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data	🟥 Config & Data
🟥 Applications	🟥 Applications	🟥 Applications	🟥 Applications	🟥 Applications	🟥 Functions	🟨 Applications
🟥 Runtime	🟥 Runtime	🟥 Runtime	🟥 Runtime	🟨 Runtime	🟨 Runtime	🟨 Runtime
🟥 Cluster Scaling	🟥 Cluster Scaling	🟥 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling	🟨 Cluster Scaling
🟥 OS	🟥 OS	🟨 OS	🟨 OS	🟨 OS	🟨 OS	🟨 OS
🟥 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization	🟨 Virtualization
🟥 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware	🟨 Hardware

Committed Resources ← → Serverless: No Server Admin, Auto-scale, Pay-per-use

🟥 You manage
🟨 Vendor manages

Implementing and Deploying Microservices

- There are several approaches or models for implementing microservice "application" logic.
  - "Code" in a language like Go, Python, … is the most common.
  - There are other, specialized, more abstract domain specific languages, e.g.
    - Step Functions/State Machines
    - Workflows
    - Data flows
    - … …
- A deployed microservice has three logical layers of functions:
  - Deployment shell functions.
  - Application framework functions
  - Application logic



Deployment/Environment Shell
Application Framework Shell
Microservice Implementation artifacts
Programming Code Based
High Level Abstraction as a Service
Virtual Machines IaaS
Container as a Service
Platform as a Service
Function as a Service
State Machine Step Functions
Google Workflow
… … … …
Self-Managed
Transparent Cloud Managed

For example, Elastic Kubernetes Service

Cloud Provider Managed based on policies and parameters.

Implementing and Deploying Microservices

- "Code" is by far the most common.

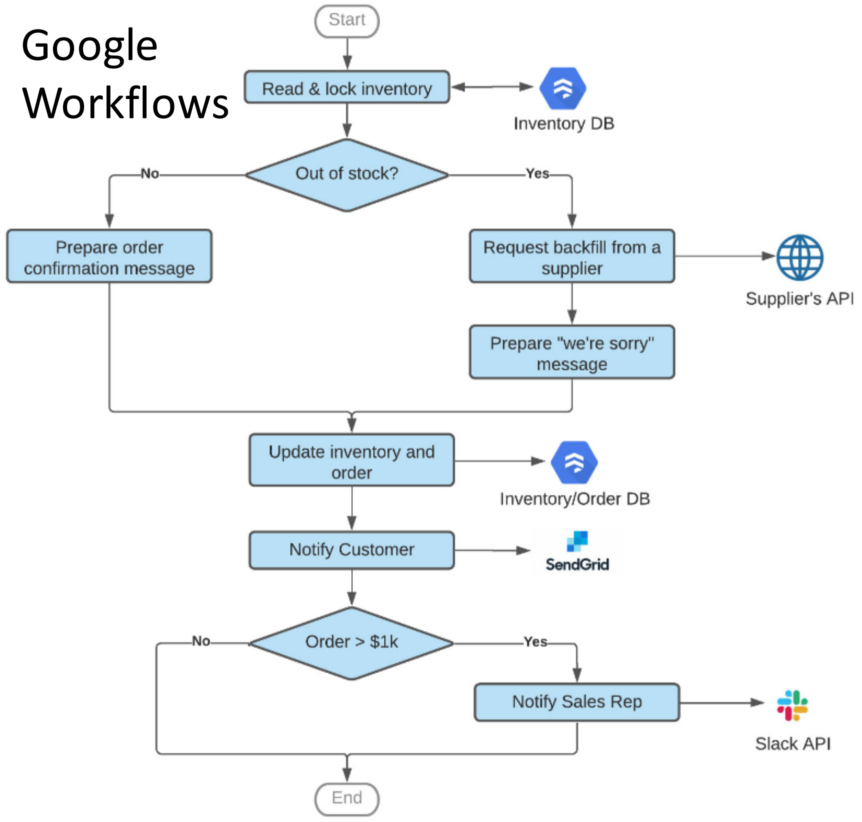
- Domain Specific Languages and their runtime environments:
  - Have a learning curve and seem complex.
  - Are extremely useful for certain complex scenarios.

- We have focused on "code" and will continue to do so.
  - We have seen a few deployment options: IaaS, Self-Managed CaaS, PaaS, …
  - We will look at more complex self-managed CaaS and understand, e.g. Kubernetes, but not use.
  - Today we will examine Function-as-a-Service.

- We will get some simple experience with a higher layer, DSL approach.

