WSO2CON 2024

Navigating API Complexity:
REST, GraphQL, gRPC, WebSockets, WebSub, AsyncAPI and more



Nuwan Dias
Deputy CTO
WSO2

REST – Representational State Transfer

- An architecture style that uses standard conventions and protocols.
- Defined by Roy Fielding as part of his PHD thesis in 2000.
- Everything in REST is a resource.
  - Each resource is identified by its URI.
    - GET /employees/{employeeId}
- REST operations often map to CRUD operations on resources.
  - POST (**c**reate), GET (**r**ead), PUT/PATCH (**u**pdate), DELETE (**d**elete)
- Based on the client-server model.
  - Client initiates request, server responds with data.
  - There can be intermediaries between client and server.
- Stateless
  - Each request should contain all data required to understand and process the request.
- Supports HATEOAS

REST in action



Pros of REST

- Simplicity
  - Easy to understand, well supported, leverages the widely adopted HTTP protocol.
  - Server implementation is very easy.
- Flexibility
  - Supports a wide range of data formats (JSON, XML, PlainText, etc).
  - Statelessness allows easy scaling / load-balancing.
- Security
  - Can leverage standard HTTP security mechanisms.
- Interoperability
  - Developers who understand REST can work with the tooling/technology of choice.
- Discoverability
  - Hyperlinks in responses allow clients to discover and navigate through resources.
- Compatibility
  - Backwards compatible changes can be done on servers without impacting clients.

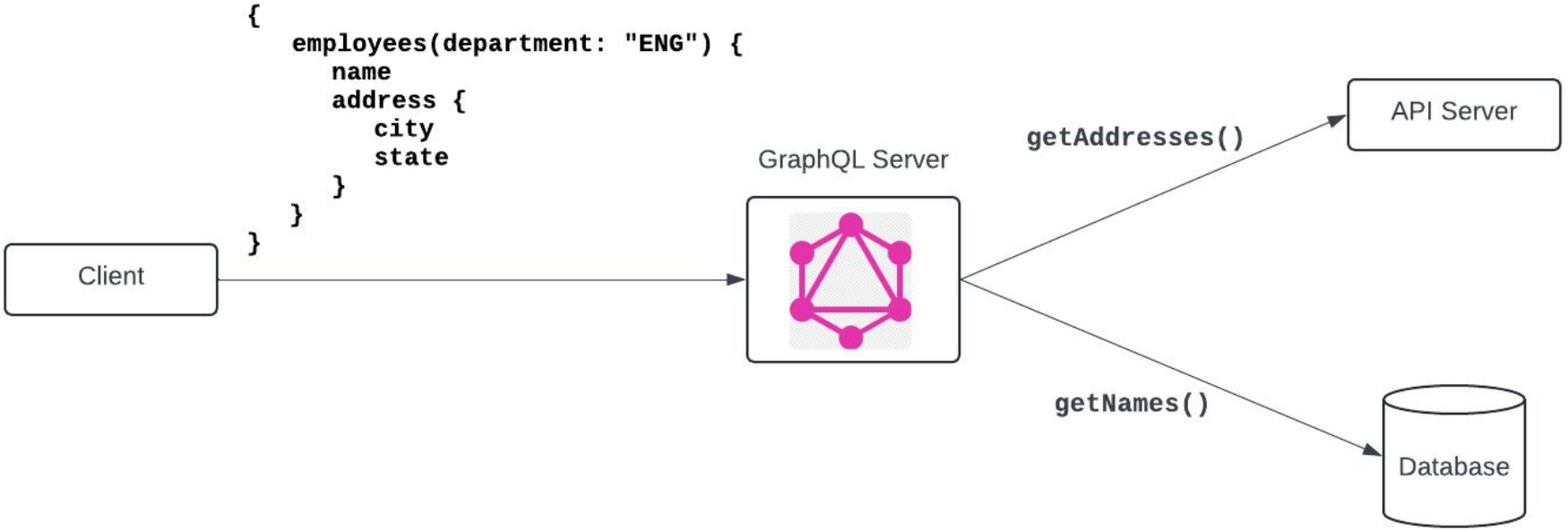
Cons of REST

- Over-fetching or under-fetching of data.
  - Returns unnecessary data or insufficient data about the requested resource.
  - Ex: Employee address.
- Verbosity for complex data retrieval.
  - Ex: `foreach employee in department (GET /api/employees/{id}/address)`
- Limited support for real-time updates.
  - Not so suitable for chatty applications.
  - In long running operations, clients would need to keep polling the server for updates.
- Versioning complexities.
  - Need to maintain multiple API versions to support old clients.

