W4153 – Cloud Computing

Lecture 12:

Sample Project, EDA/MDP, API-GW, Advanced Composition, GraphQL, Kubernetes

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

Contents

Contents

- Semester roadmap and completion
  - Project presentation and demo schedule and format.
  - Next few lectures.
- Sample, illustrative project
  - Architecture “big picture.”
  - API gateway.
  - Tokens and context headers the simple version.
  - Some “internet stuff:” domains, DNS, certificates, … …
  - Advanced composition
- Project discussion:
  - Power Rangers, Application: Cloud Casino
  - CC Team, Project Idea: Food Rating App

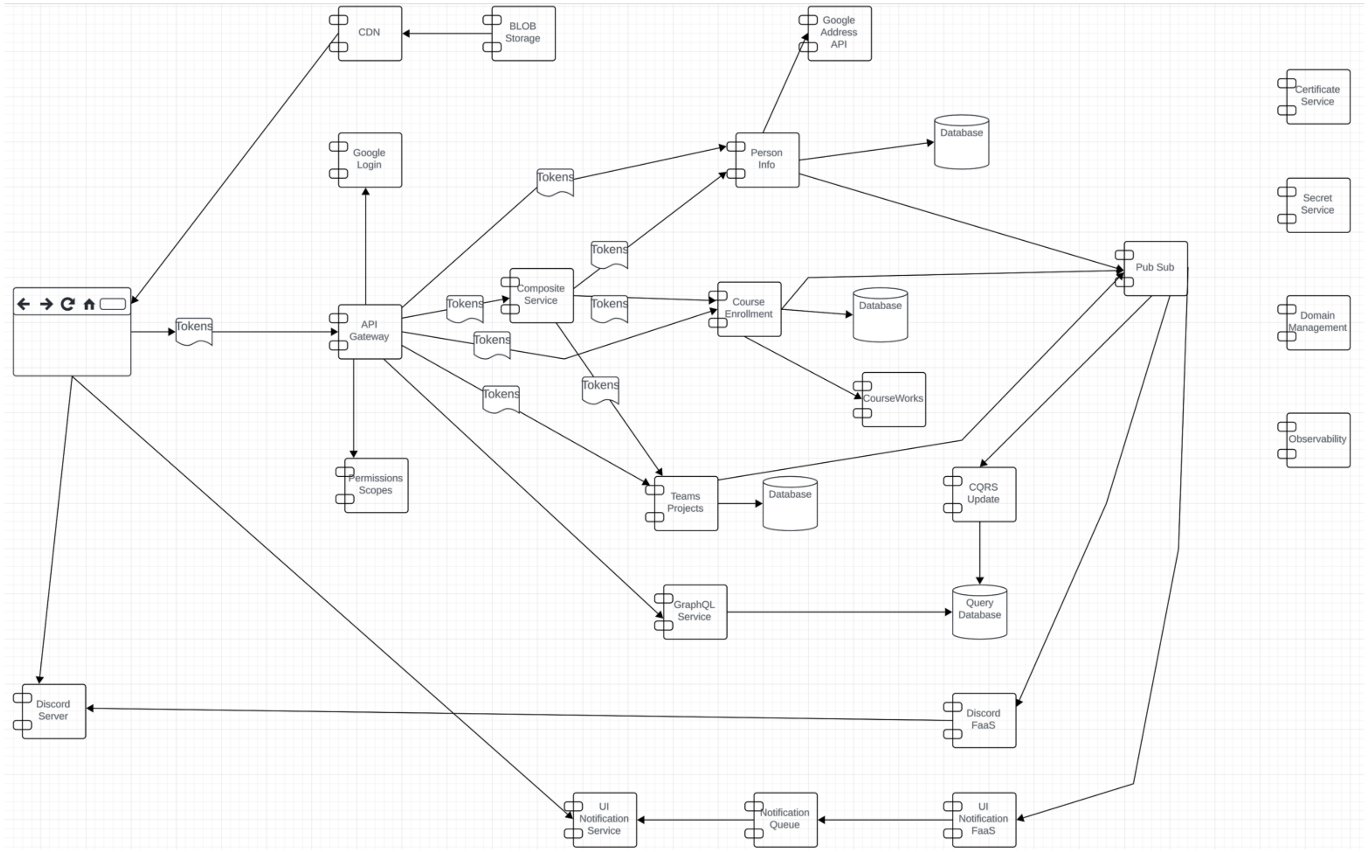
Semester Roadmap and Completion

Completion and Roadmap

- Lectures remaining:
  - Today introduces material
  - 22-NOV will introduce some additional concepts that you will include in your final project.
  - 29-NOV is a holiday.
  - 06-DEC will cover some additional, interesting topics and *may* introduce one or two very simple additional requirements.
- Final project:

Overall Architecture

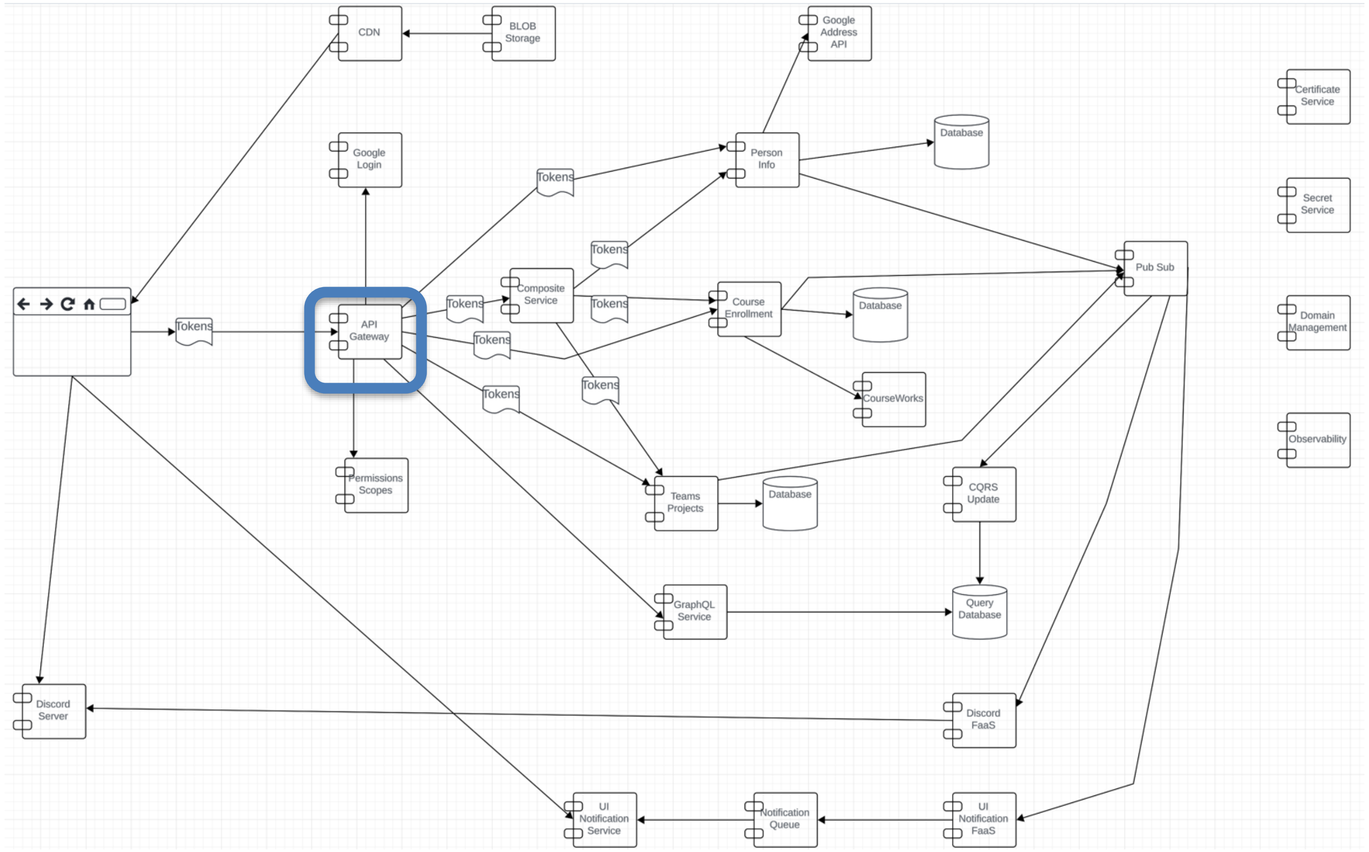
Overall Architecture



API Gateway

Overview and Concepts

API Gateway


CDN
BLOB Storage
Google Address API
Certificate Service
Google Login
Tokens
Person Info
Database
Secret Service
Tokens
Pub Sub
Composite Service
Tokens
Tokens
Course Enrollment
Database
Domain Management
Tokens
API Gateway
Tokens
Tokens
Tokens
CourseWorks
Observability
Permissions Scopes
Teams Projects
Database
CQRS Update
GraphQL Service
Query Database
Discord Server
Discord FaaS
UI Notification Service
Notification Queue
UI Notification FaaS

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.

API Gateway

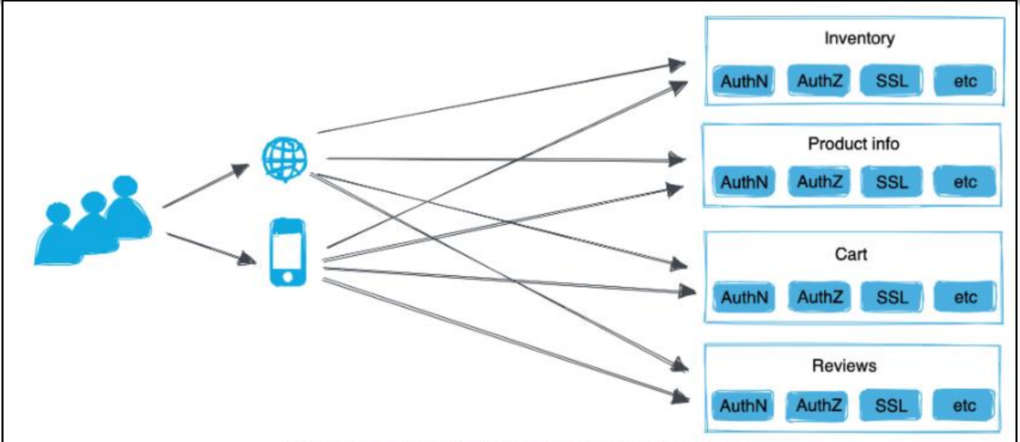


Figure: Client requests to microservices without an API Gateway.