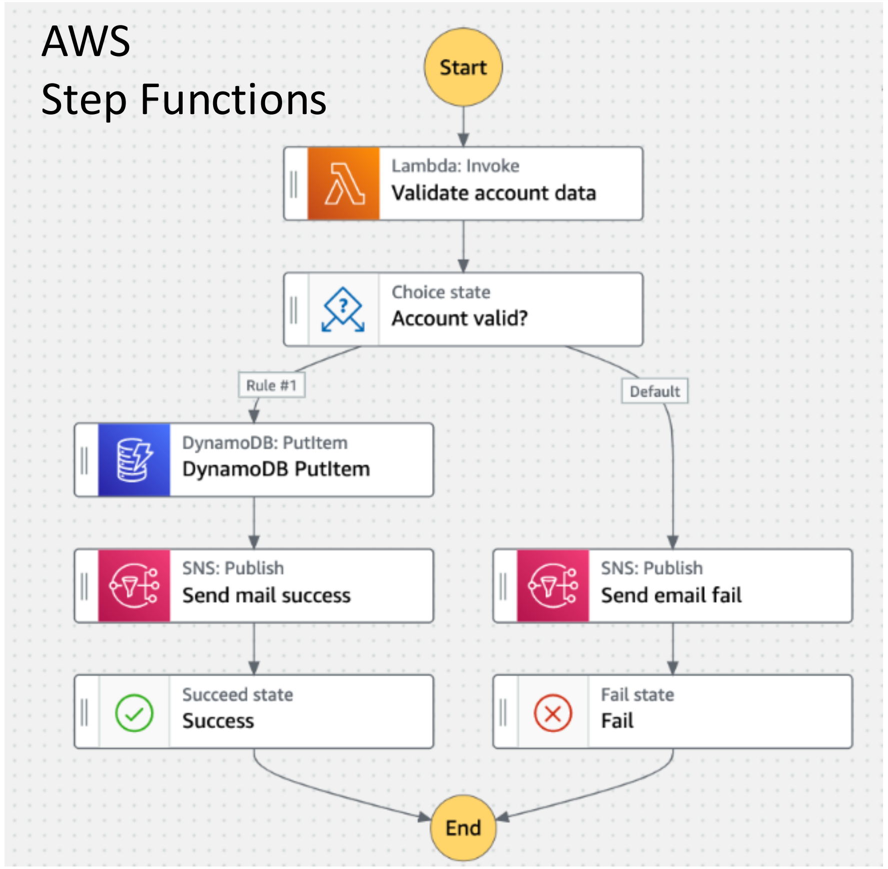
Higher Layer Languages

Google Workflows



https://cloud.google.com/blog/products/application-development/get-to-know-google-cloud-workflows

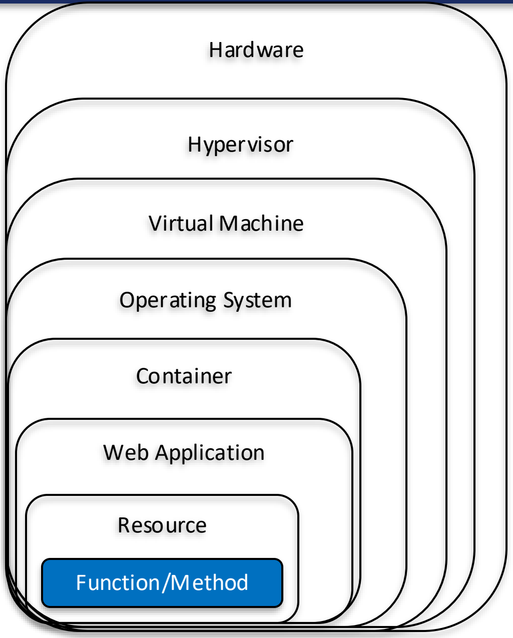
AWS Step Functions



Function-as-a-Service

Computer Models

- At the very core, you write a function or method.
- A set of functions or methods become a resource.
- A resource runs in a web application and uses required software, e. g. packages, libraries.
- A web application run in some kind of container, which provides isolation, ensures pre-reqs, etc.
- Containers run in an operating system.
- An operating systems runs on HW, which may be a virtual machine and virtual resources.
- Virtual resources run on hypervisors, which
- Run on hardware.
- This is like a *matryoshka doll*. Ultimately, I decide:
  - Which level of nesting is the package I am willing to/want to produce and deploy.
  - I provide that as a package/bundle and provide a declaration/manifest of what I want created and managed automatically for me.
- This is over simplified and the boundaries blur.



Serverless and Function-as-a-Service

- **Serverless computing** is a cloud computing execution model in which the cloud provider runs the server, and dynamically manages the allocation of machine resources. Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity.[1] It can be a form of utility computing.

  Serverless computing can simplify the process of deploying code into production. Scaling, capacity planning and maintenance operations may be hidden from the developer or operator. Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.[2] This should not be confused with computing or networking models that do not require an actual server to function, such as peer-to-peer (P2P)."

- **Function as a service** (**FaaS**) is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.[1] Building an application following this model is one way of achieving a "serverless" architecture, and is typically used when building microservices applications."

- That was baffling:
  - IaaS, CaaS – You control the SW stack and the cloud provides (virtual) HW.
  - PaaS – The cloud hides the lower layer SW and provides an application container (e.g. Flask) with "Your code goes here." You are aware of container.
  - FaaS – The cloud provides the container, and you implement functions (corresponding to routes in REST).

What is Serverless Good/Not Good For … …

Serverless is good for

short-running
stateless
event-driven

- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

Serverless is not good for

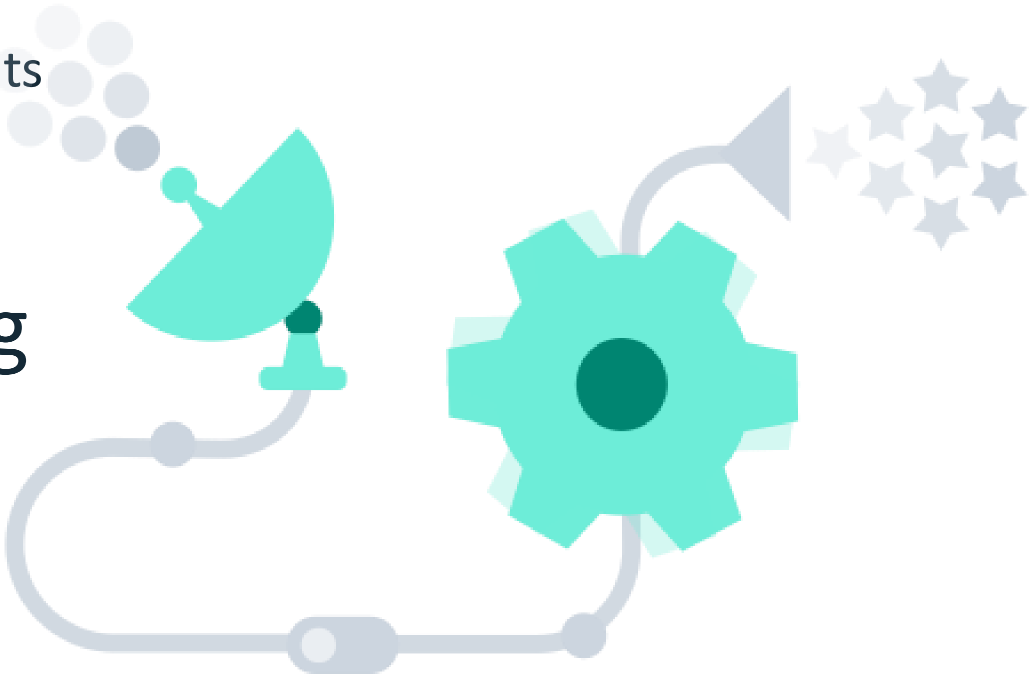
long-running
stateful
number crunching

- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

What triggers code execution?