GraphQL

- Developed by Facebook in 2012, open sourced in 2015.
- A query language for APIs that allow clients to request the exact data they want.
  - Takes a declarative approach. Clients define the data they want, not how to fetch it.
- Key Concepts
  - Schema: Defines the data and relationships. I.e. A blueprint of the data structure.
  - Queries: Used by clients for fetching/reading data.
  - Mutations: Used by clients for modifying data.
  - Subscriptions: Used for real time updates on data.
- Strongly typed
  - Strong type enforcement reduces the possibility of errors.

GraphQL in action



Pros and cons of GraphQL

Pros

- No over-fetching or under-fetching of data.
- Efficient usage of network bandwidth between client and server.
- Efficient for complex data relationships.
- Strong typing eliminates errors early.
- Supports real-time updates via subscriptions.

Pros

- Complex server side implementations.
- Clients may request more data than they actually need (over-fetching).
- Hard to understand.
- Complex error handling.
- Caching requires more work (i.e. Everything is a POST)

REST vs GraphQL

REST is better for,

- Simple operations such as CRUD.
- Public* APIs that drive business. I.e. Twilio.
- Idempotent operations that may rely on caching.
- Scenarios that require monetization by number of requests.
- Highly scalable applications where statelessness matters.

GraphQL is better for,

- Complex data relationships.
- Scenarios where clients benefit from querying the data instead of navigating.
- Scenarios where clients need real-time updates of data.

* Public - APIs used by people outside the organization, over the internet.

gRPC

- An RPC framework that allows defining functions on servers and invoking them via remote clients.
- Developed by Google in 2015.
- Utilizes protocol buffers for data serialization.
- Relies on HTTP/2 as its transport protocol.
  - gRPC is highly performant compared to HTTP/REST due to its binary protocol, multiplexing support and header compression.
- Strongly typed, similar to GraphQL.
- Supports bi-directional streaming.
- Works across programming languages using generated code for server and client stubs.