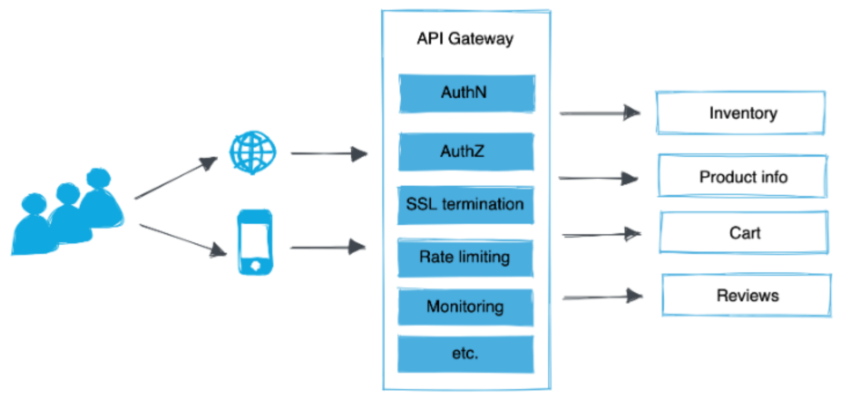
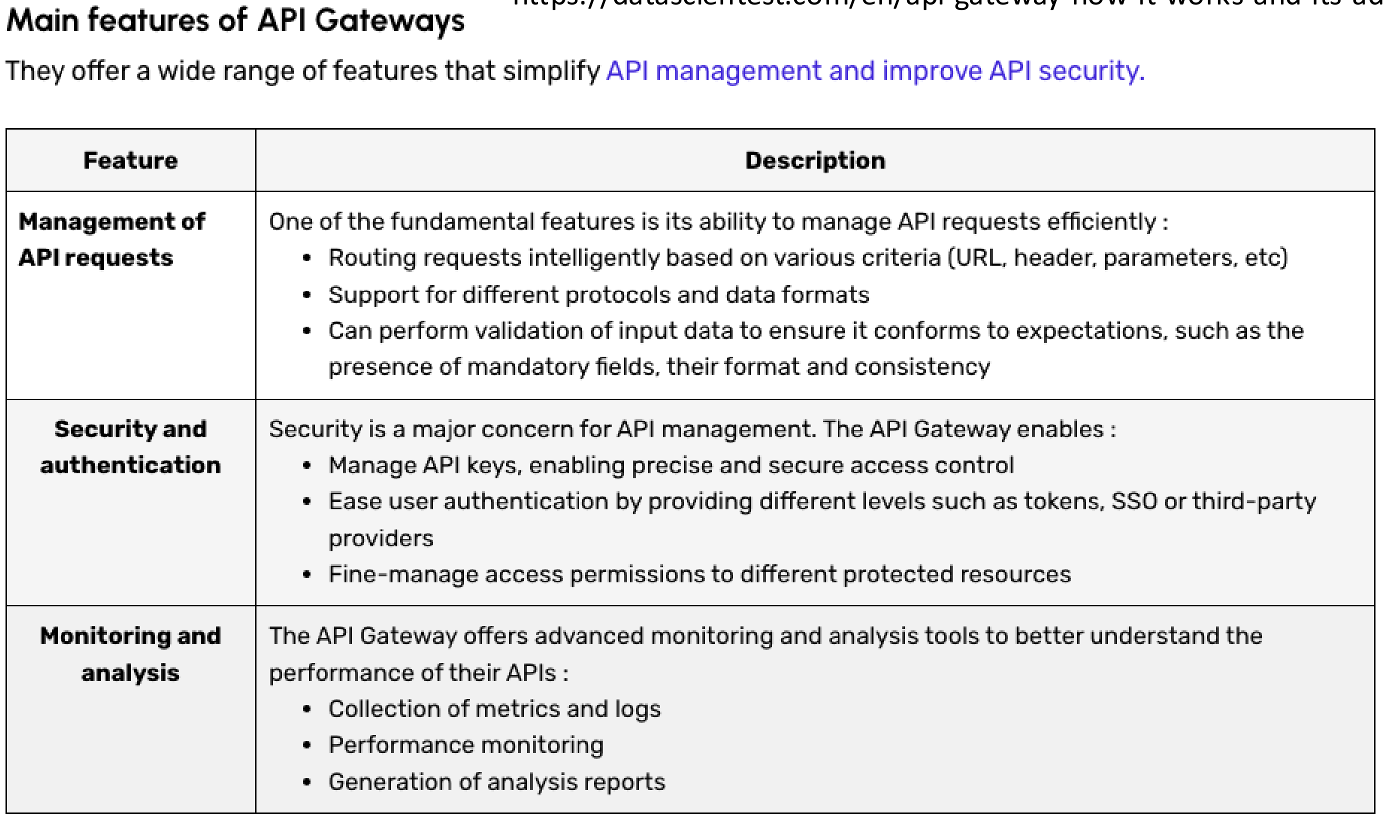


Figure: Client requests to microservices with an API Gateway.

- 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

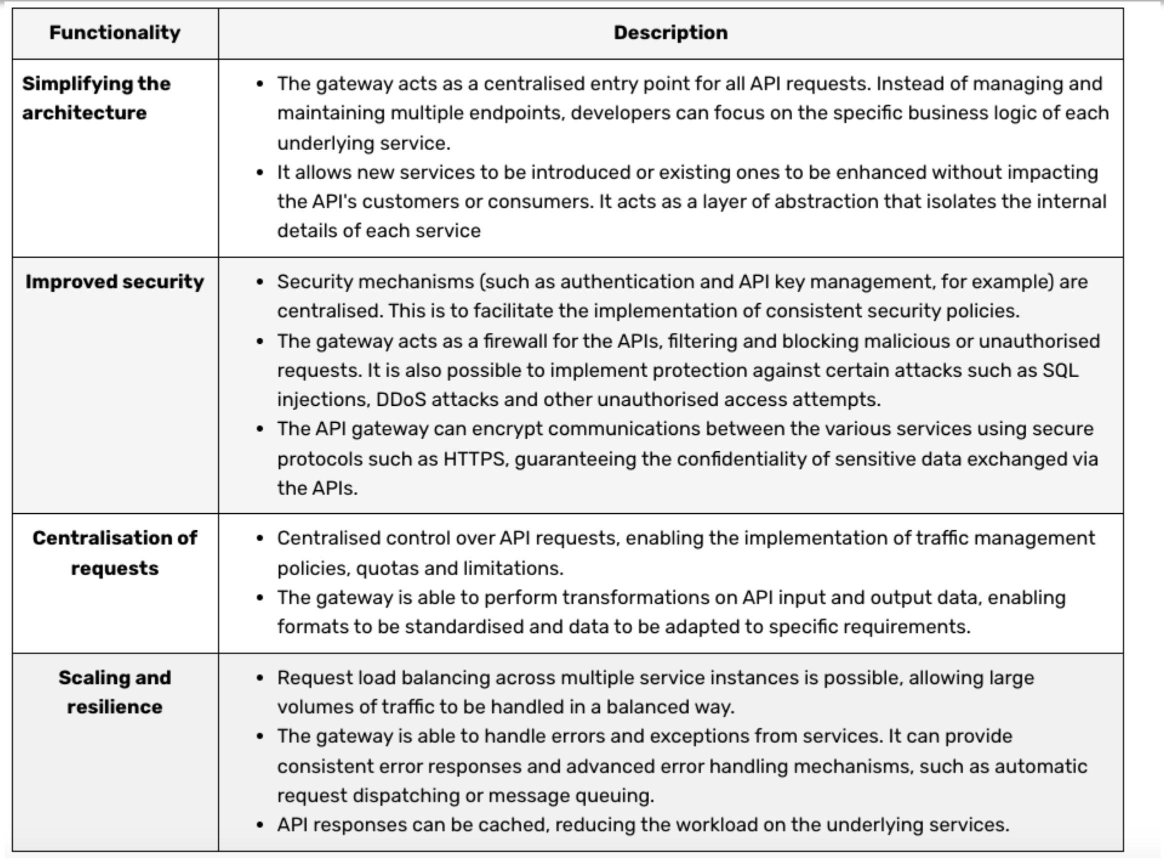


Main features of API Gateways

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.

Google API Gateway

API Gateway is an API management system that provides management, monitoring, and authentication for your APIs. The components that make up API Gateway include:

- **API Gateway**: for managing all aspects of a deployed API
- **Service Control**: for applying API management rules
- **Service Management**: for managing API configurations
- **gcloud CLI**: for deploying and managing your APIs
- **Google Cloud console**: for logging, monitoring and sharing

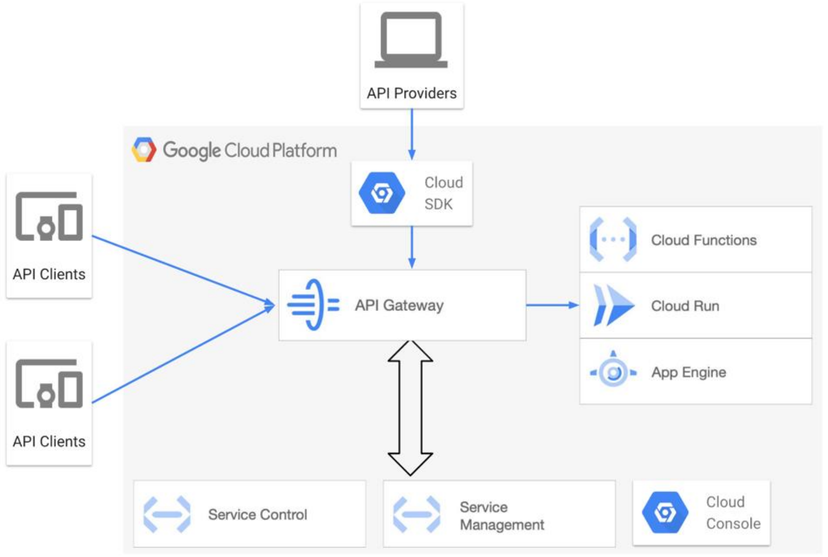
Request routing

When a request is received:

1. API Gateway creates a trace token for Cloud Trace.
2. API Gateway matches the path of the incoming requests with the target API. After finding a matching route, API Gateway performs any authentication steps for the specified API.
3. If JWT validation is necessary, API Gateway validates the authentication using the appropriate public key for the signer, and validates the audience field in the JWT. If an API key is required, API Gateway calls the Service Control API to validate the key.
4. Service Control looks up the key to validate it, and ensures that the project associated with the key has enabled the API. If the key isn't valid or the project hasn't enabled the API, the call is rejected and it is logged via the Service Control API.
5. If Service Control successfully validates the key, the request along with all original headers, plus a JWT validation header, if appropriate, is forwarded to the backend.
6. When a response is received from the backend, API Gateway returns the response to the caller and sends the final timing information to Trace. The call points are logged by the Service Control API, which then writes metrics and logs to their appropriate destinations.

Architecture

Below is a high level diagram of the major components involved in API Gateway:



API Gateway Summary

- You will have to set up an API gateway for two or three of your microservices.

- You can use either AWS or GCP.

- The minimal functionality expected is:
  - A single, logical URL space with routing.
  - Some form of authentication/authorization checking (but can be very simple).
  - Basic logging and trace.