Runs code **in response** to events

Event-programming model



(Some) Current Platforms for Serverless

APACHE OpenWhisk

AWS Lambda

OpenLambda

Iron.io

Azure Functions

fission

Kubernetes

IBM Cloud Functions

Red-Hat

Google Functions

fn

Google Cloud Function Example

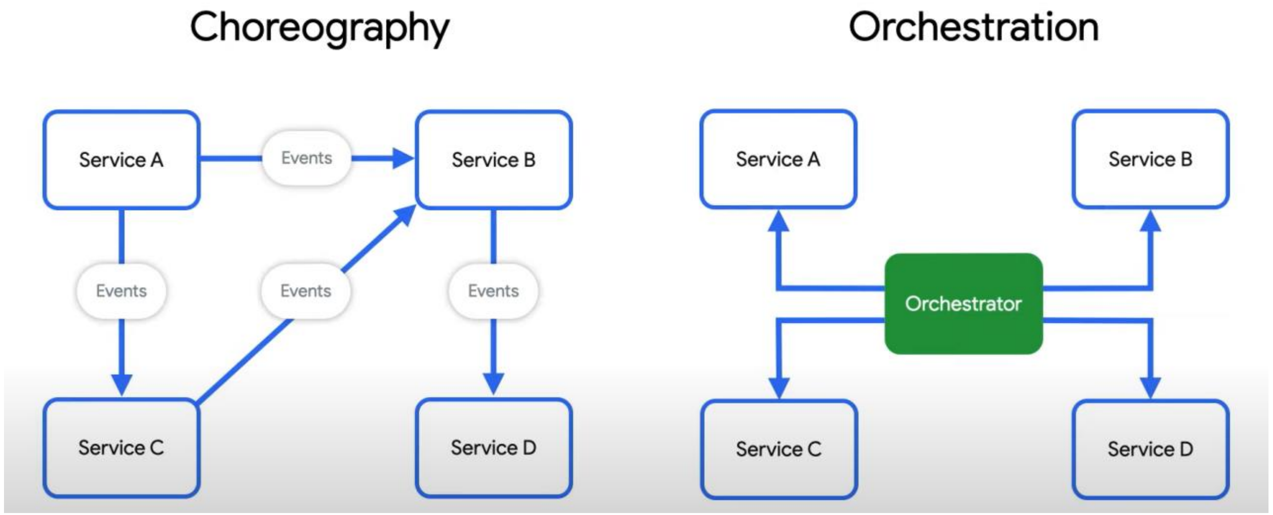
- Pub_Sub_New: /Users/donald.ferguson/Dropbox/000/00-Current-Repos/Pub_Sub_New

- Explain concepts
  - Discord
  - Google Cloud Function
  - Google Pub Sub

- Run demos

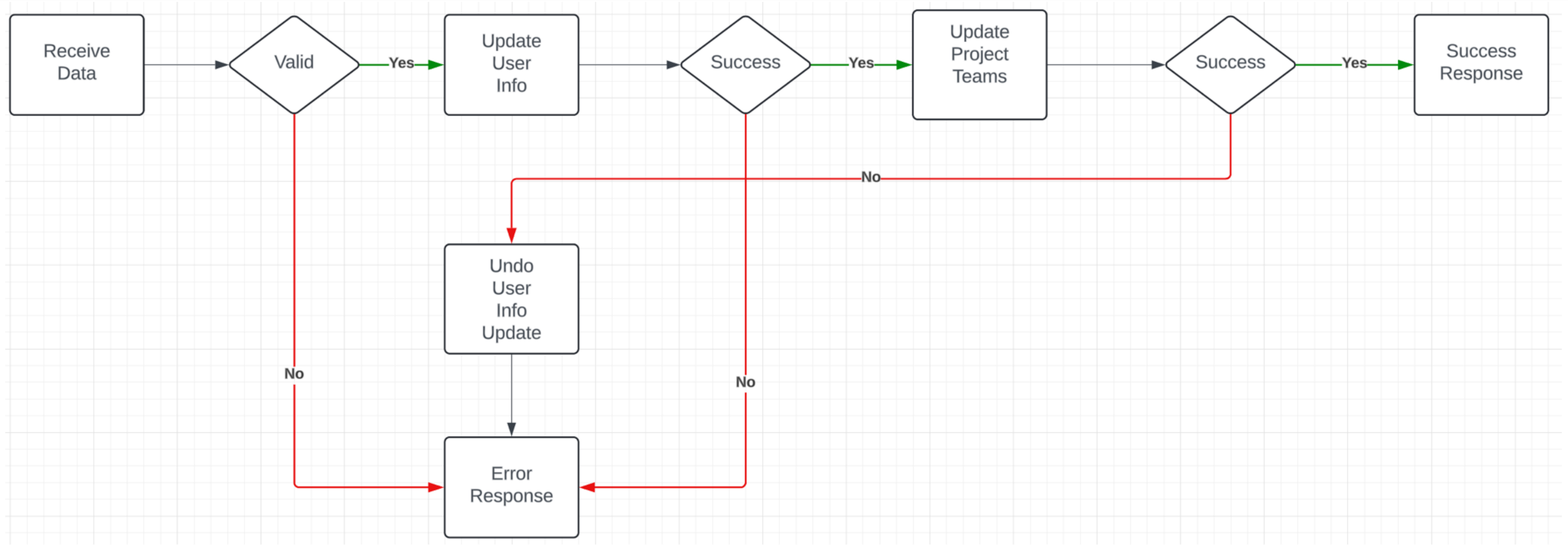
Composition

Behavioral Composition



- There are two base patterns: choreography and orchestration.
- There also several different implementation technologies and choices.
- And many related concepts, e.g. Sagas, Event Driven Architectures, Pub/Sub, … …
- We will cover and explore incrementally, but for now we will keep it simple and do traditional "if … then …" code.