Pros and cons of gRPC

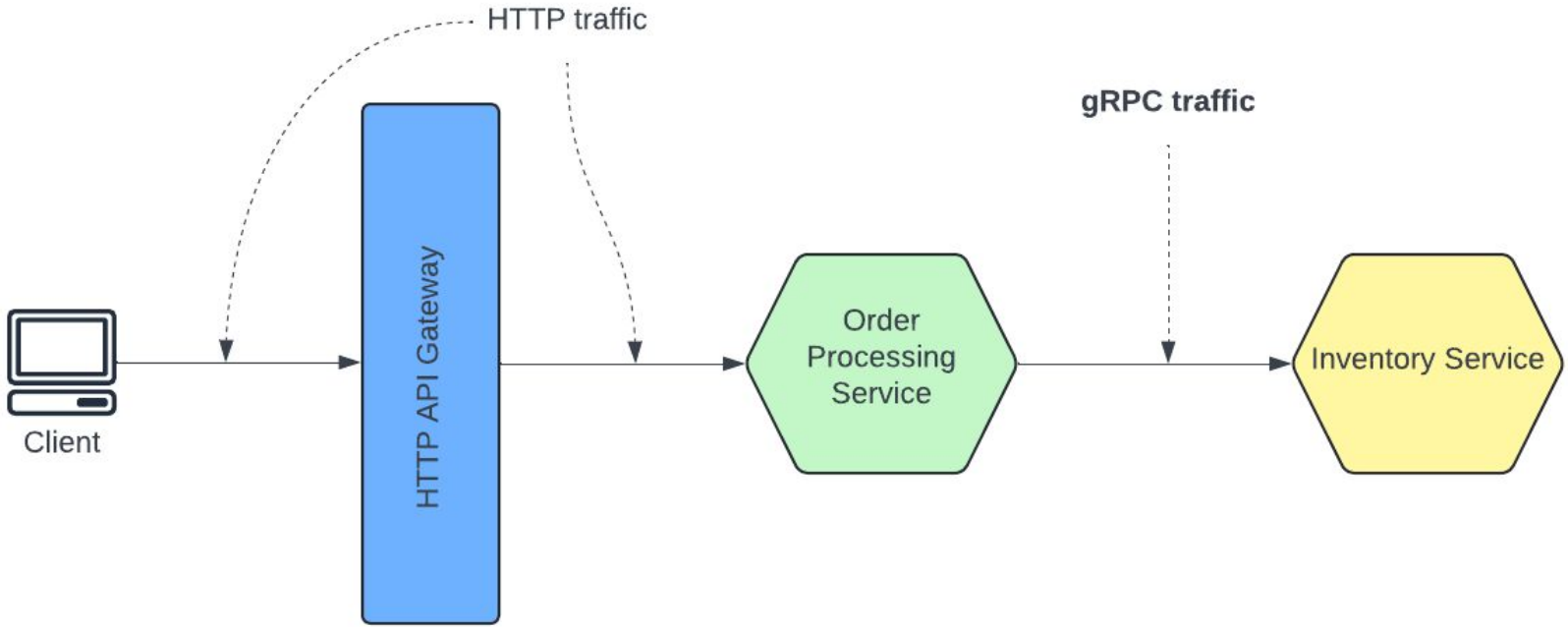
Pros

- Highly performant due to using protocol buffers for data serialization.
- Strongly typed, limiting errors.
- Based on HTTP/2, hence making it possible to leverage benefits of HTTP.
- Supports bi-directional streaming.

Pros

- Has a considerable learning curve since it is based on a binary protocol (protocol buffers).
- Limited browser (client) support.
- Any changes on the server will break clients unless they are updated too.
- Frequent bi-directional comms utilizes a lot of energy.
- Cannot leverage HTTP caching.