- You do not need to deeply or completely integrate with end-to-end trace, observability, single sign on/login, etc.

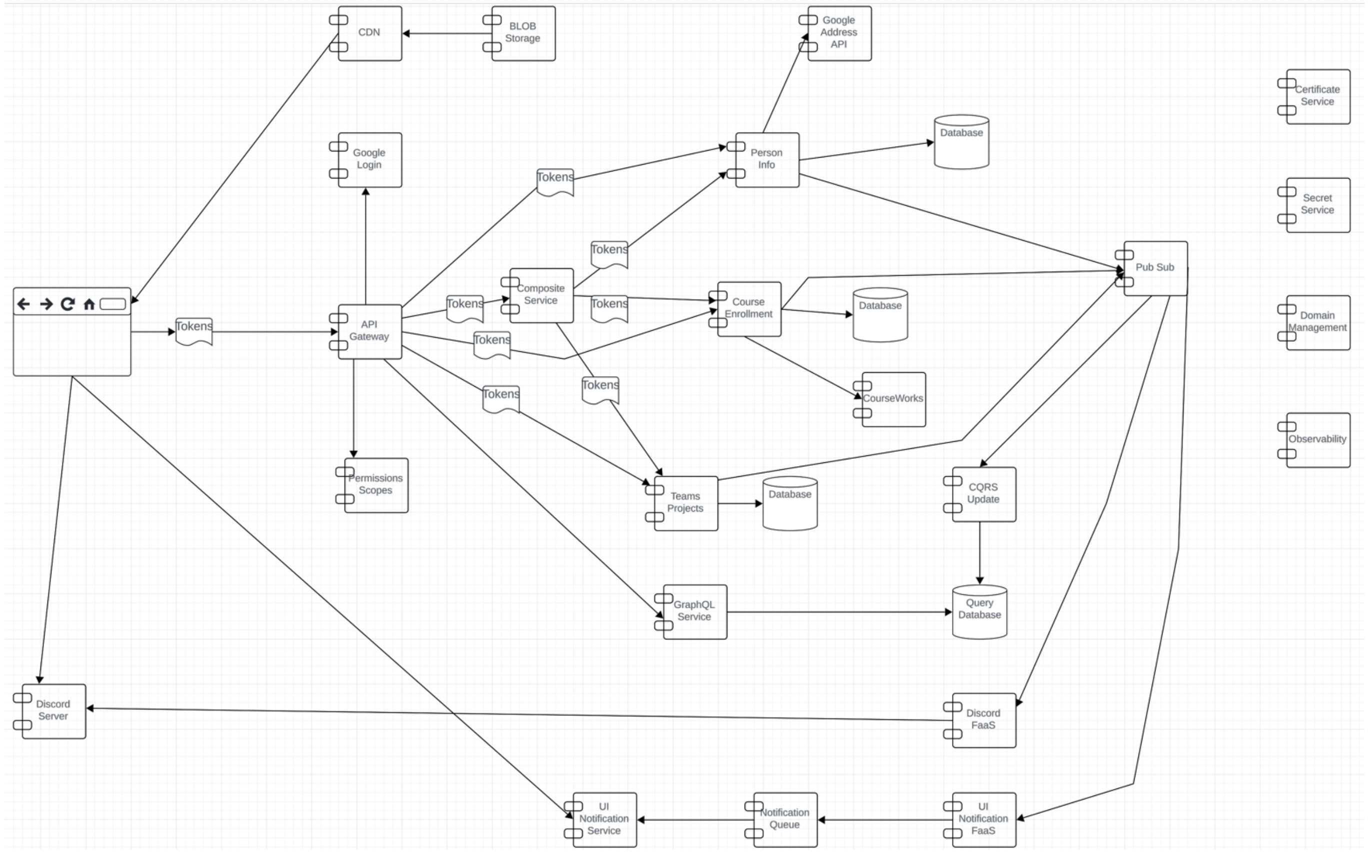
Tokens

Scope

Propagation

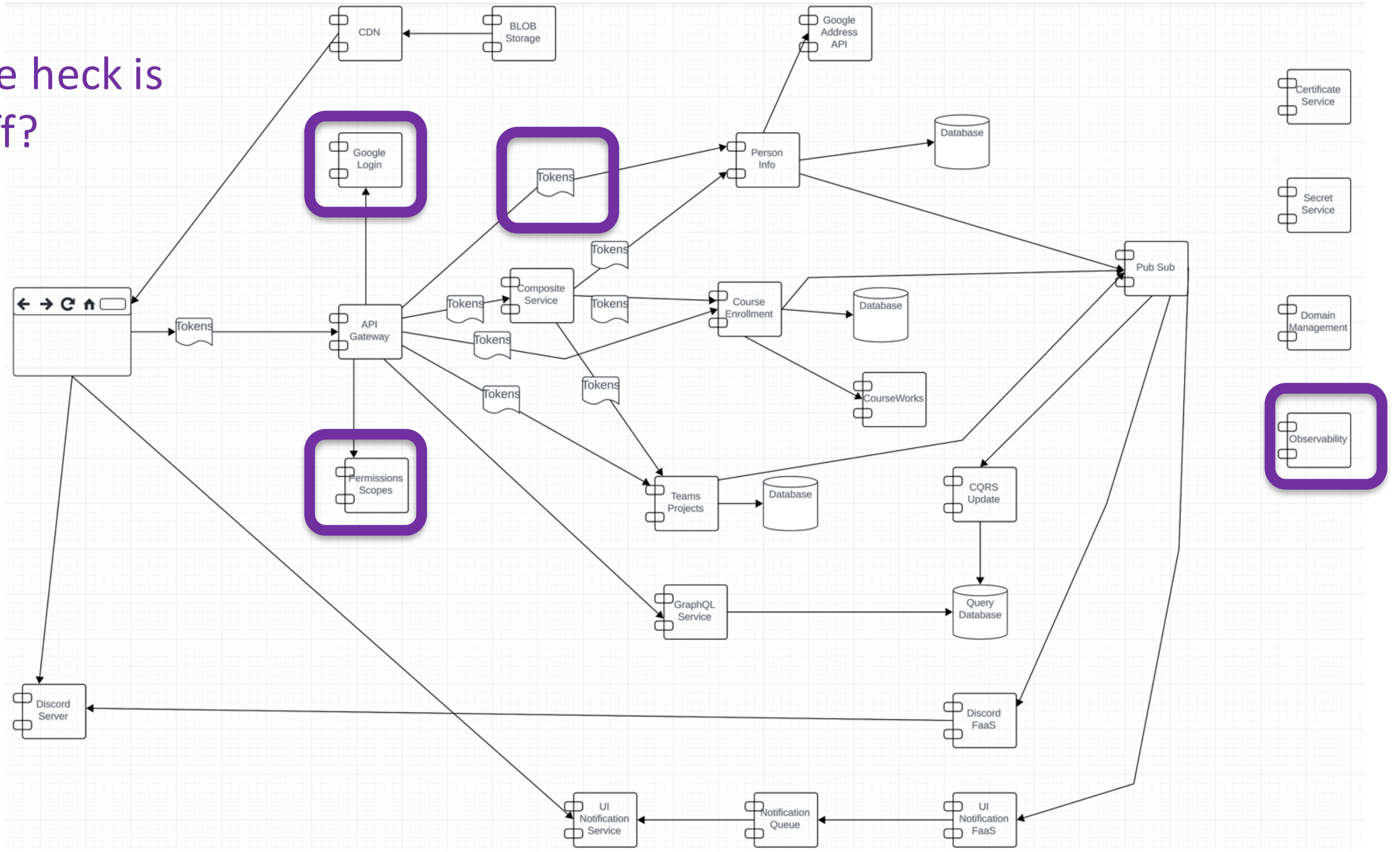
Observability

API Gateway

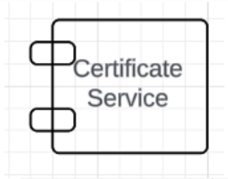

CDN
BLOB Storage
Google Address API
Certificate Service
Google Login
Tokens
Person Info
Database
Secret Service
Tokens
Pub Sub
Composite Service
Tokens
Course Enrollment
Database
Domain Management
Tokens
Tokens
API Gateway
Tokens
Tokens
Tokens
CourseWorks
Observability
Permissions Scopes
Teams Projects
Database
CQRS Update
GraphQL Service
Query Database
Discord Server
Discord FaaS
UI Notification Service
Notification Queue
UI Notification FaaS

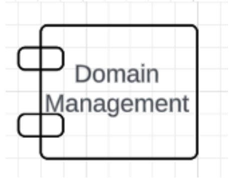
API Gateway

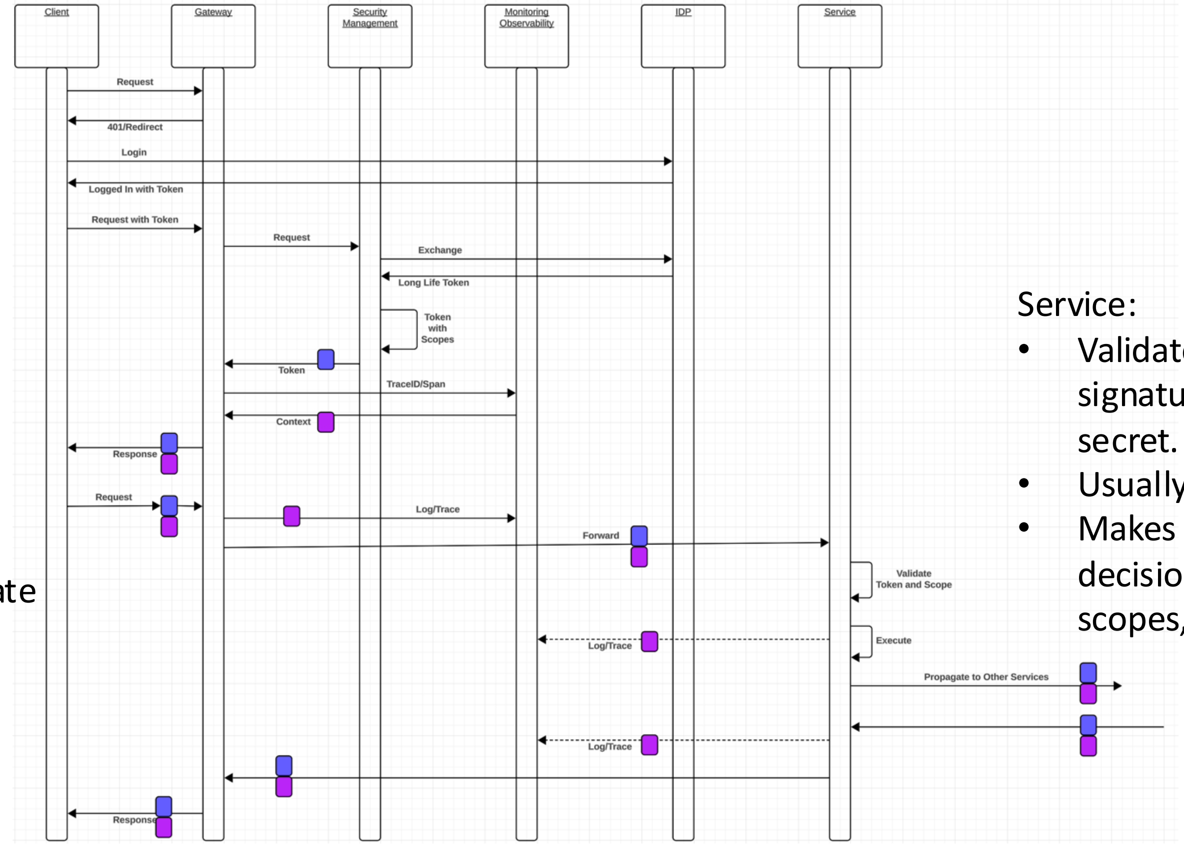
What the heck is that stuff?