Composite Microservice Logic



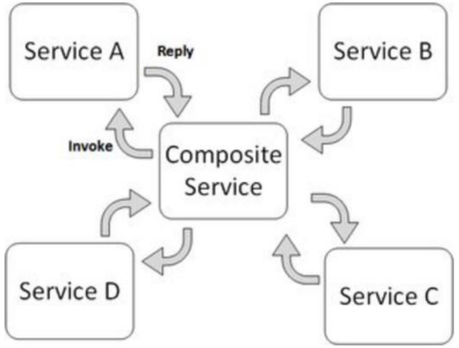
- The GET on the composite invokes GET on several atomic services. This can occur synchronously or asynchronously in parallel.
- PUT, POST and DELETE are more complex. Your logic may have to undo some delegated updates that occur if others fail.
- To get started, we will just use simple logic but explore more complex approaches.

Concepts

Service orchestration

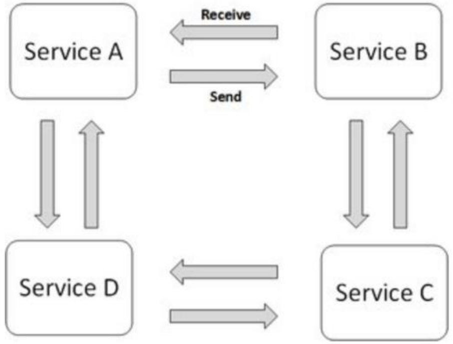
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

The relationship between all the participating services are described by a single endpoint (i.e the composite service). The orchestration includes the management of transactions between individual services. Orchestration employs a centralized approach for service composition.



Service Choreography

Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.

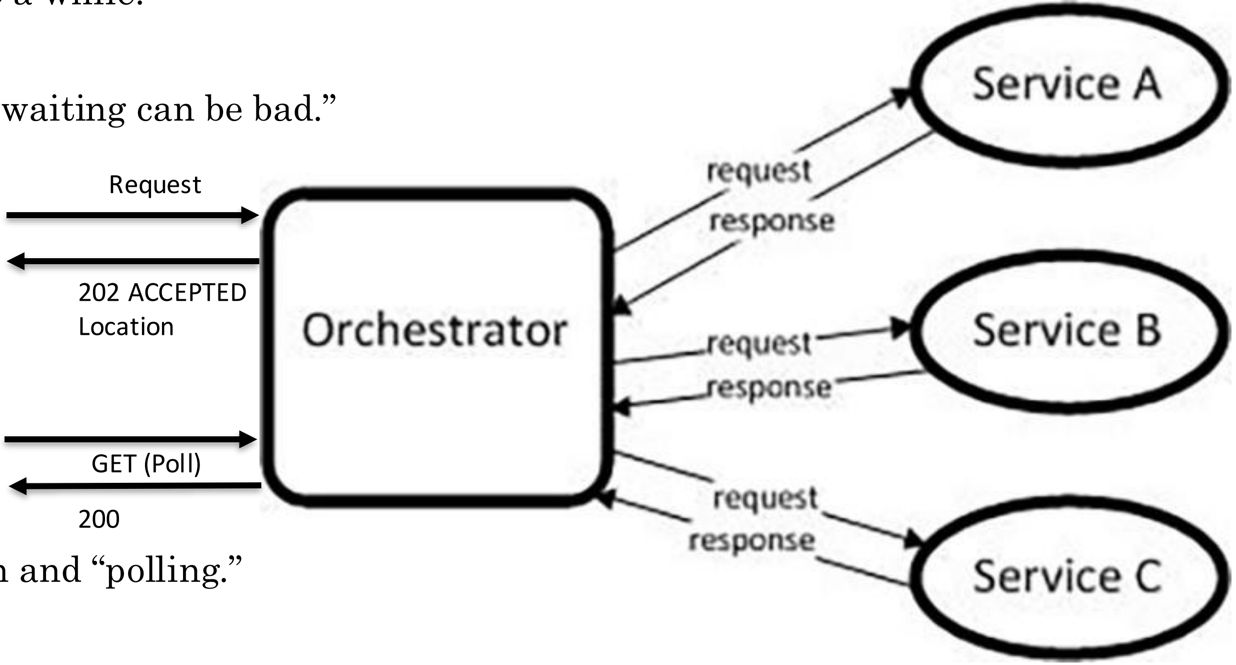


The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

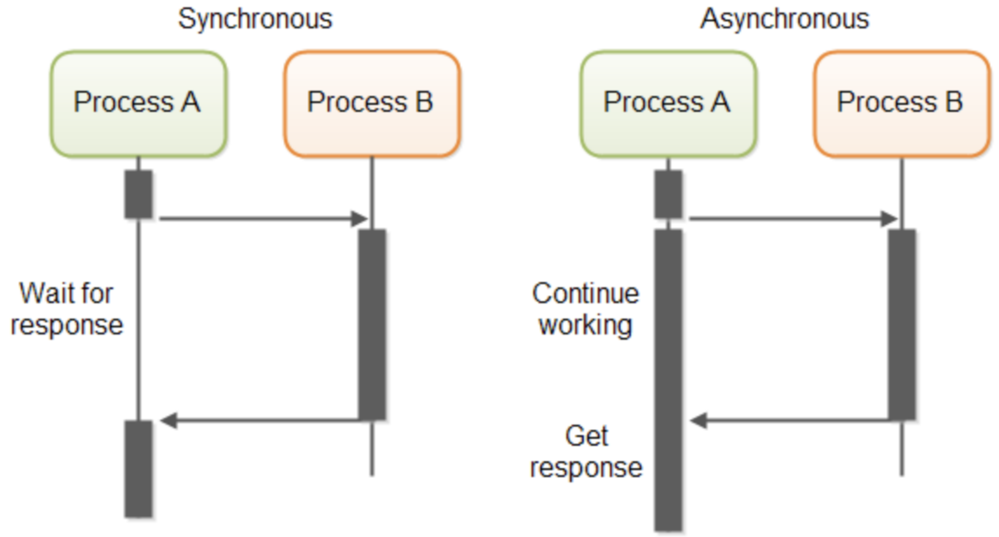
These are not formally, rigorously defined terms.