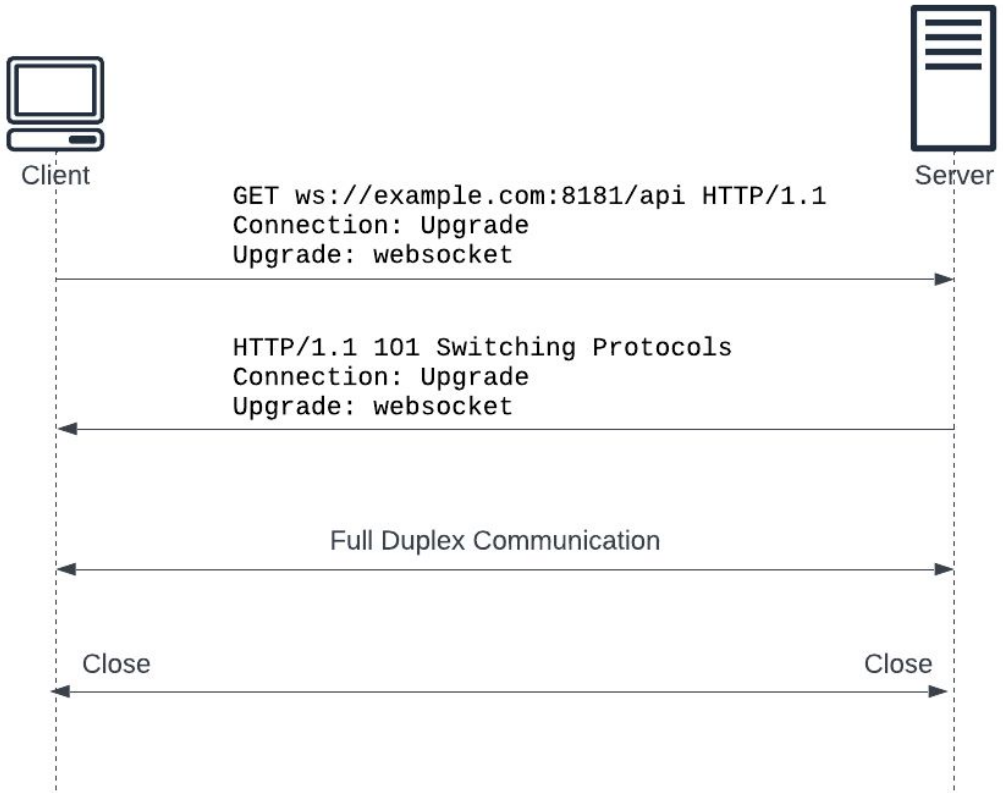
gRPC


HTTP traffic
gRPC traffic
Client
HTTP API Gateway
Order Processing Service
Inventory Service

WebSockets

- Before WebSockets, clients used to use long polling to receive real time updates from servers.
- WebSockets allowed bi-directional communications over a single HTTP connection, allowing low-latency real time updates.
- Suitable for chatty applications which require full-duplex communications (i.e chat systems, live data feeds).
- Compared to REST, operates with less HTTP connections, but each connection will have a much longer TTL.
- gRPC vs WebSockets?
  - WebSockets are for end applications (browser, mobile), gRPC is better for internal communications between services.

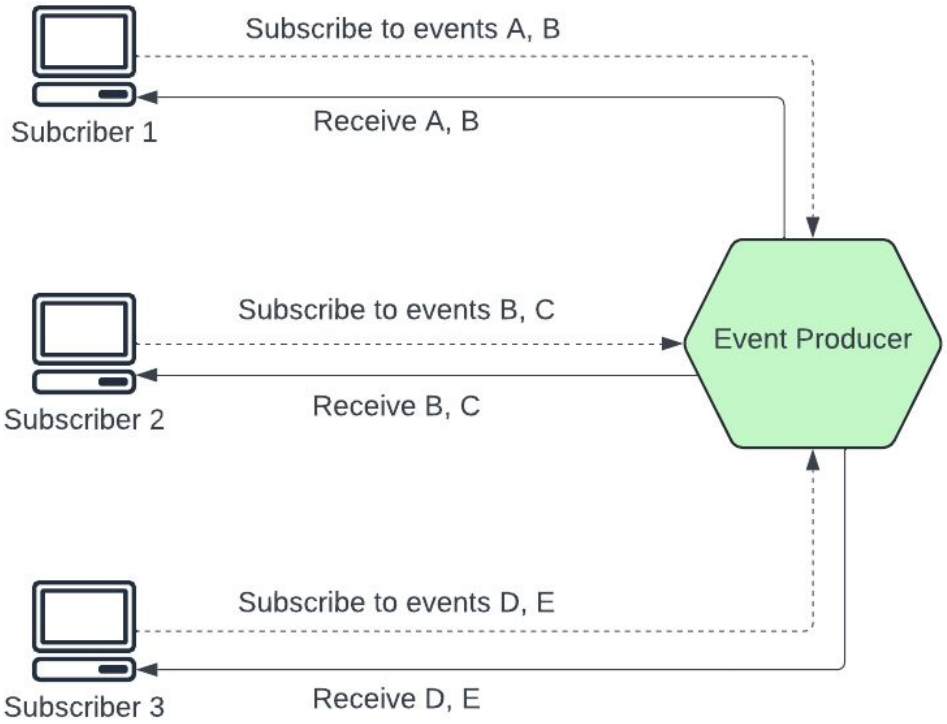
WebSockets


Client
Server
GET ws://example.com:8181/api HTTP/1.1
Connection: Upgrade
Upgrade: websocket
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: websocket
Full Duplex Communication
Close
Close