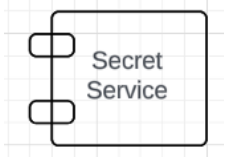


Conceptual Model









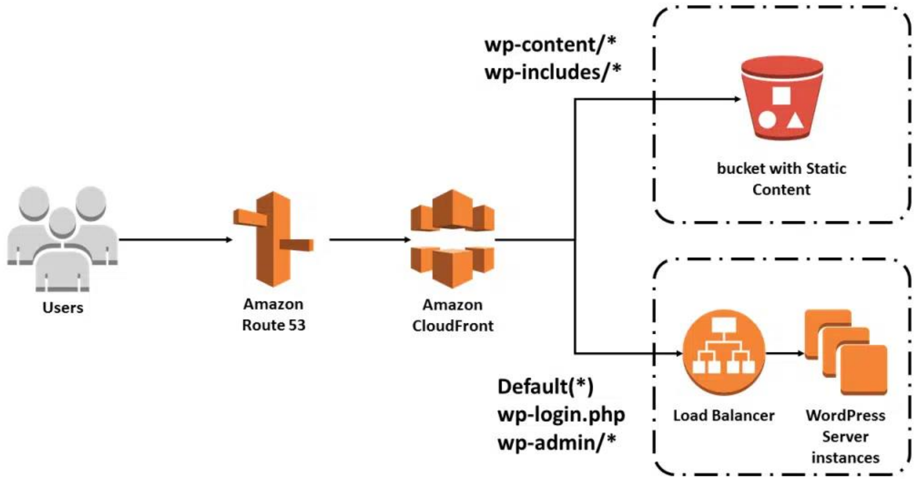
- Browser relies on certificate for trusting the site/gateway.
- Configure the certificate in the gateway, etc.
- Browser maintains JWT tokens in session or local storage.

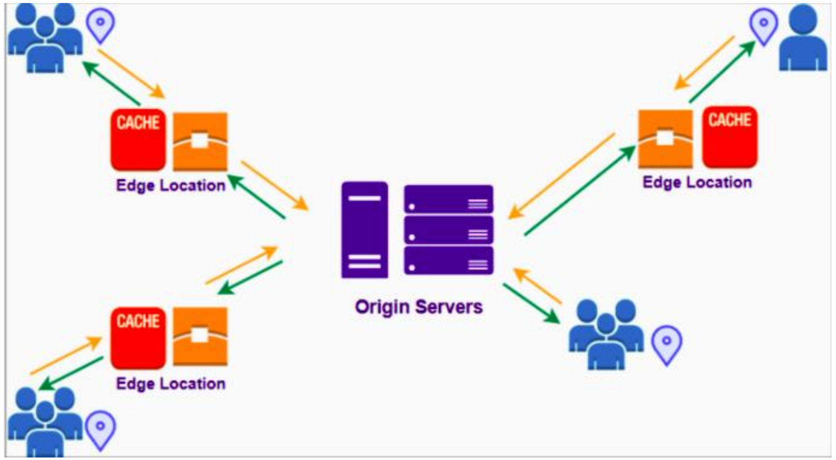
Service:

- Validates token using signature and shared secret.
- Usually in middleware.
- Makes authorization decision based on claims, scopes, etc.

Some Network Stuff

CloudFront Example

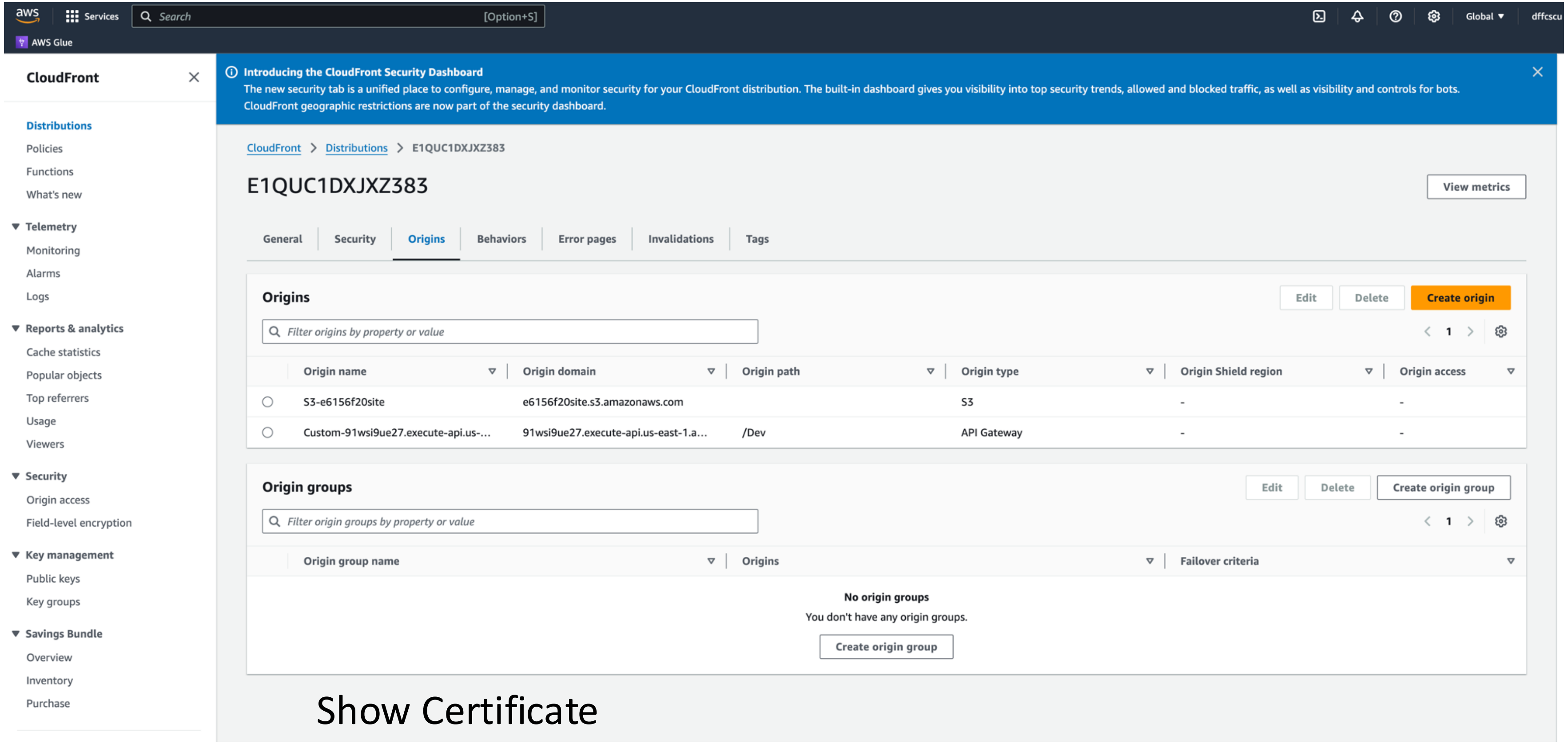




- Show the following on AWS:
  - CloudFront
  - Route 53
  - AWS Certificate
  - S3 Buckets

- What do you have to do?
- Do not worry about DNS, certificates, etc. You can usually use an icky one that comes with the CSP.
- You should serve content from blob storage.

CloudFront



Introducing the CloudFront Security Dashboard

The new security tab is a unified place to configure, manage, and monitor security for your CloudFront distribution. The built-in dashboard gives you visibility into top security trends, allowed and blocked traffic, as well as visibility and controls for bots. CloudFront geographic restrictions are now part of the security dashboard.

CloudFront > Distributions > E1QUC1DXJXZ383

E1QUC1DXJXZ383

Origins

Origin name	Origin domain	Origin path	Origin type	Origin Shield region	Origin access
S3-e6156f20site	e6156f20site.s3.amazonaws.com		S3	-	-
Custom-91wsi9ue27.execute-api.us-...	91wsi9ue27.execute-api.us-east-1.a...	/Dev	API Gateway	-	-

Origin groups

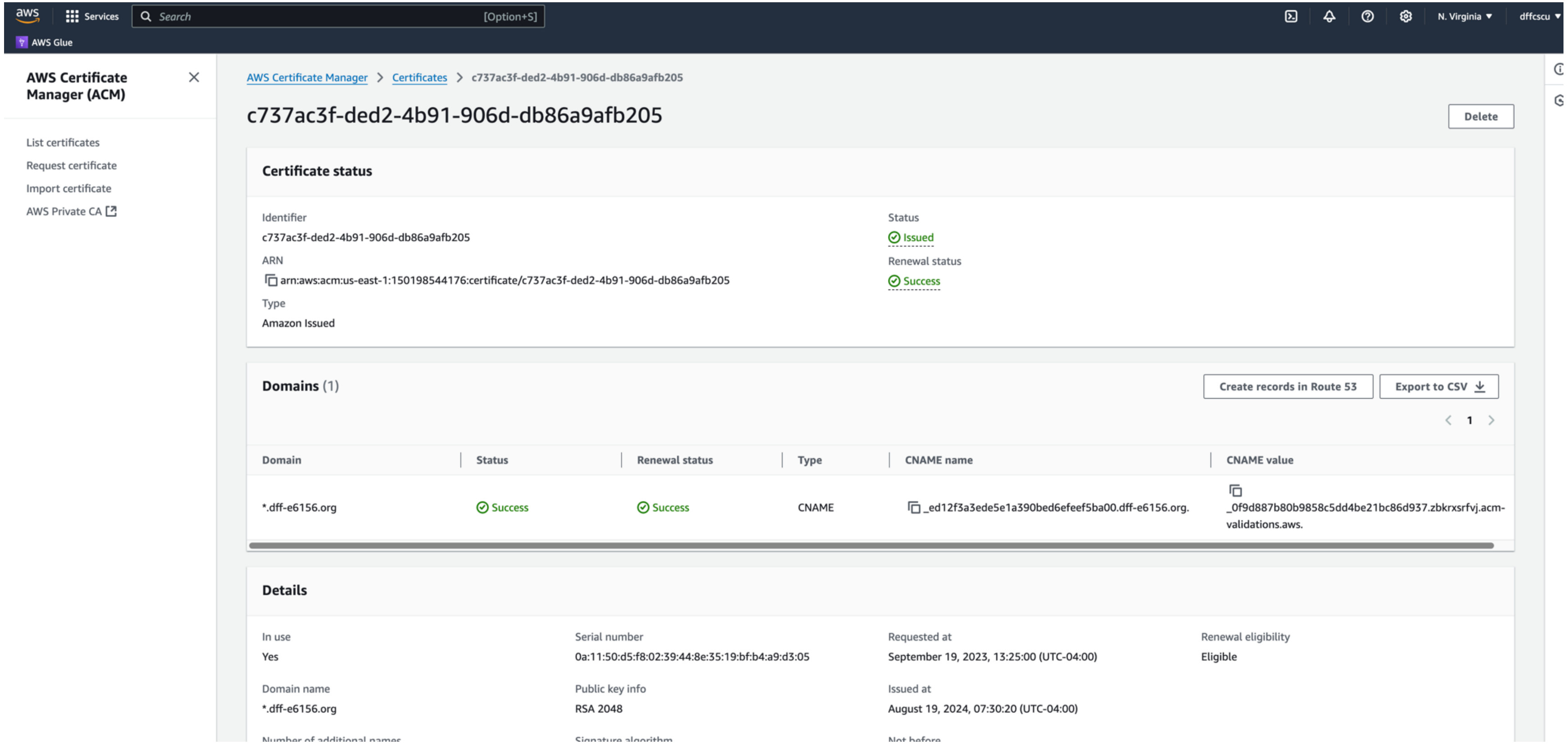
Origin group name	Origins	Failover criteria