In Code: Composition and Asynchronous Method Invocation

- Composite operations can "take a while."
- "Holding open connections and waiting can be bad."
  - Consumes resources.
  - Connections can break.
  - … …
- There are two options:
  - Polling
  - Callbacks
- Our next pattern is composition and "polling."
  Create a new user requires:
  - Updating the user database.
  - Verifying a submitted address.
  - Creating an account in the catalog service.



In Code: Service Orchestration/Composition

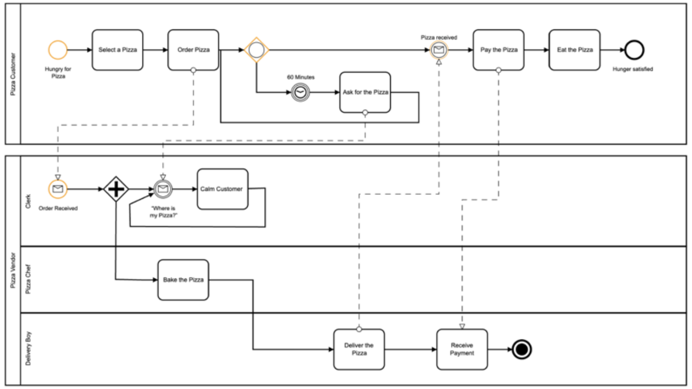


RESTFul HTTP calls can be implemented in both a synchronous and asynchronous fashion at an IO level.

- One call to a composite service
  - May drive calls to multiple other services.
  - Which in turn, may drive multiple calls to other services.
- Synchronous (Wait) and calling one at a time, in order is
  - Inefficient
  - Fragile
  - … …
- Asynchronous has benefits but can result in complex code.

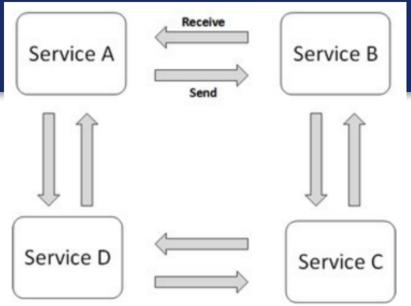
Service Orchestration

- Implementing a complex orchestration in code can become complex.
- High layer abstractions and tools have emerged, e.g. BPMN.
  - "Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model." expand in new window

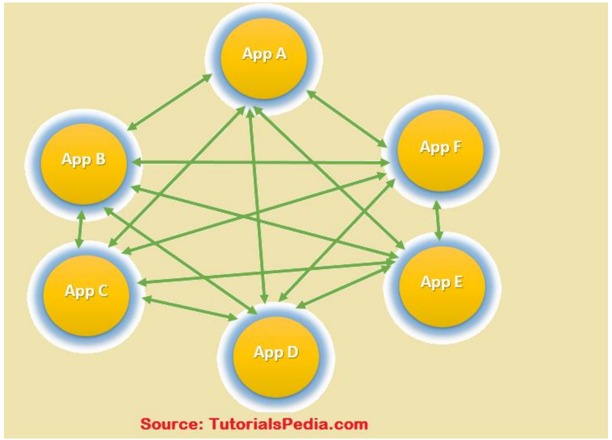


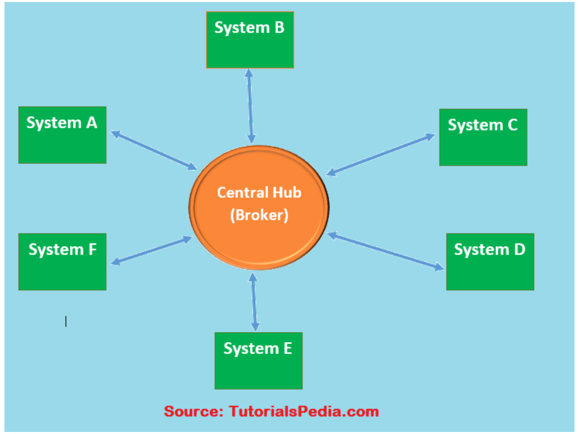
  - There are products for defining BPMN processes, generating code, executing, … …, e.g. Camunda: https://camunda.com/bpmn/
- There are other languages, tools, execution engines, … …

Choreography



- We saw the basic diagram, but …
  this is an anti-pattern and does not scale.