WebHooks

- Originated in the early 2000s as a more dynamic way for web applications to communicate compared to polling.
- It is a user/client configured call-back mechanism over HTTP.
- A publisher (event producer) registers callbacks of parties interested in receiving events and notifies them.
  - Notifications usually happen over HTTP POST.
- Subscribers (event receivers) act upon received events.
- Usually used to integrate software with partner systems.
  - Loosely coupled integrations.
- Publisher has to maintain list of subscribers and sending events to each subscriber.
  - Complex server-side implementation

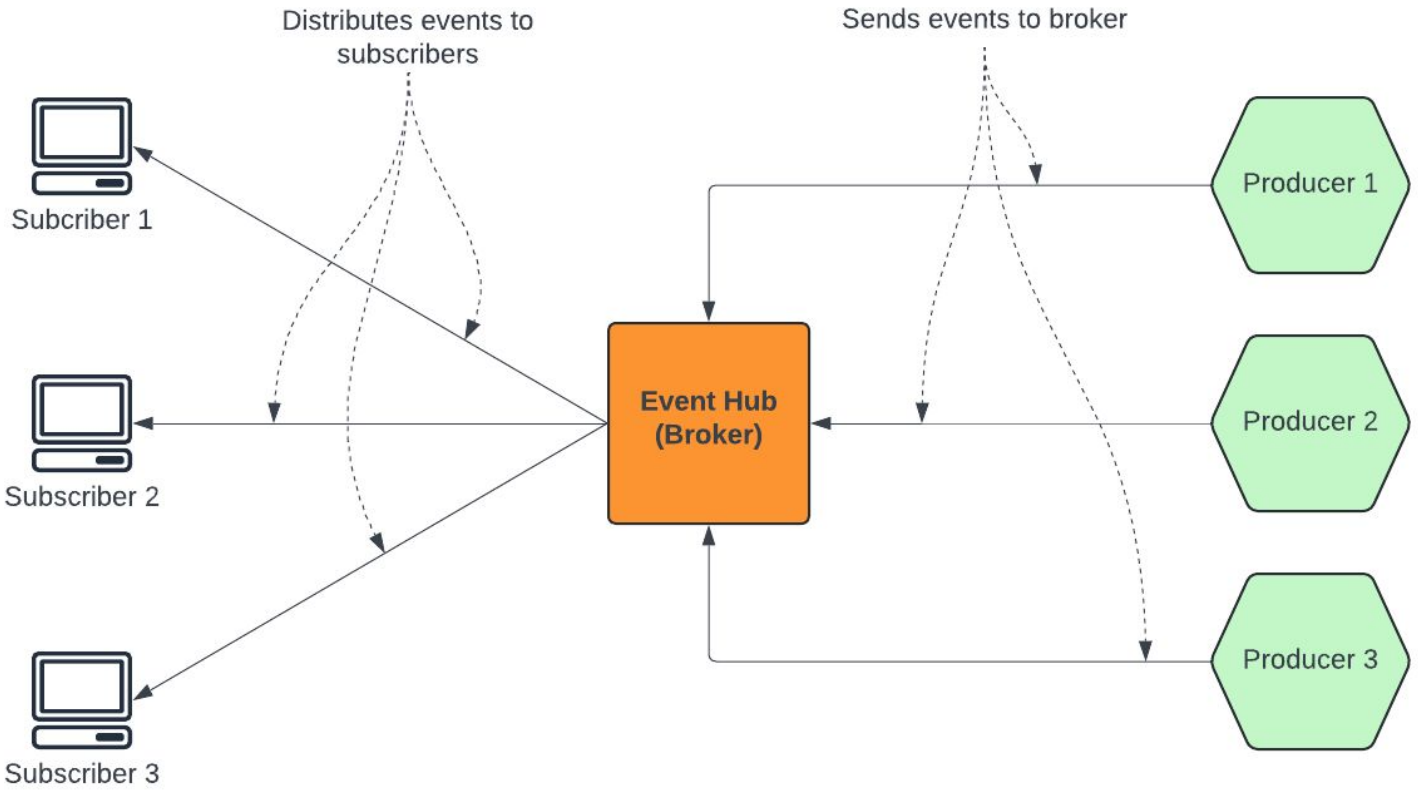
WebHooks illustrated



WebSub

- Emerged in around 2017.
- Addresses the limitations in the WebHook approach.
- Utilizes a Hub based architecture to decouple publishers from subscribers.
  - Publisher notifies Hub, Hub notifies interested subscribers.
- Simplifies the publisher architecture.
- Provides more reliability and robustness due to better error handling.
- Allows the Hub to scale independently.