No origin groups

You don't have any origin groups.

Show Certificate

Certificate



AWS Certificate Manager (ACM)

- List certificates
- Request certificate
- Import certificate
- AWS Private CA

AWS Certificate Manager > Certificates > c737ac3f-ded2-4b91-906d-db86a9afb205

c737ac3f-ded2-4b91-906d-db86a9afb205

Delete

Certificate status

Identifier
c737ac3f-ded2-4b91-906d-db86a9afb205

ARN
arn:aws:acm:us-east-1:150198544176:certificate/c737ac3f-ded2-4b91-906d-db86a9afb205

Type
Amazon Issued

Status
Issued

Renewal status
Success

Domains (1)

Create records in Route 53 | Export to CSV

Domain	Status	Renewal status	Type	CNAME name	CNAME value
*.dff-e6156.org	Success	Success	CNAME	_ed12f3a3ede5e1a390bed6efeef5ba00.dff-e6156.org.	_0f9d887b80b9858c5dd4be21bc86d937.zbkrxsrfvj.acm-validations.aws.

Details

In use
Yes

Serial number
0a:11:50:d5:f8:02:39:44:8e:35:19:bf:b4:a9:d3:05

Requested at
September 19, 2023, 13:25:00 (UTC-04:00)

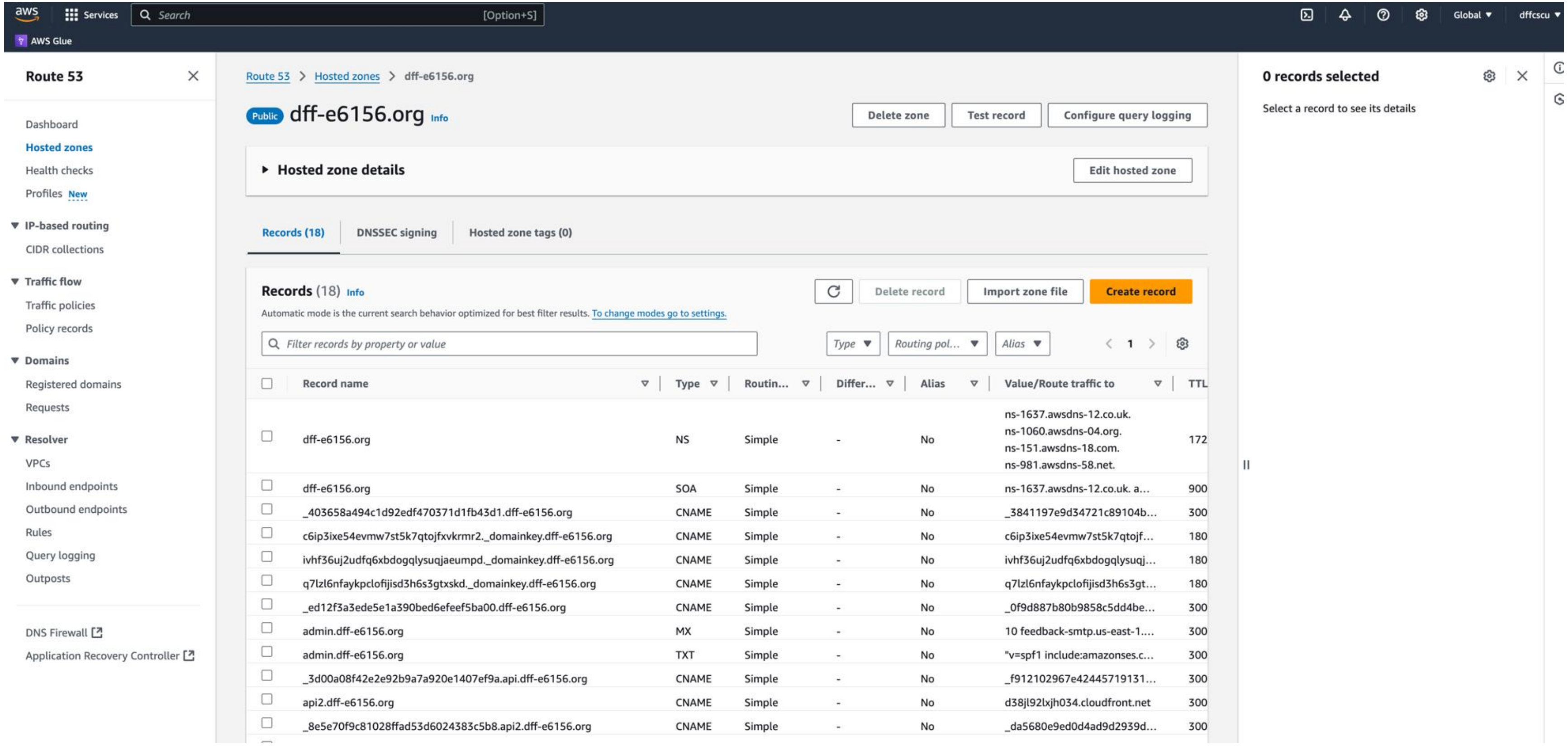
Renewal eligibility
Eligible

Domain name
*.dff-e6156.org

Public key info
RSA 2048

Issued at
August 19, 2024, 07:30:20 (UTC-04:00)

Domain



Route 53 > Hosted zones > dff-e6156.org

Public dff-e6156.org Info

Hosted zone details

Records (18) | DNSSEC signing | Hosted zone tags (0)

Records (18) Info

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Record name	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL
dff-e6156.org	NS	Simple	-	No	ns-1637.awsdns-12.co.uk. ns-1060.awsdns-04.org. ns-151.awsdns-18.com. ns-981.awsdns-58.net.	172
dff-e6156.org	SOA	Simple	-	No	ns-1637.awsdns-12.co.uk. a...	900
_403658a494c1d92edf470371d1fb43d1.dff-e6156.org	CNAME	Simple	-	No	_3841197e9d34721c89104b...	300
c6ip3ixe54evmw7st5k7qtojfxvkrmr2._domainkey.dff-e6156.org	CNAME	Simple	-	No	c6ip3ixe54evmw7st5k7qtojf...	180
ivhf36uj2udfq6xbdogqlysuqjaeumpd._domainkey.dff-e6156.org	CNAME	Simple	-	No	ivhf36uj2udfq6xbdogqlysuqj...	180
q7lzl6nfaykpclofijisd3h6s3gtxskd._domainkey.dff-e6156.org	CNAME	Simple	-	No	q7lzl6nfaykpclofijisd3h6s3gt...	180
_ed12f3a3ede5e1a390bed6efeef5ba00.dff-e6156.org	CNAME	Simple	-	No	_0f9d887b80b9858c5dd4be...	300
admin.dff-e6156.org	MX	Simple	-	No	10 feedback-smtp.us-east-1....	300
admin.dff-e6156.org	TXT	Simple	-	No	"v=spf1 include:amazonses.c...	300
_3d00a08f42e2e92b9a7a920e1407ef9a.api.dff-e6156.org	CNAME	Simple	-	No	_f912102967e42445719131...	300
api2.dff-e6156.org	CNAME	Simple	-	No	d38jl92lxjh034.cloudfront.net	300
_8e5e70f9c81028ffad53d6024383c5b8.api2.dff-e6156.org	CNAME	Simple	-	No	_da5680e9ed0d4ad9d2939d...	300