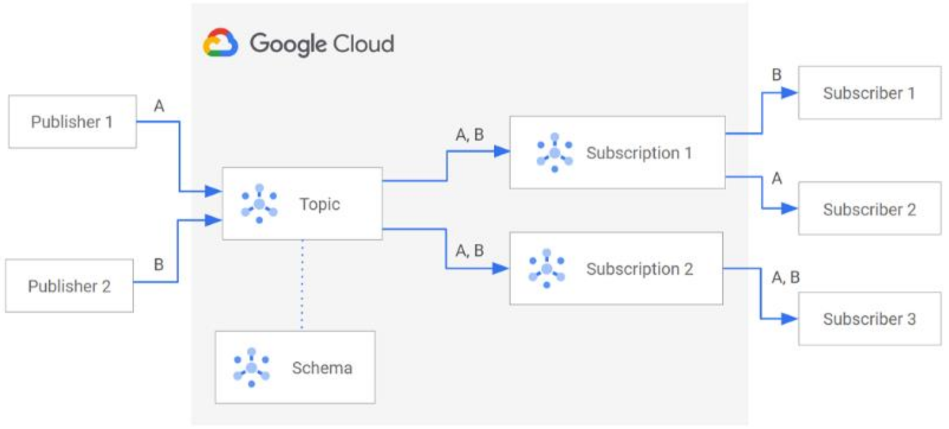


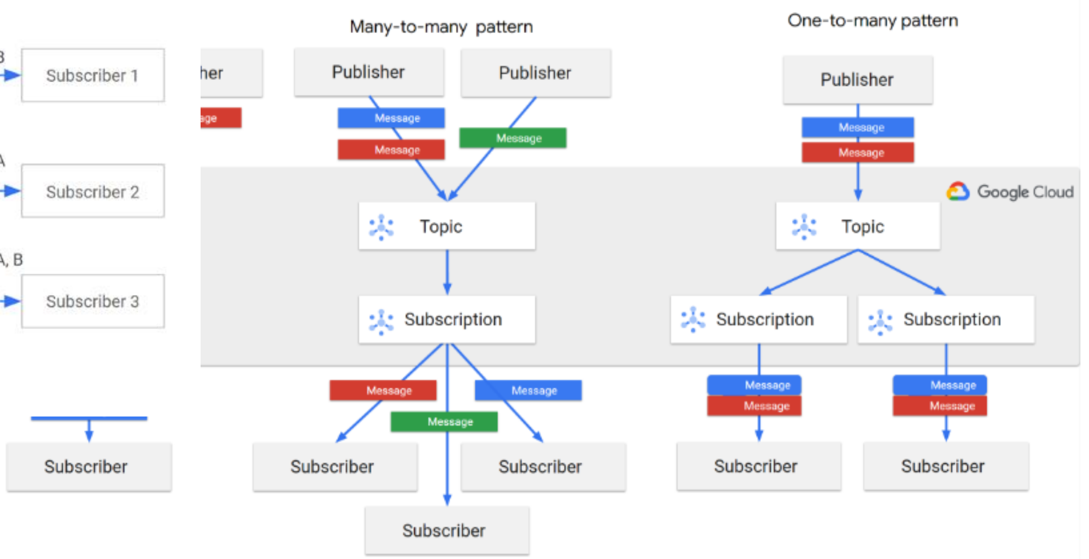


- But, how do you write the event driven microservices?
  - Well, you can just write code … …
  - Or use a state machines abstraction.
  - … …

© Donald F. Ferguson, 2024

Google Pub Sub





The following are the components of a Pub/Sub service:

- **Publisher** (also called a producer): creates messages and sends (pu specified topic.
- **Message**: the data that moves through the service.
- **Topic**: a named entity that represents a feed of messages.
- **Schema**: a named entity that governs the data format of a Pub/Sub message.
- **Subscription**: a named entity that represents an interest in receiving messages on a particular topic.
- **Subscriber** (also called a consumer): receives messages on a specified subscription.

Show Examples

API Gateway

Terms

From Wikipedia

- "**API management** is the process of creating and publishing web application programming interfaces (APIs), enforcing their usage policies, controlling access, nurturing the subscriber community, collecting and analyzing usage statistics, and reporting on performance. API Management components provide mechanisms and tools to support developer and subscriber communities."
- **Gateway:** A server that acts as an API front-end, receives API requests, enforces throttling and security policies, passes requests to the back-end service and then passes the response back to the requester. A gateway often includes a transformation engine to orchestrate and modify the requests and responses on the fly. A gateway can also provide functions such as collecting analytics data and providing caching. The gateway can provide the functionality to support authentication, authorization, security, audit and regulatory compliance. Gateways can be implemented using technologies like Nginx or HAProxy.

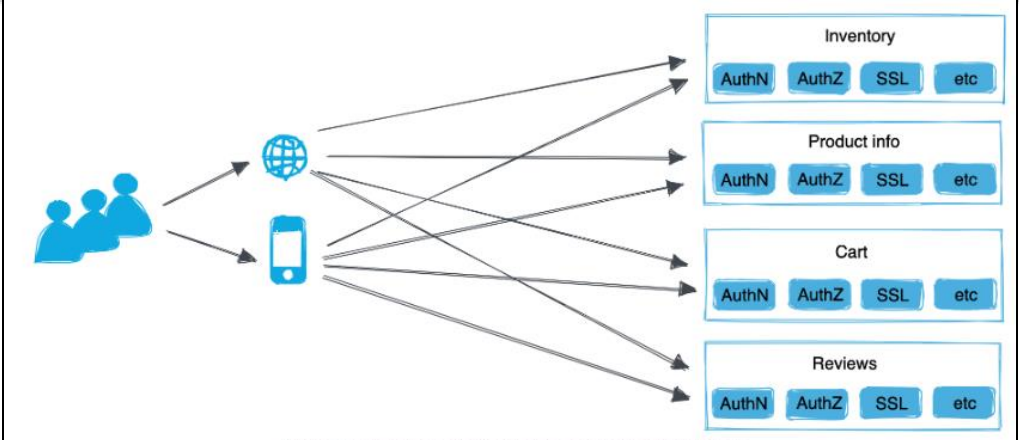


Figure: Client requests to microservices without an API Gateway.