0 records selected

Select a record to see its details

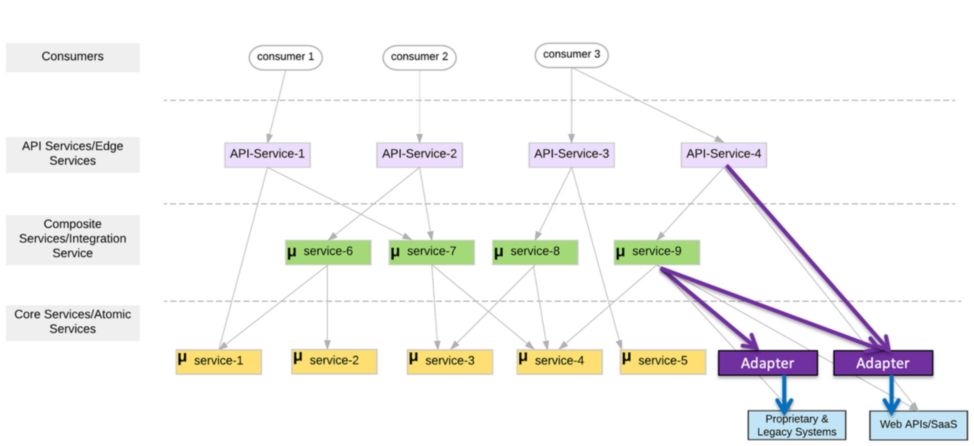
Some URLs

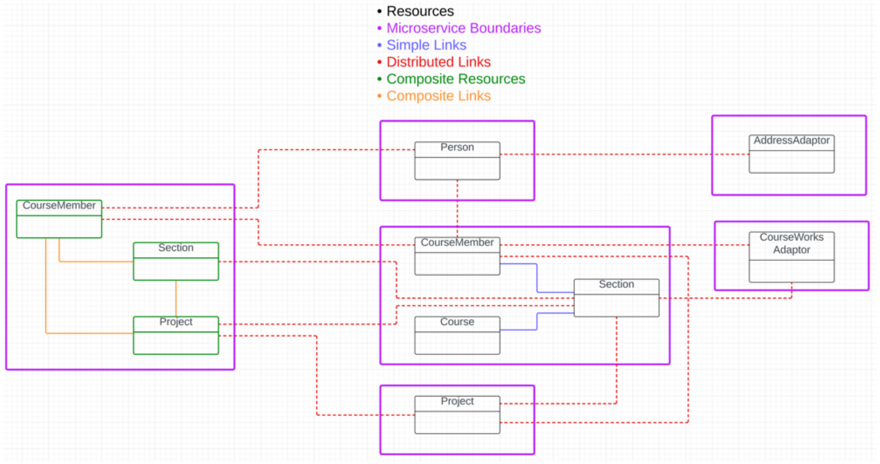
- Some URLs
  - https://e6156f20site.s3.us-east-2.amazonaws.com/index.html
  - https://d1kc4b5azlamci.cloudfront.net/index.html
  - https://www.dff-e6156.org/index.html
- Implications for your project:
  - You should use a "BLOB storage service" for your "web frontend content."
    There are countless articles about the pros and cons of various approaches, e.g. https://medium.com/dowebthings/improve-your-page-speed-by-hosting-your-static-website-with-aws-s3-and-cloudfront-d0e881812092
  - Getting and managing a domain, DNS entries, etc. is a pain in the neck.
  - Figuring out how to integrate with custom certificates adds lower back pain.
  - Do not worry about domains and certificates. Most CSPs will generate ugly, icky certificates and domain names for BLOB stores, microservice endpoints, … …

Advanced Service Composition

Concepts
Reminder and Overview

Composition Concepts and Reminder



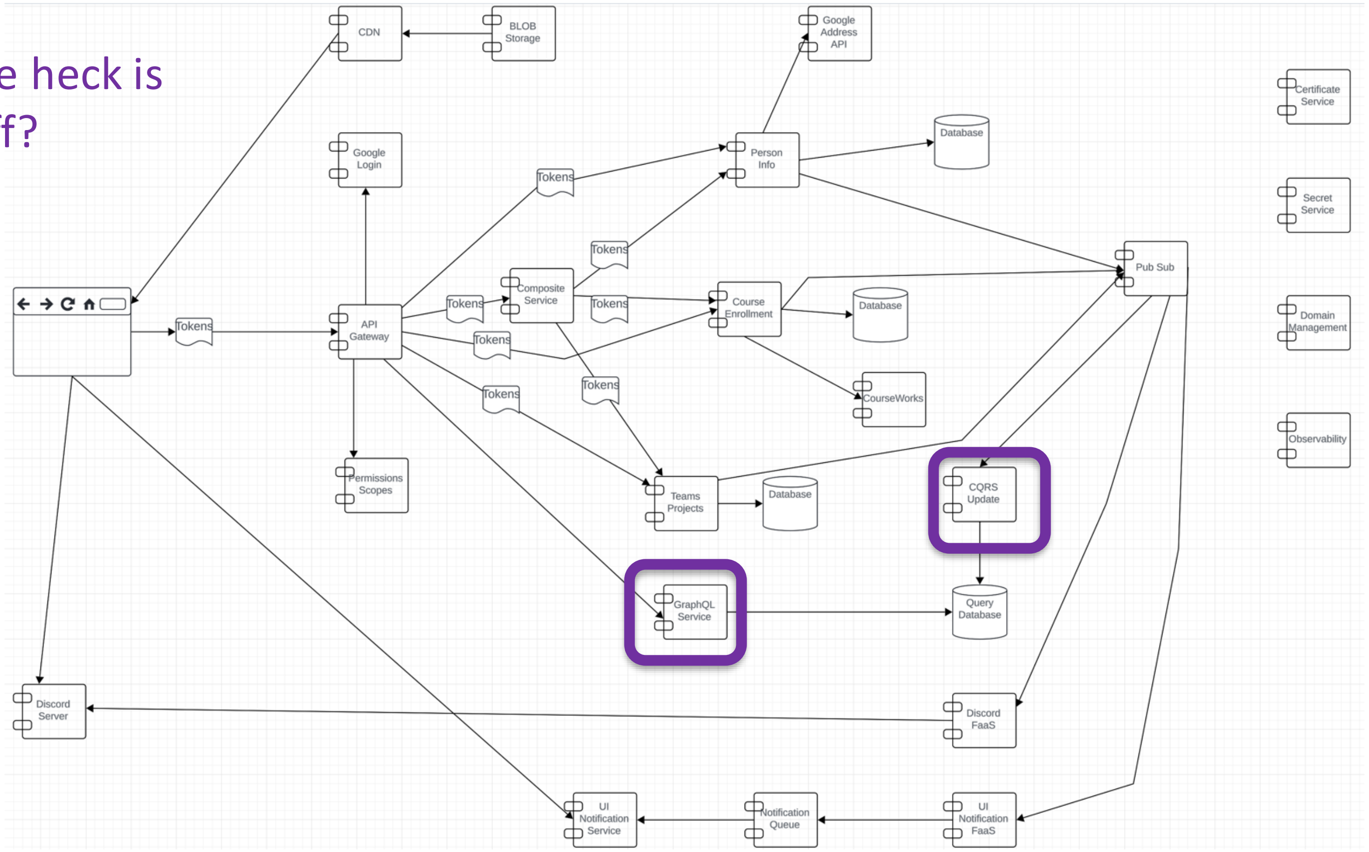


- There are two aspects to “composition:”
  - Structure: How do I “link things together” to “find services” that a microservice needs?
  - Behavior: How do I implement the application of logic of the composite?
    - Two models: *Orchestration* and *Choreography.*
    - Several approaches to implementing the service methods: synchronous code, asynchronous code, event driven architecture, message driven architecture, state machines/workflows, … …
    - Use pattern like CQRS to “optimize” queries?

CQRS
and
GraphQL

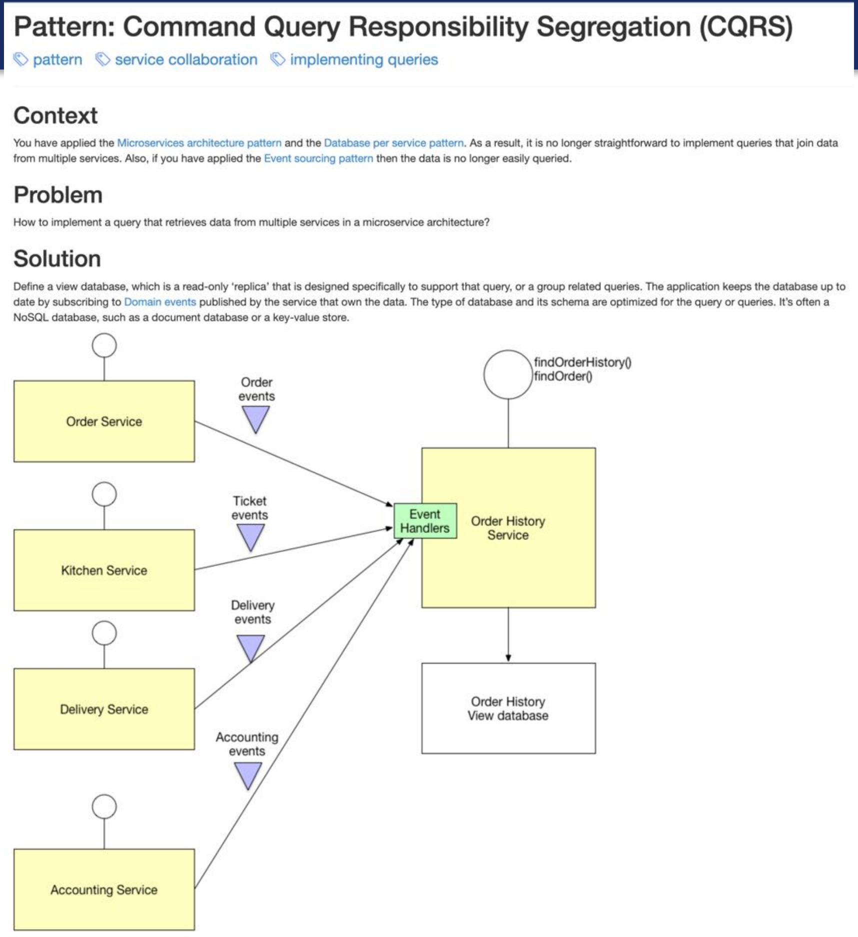
CQRS and GraphQL

What the heck is that stuff?



CQRS

- We have seen that microservices leads to little bits of “data” all over the place.
- Something that a user thinks of as a “thing” is scattered all over the place.
- We have seen how to handle this issue using various patterns for composition.
- But,
  - Most of the time things are not changing.
  - The vast majority of requests are reads.
  - I would like something more view like and with better query support than just query string in URLs.


Pattern: Command Query Responsibility Segregation (CQRS)

pattern service collaboration implementing queries

Context

You have applied the Microservices architecture pattern and the Database per service pattern. As a result, it is no longer straightforward to implement queries that join data from multiple services. Also, if you have applied the Event sourcing pattern then the data is no longer easily queried.

Problem

How to implement a query that retrieves data from multiple services in a microservice architecture?

Solution

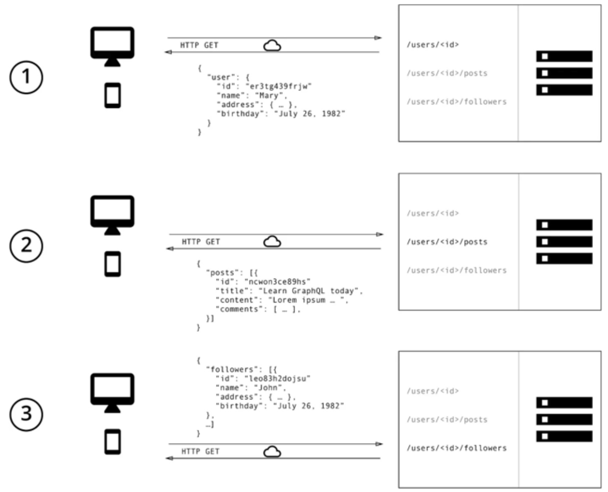
Define a view database, which is a read-only 'replica' that is designed specifically to support that query, or a group related queries. The application keeps the database up to date by subscribing to Domain events published by the service that own the data. The type of database and its schema are optimized for the query or queries. It's often a NoSQL database, such as a document database or a key-value store.

REST vs GraphQL in a Nutshell

Data fetching with REST vs GraphQL

With a REST API, we'll probably gather data through multiple endpoints. These endpoints could be

- **/users/<id>** — to fetch the initial users data
- **/users/<id>/posts** — to fetch all the posts of the user
- **/users/<id>/followers** — to fetch list of followers of the user



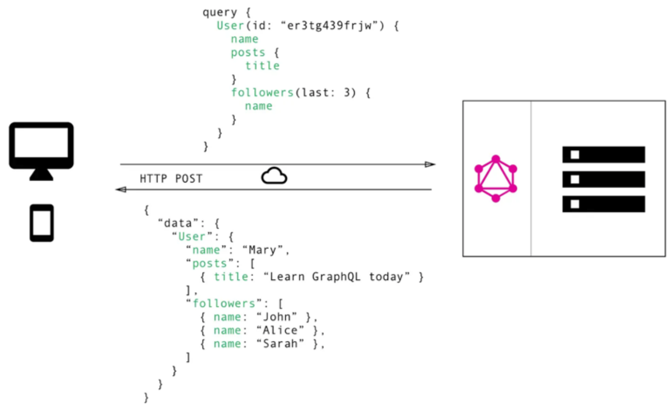
```
{
  "user": {
    "id": "er3tg439frjw"
    "name": "Mary",
    "address": { … },
    "birthday": "July 26, 1982"
  }
}
```

```
{
  "posts": [{
    "id": "ncwon3ce89hs"
    "title": "Learn GraphQL today",
    "content": "Lorem ipsum … ",
    "comments": [ … ],
  }]
}
```

```
{
  "followers": [{
    "id": "leo83h2dojsu"
    "name": "John",
    "address": { … },
    "birthday": "July 26, 1982"
  },
  …]
}
```

With REST, we will have to make three requests to different endpoints to get the required data. Also, there will be a problem of overfetching as excess data will be received than what is needed. So the data has to be formatted on the client-side before it can be rendered.

In GraphQL, a single query will be sent to the GraphQL server that specifies the exact data requirements. The server sends back the response with the desired data.



```
query {
  User(id: "er3tg439frjw") {
    name
    posts {
      title
    }
    followers(last: 3) {
      name
    }
  }
}
```

```
{
  "data": {
    "User": {
      "name": "Mary",
      "posts": [
        { title: "Learn GraphQL today" }
      ],
      "followers": [
        { name: "John" },
        { name: "Alice" },
        { name: "Sarah" },
      ]
    }
  }
}
```

Client can specify exactly the data it needs in the query. And the server responds in the exact format as the nested query.

GraphQL Concepte

Here are some core concepts in GraphQL:

- Schema: GraphQL uses schemas to define the types of operations or queries that can be performed. The schema acts as an interface between the client and the server to fetch or modify data.
- Types: GraphQL, like REST, has various data types in an API. These include String, Int, Boolean, or custom object types that the developer defines. Each field of a GraphQL type has its own type. For example, in your GraphQL, the book_name field of the Book type is a String, and the publish_date field is an Int. Is that clear?
- Queries: GraphQL is a query language that lets clients specify the shape and content of the data they want from the server. Clients can use GraphQL to request specific fields and nested data in their queries, and the server will respond with data that matches the same structure as the queries. This enables clients to fetch only the data they need, avoiding unnecessary or excessive requests to the server.
- Mutations: Mutations are a way of modifying data with GraphQL. Unlike queries, which are mainly for retrieving data from the server, mutations are for creating, updating, or deleting data on the server. Mutations ensure that the data written to the server is predictable.
- Subscriptions: Subscriptions in GraphQL enable real-time updates such as notifications. Clients can use subscriptions to listen for specific events or actions on the server. When those events or actions occur, the server sends notifications to the subscribed clients.
- Resolvers: A resolver is a function that connects a schema field to a data source, where it can fetch or modify data according to the query. Resolvers are the bridge between the schema and the data, and they handle the logic for data manipulation and retrieval.

GraphQL

- Show the GraphQL Overviews
  - REST vs. GraphQL: A Detailed Comparison of API Architectures for Developers
  - GraphQL: An Introduction
  - FastAPI with GraphQL: Delving Deep into Modern API Development
  - Lectures/W4153-2024F-12-Sample-Project-EDA:MDP-AdvancedCompotion-APIGW-GraphQL-Kubernetes/generalpresentation1-180129061029.pdf

- How this comes together:
  - A script (or scripts) creates the query schema and database, and loads the data.
  - Microservices emit a resource change event via middleware to a topic, e.g. “person_change_event.”
  - A FaaS instance reacts to the event and updates “the copy.”
  - Simple, very simple, GraphQL service for query. You only need to do resolvers.
  - Links in query enable update of the underlying data through the composites.

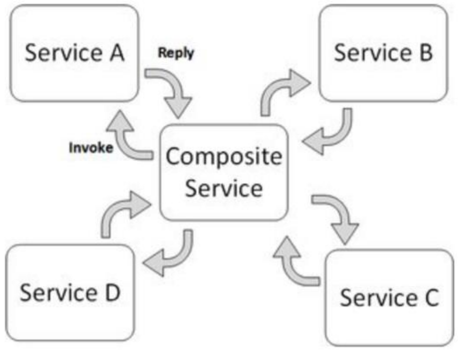
Updates and Sagas

Concepts

Service orchestration

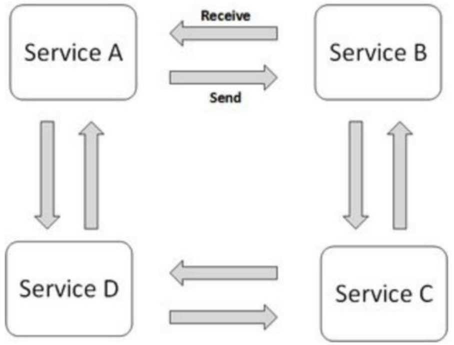
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

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

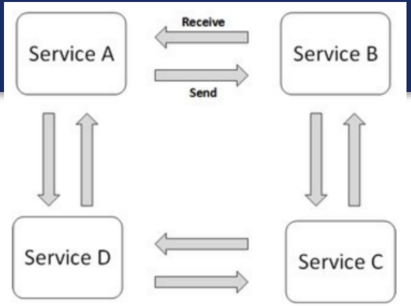
Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.



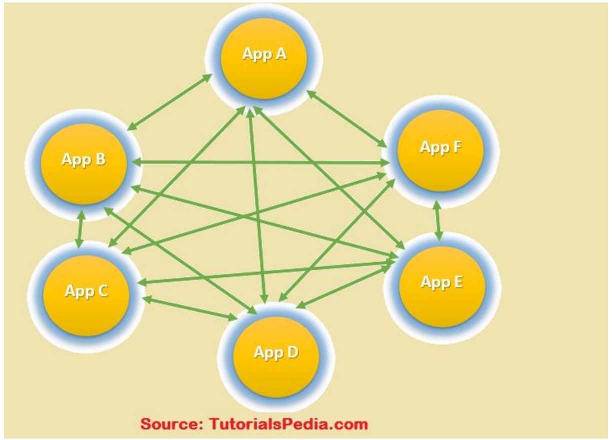
The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved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

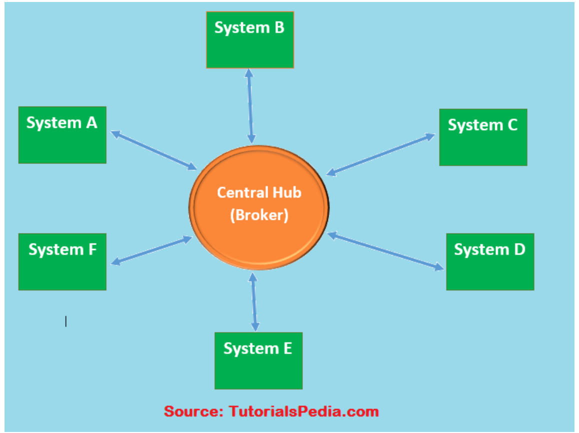
These are not formally, rigorously defined terms.

Choreography



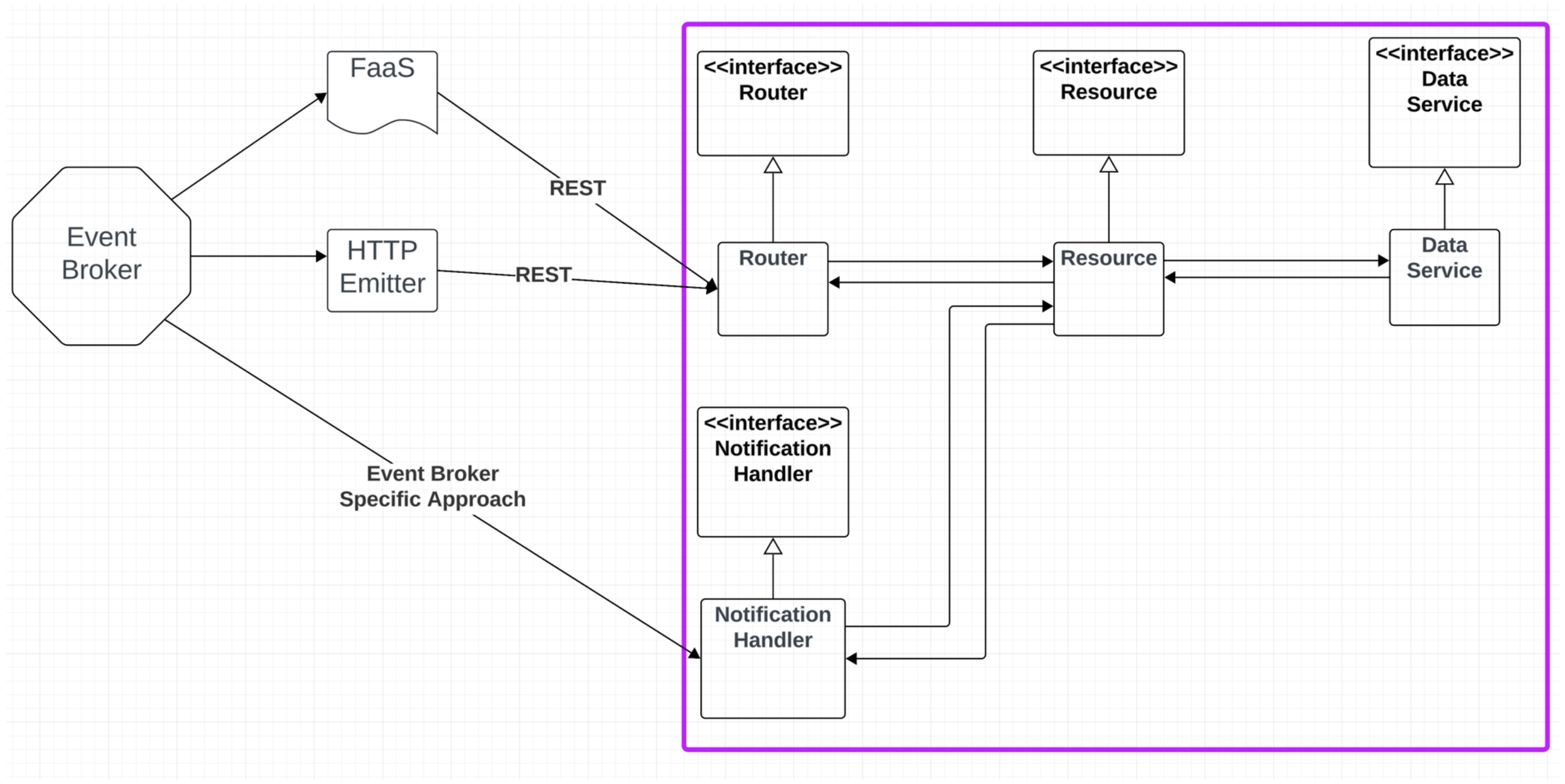
- We saw the basic diagram, but …
  this is an anti-pattern and does not scale.