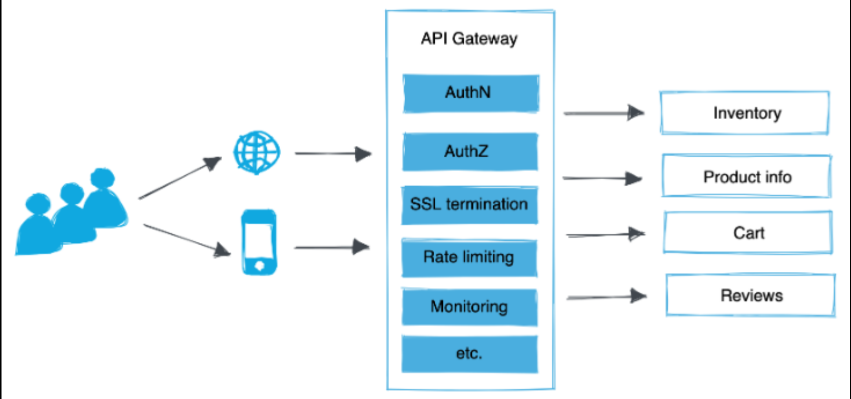
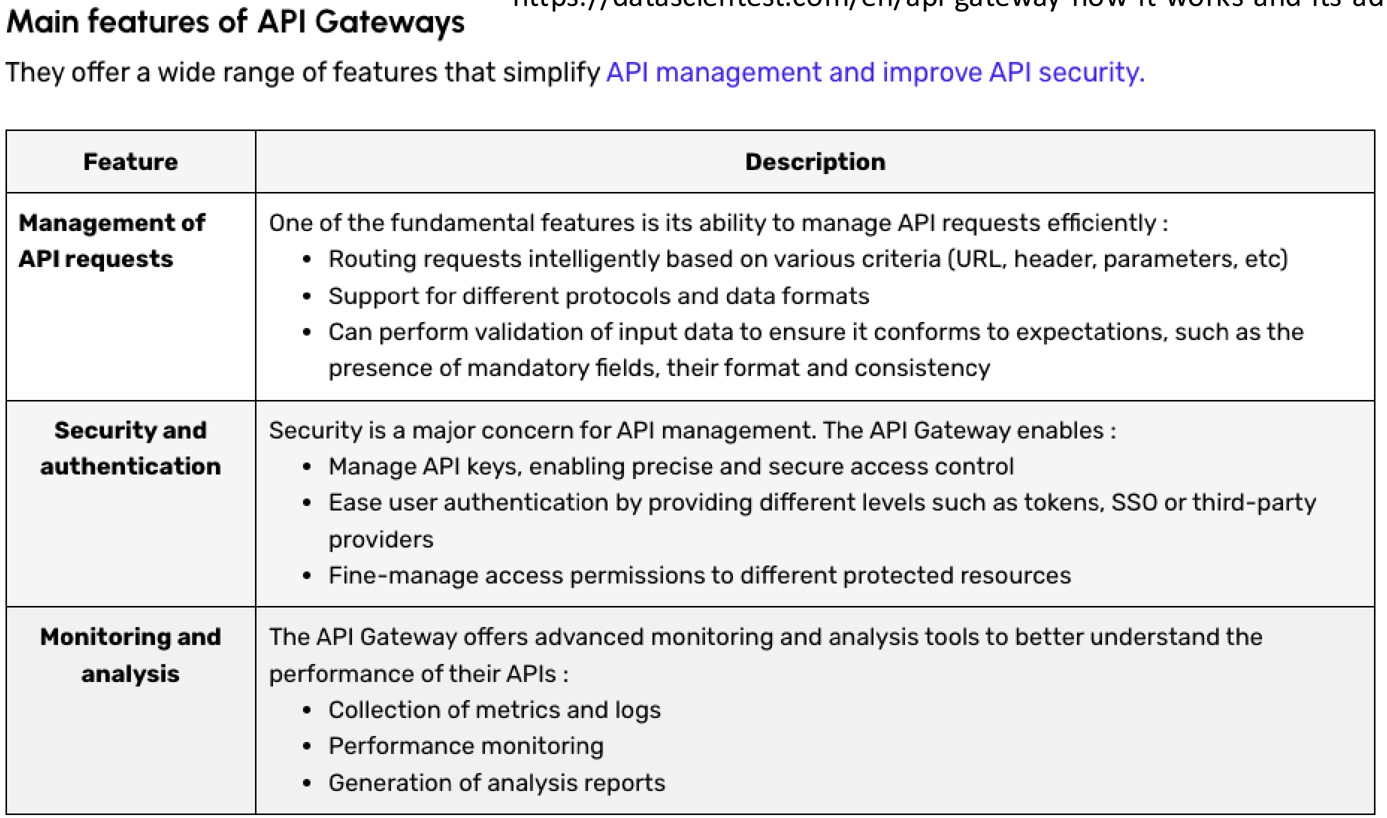


Figure: Client requests to microservices with an API Gateway.

- Solo.io is an example (this is not a recommendation): https://docs.solo.io/gloo-mesh-gateway/main/concepts/about/api-gateway/
- There are many, many API GWs
  - All major cloud service providers have one or more API gateways.
  - There are several open-source API GWs.
  - There are several commercial products.