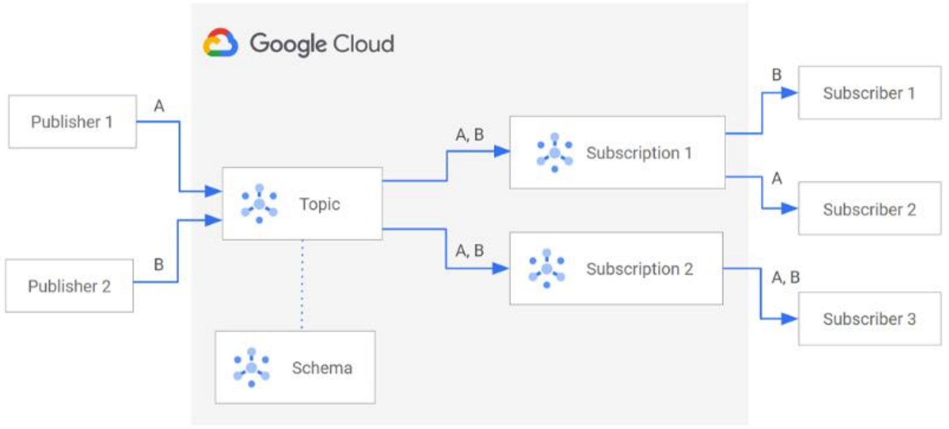


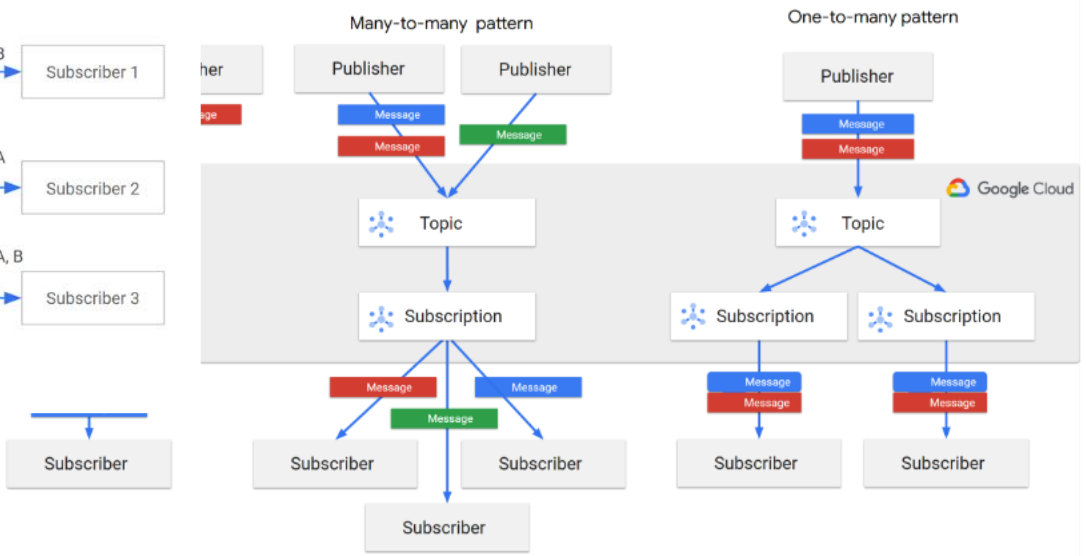
- But, how do you write the event driven microservices?
  - Well, you can just write code … …
  - Or use a state machines abstraction.
  - … …

Conceptual Approach



Google Pub Sub

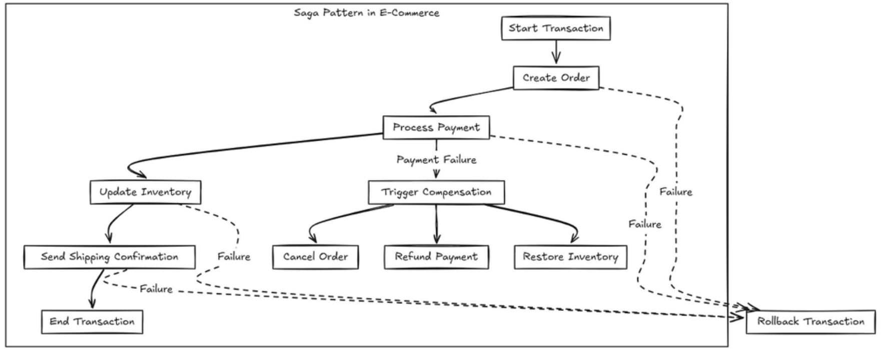


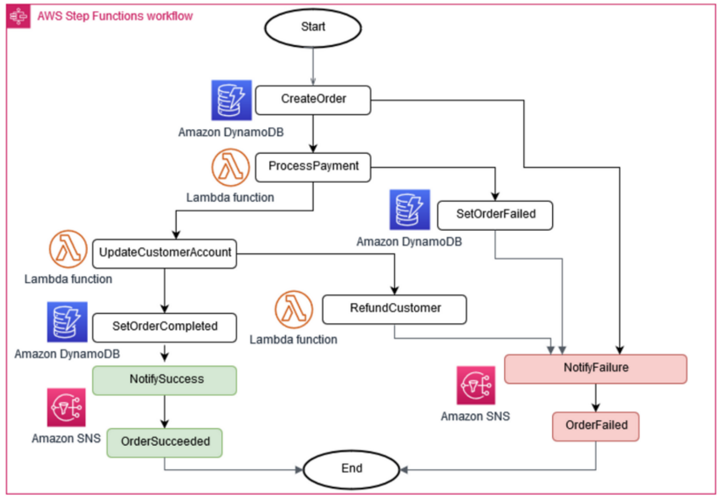


The following are the components of a Pub/Sub service:

- **Publisher** (also called a producer): creates messages and sends (put 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.

Service Orchestration and Saga

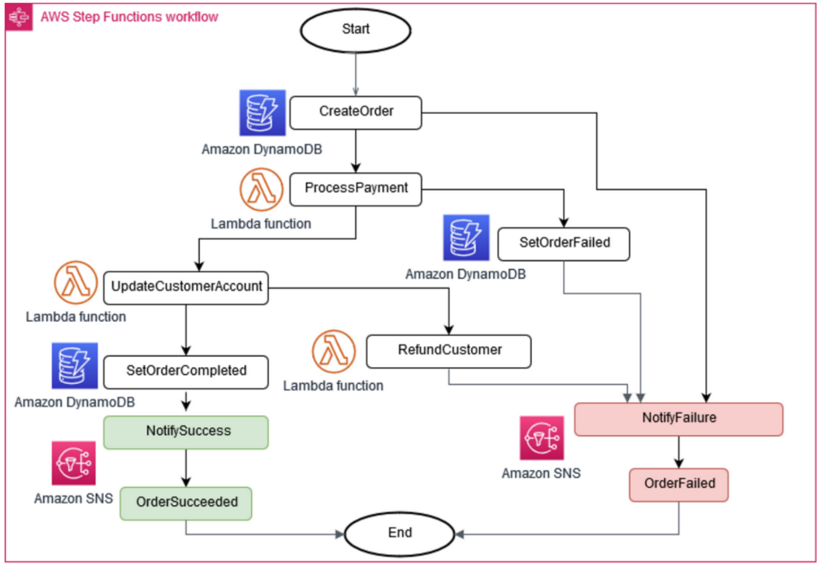


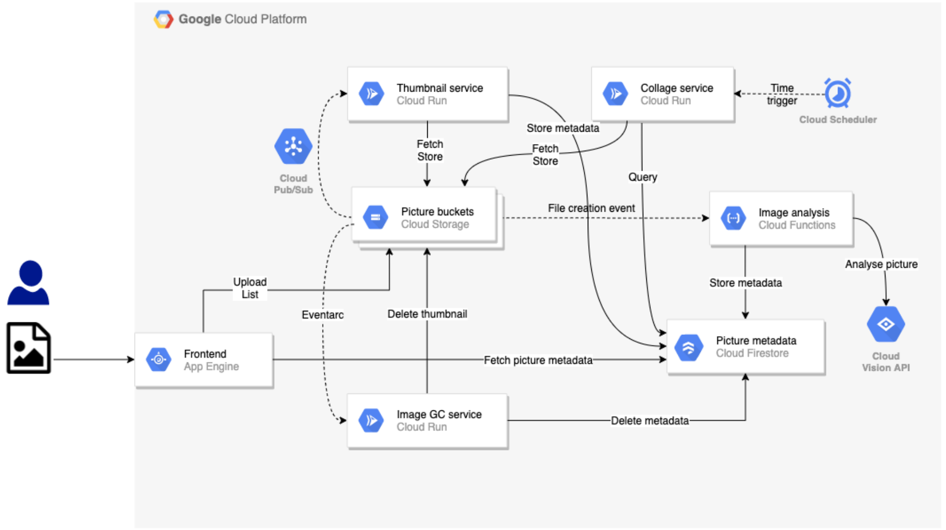


Saga (https://microservices.io/patterns/data/saga.html):

- You have applied the Database per Service pattern. Each service has its own database. Some business transactions, however, span multiple service so you need a mechanism to implement transactions that span services.
- Implement each business transaction that spans multiple services as a saga. A saga is a sequence of local transactions. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails because it violates a business rule then the saga executes a series of compensating transactions that undo the changes that were made by the preceding local transactions.
- There are two ways of coordination sagas:
  - Choreography - each local transaction publishes domain events that trigger local transactions in other services
  - Orchestration - an orchestrator (object) tells the participants what local transactions to execute

Orchestration





Both Google and AWS have approaches to “workflow” and “orchestration:”

- Development and management studios.
- High level language.
- Execution engine.

Summary

- You have implemented composition using code:
  - Synchronous
  - Asynchronous
- You will do something simple for some composite updates that implements
  - Choreography
    - Pub/Sub
    - Topics
    - Events and subscriptions
    - ”Do” and “Rollback” functions
  - Orchestration with compensation using one of the engines
  - Access via an API
  - Something very simple