API Gateway

https://datascientest.com/en/api-gateway-how-it-works-and-its-advantages

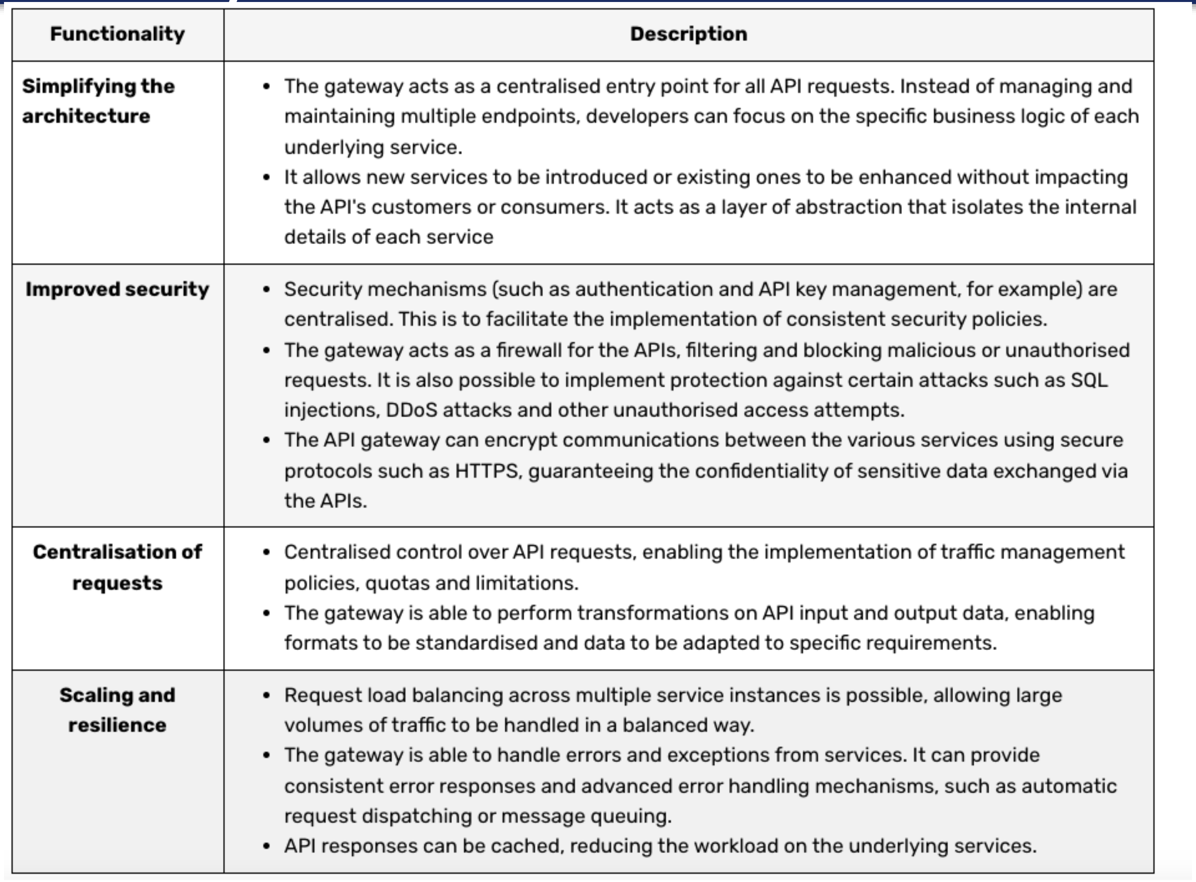


Main features of API Gateways

They offer a wide range of features that simplify API management and improve API security.

Feature	Description
Management of API requests	One of the fundamental features is its ability to manage API requests efficiently : • Routing requests intelligently based on various criteria (URL, header, parameters, etc) • Support for different protocols and data formats • Can perform validation of input data to ensure it conforms to expectations, such as the presence of mandatory fields, their format and consistency
Security and authentication	Security is a major concern for API management. The API Gateway enables : • Manage API keys, enabling precise and secure access control • Ease user authentication by providing different levels such as tokens, SSO or third-party providers • Fine-manage access permissions to different protected resources
Monitoring and analysis	The API Gateway offers advanced monitoring and analysis tools to better understand the performance of their APIs : • Collection of metrics and logs • Performance monitoring • Generation of analysis reports

API Gateway Benefits

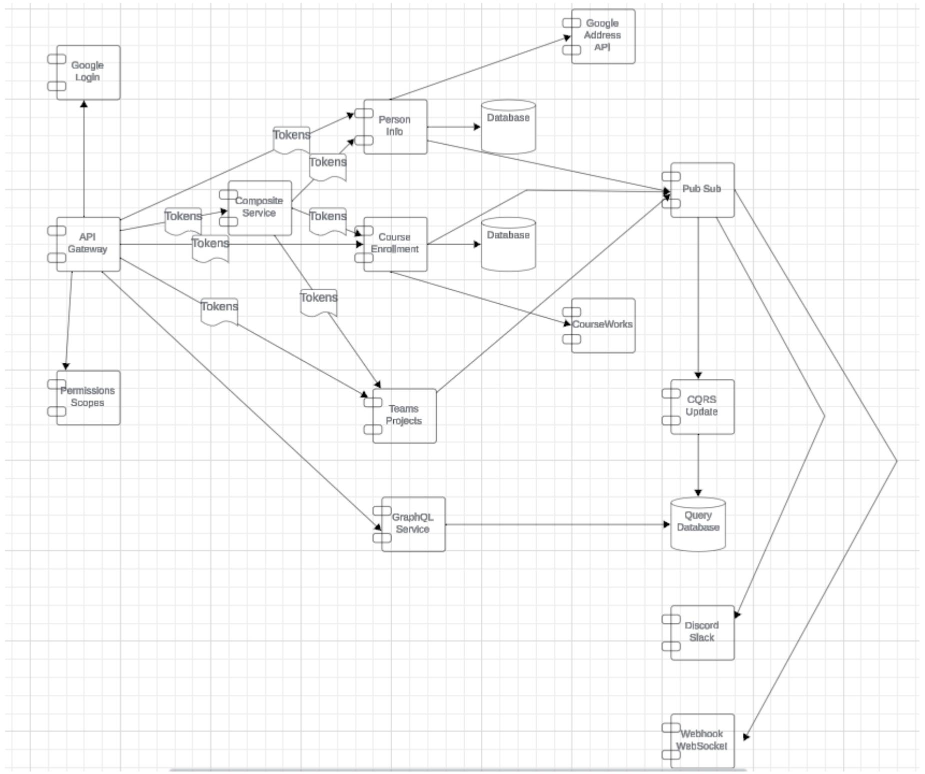


Functionality	Description
Simplifying the architecture	• The gateway acts as a centralised entry point for all API requests. Instead of managing and maintaining multiple endpoints, developers can focus on the specific business logic of each underlying service. • It allows new services to be introduced or existing ones to be enhanced without impacting the API's customers or consumers. It acts as a layer of abstraction that isolates the internal details of each service
Improved security	• Security mechanisms (such as authentication and API key management, for example) are centralised. This is to facilitate the implementation of consistent security policies. • The gateway acts as a firewall for the APIs, filtering and blocking malicious or unauthorised requests. It is also possible to implement protection against certain attacks such as SQL injections, DDoS attacks and other unauthorised access attempts. • The API gateway can encrypt communications between the various services using secure protocols such as HTTPS, guaranteeing the confidentiality of sensitive data exchanged via the APIs.
Centralisation of requests	• Centralised control over API requests, enabling the implementation of traffic management policies, quotas and limitations. • The gateway is able to perform transformations on API input and output data, enabling formats to be standardised and data to be adapted to specific requirements.
Scaling and resilience	• Request load balancing across multiple service instances is possible, allowing large volumes of traffic to be handled in a balanced way. • The gateway is able to handle errors and exceptions from services. It can provide consistent error responses and advanced error handling mechanisms, such as automatic request dispatching or message queuing. • API responses can be cached, reducing the workload on the underlying services.

Overall Architecture

Overall Architecture

- API Gateway
  - Enforces login
  - Generates observability scope
  - Updates tokens returned to client
- Authorization:
  - Permissions and scopes generates a signed JWT token with claims.
  - Each microservice authorizes request based on JWT, URL, … …
- All microservices:
  - Check authorization in middleware
  - Log before/after in middleware
  - Generate “update event” on PUT/POST/DELETE
- Pub/Sub for: CQRS, Discord/Slack, Webhook/Slack
- GraphQL



© Donald F. Ferguson, 2024 COLUMBIA | ENGINEERING The Fu Foundation School of Engineering and Applied Science