WebSub illustrated



API Specification: REST

- OpenAPI is the most popular API specification for REST.
  - Others include RAML and API Blueprint.
- Latest active version: v3.1.0
- OpenAPI v4.0 (Project Moonwalk) is in the works.

https://github.com/OAI/sig-moonwalk

```yaml
openapi: 3.1.0
info:
  title: Inventory Management API
  version: 1.0.0
  description: API for managing items in an inventory system

servers:
  - url: https://your-api-domain.com/api/v1

paths:
  /items:
    get:
      summary: List all items
      description: Retrieves a paginated list of all items in the inver
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Item'
        '401':
          description: Unauthorized
          content: {}
    post:
      summary: Create a new item
      description: Creates a new item in the inventory.
```

API Specification: GraphQL

- GraphQL Schema Definition Language (SDL).

Query

```
{
  hero {
    name
    friends {
      name
      homeWorld {
        name
        climate
      }
      species {
        name
        lifespan
        origin {
          name
        }
      }
    }
  }
}
```

Schema

```
type Query {
  hero: Character
}

type Character {
  name: String
  friends: [Character]
  homeWorld: Planet
  species: Species
}

type Planet {
  name: String
  climate: String
}

type Species {
  name: String
  lifespan: Int
  origin: Planet
}
```

Image source: https://graphql.org/

API Specification: WebSocket, WebHooks, WebSub, etc.

- AsyncAPI (https://www.asyncapi.com/) is the most popular API specification for event driven APIs.
- It serves both producers and subscribers.

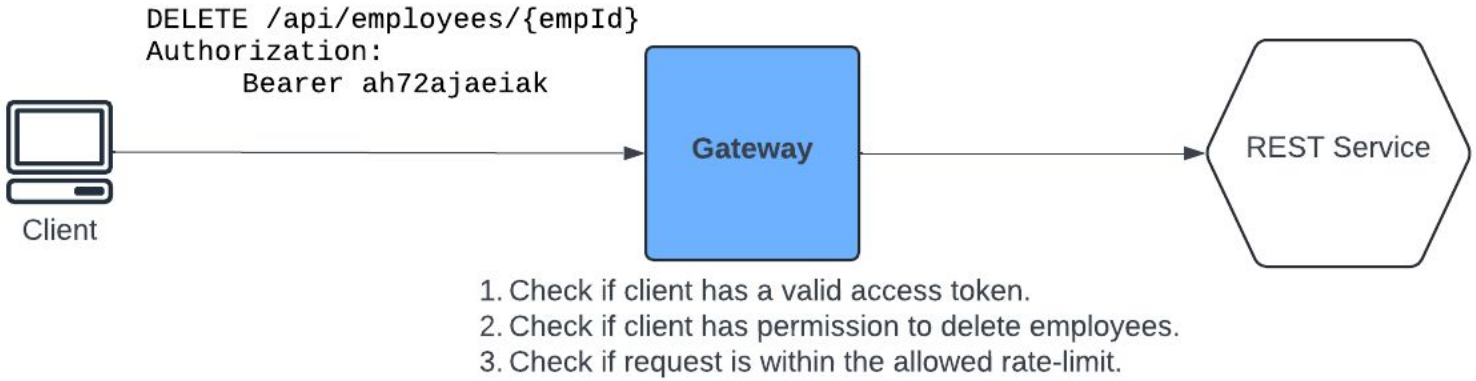
```yaml
servers:
  public:
    url: ws.kraken.com
    protocol: wss
    description: |
      Public server available without authorization.

channels:
  /:
    publish:
      description: Send messages to the API
      operationId: processReceivedMessage
      message:
        oneOf:
          - $ref: '#/components/messages/ping'
          - $ref: '#/components/messages/subscribe'
          - $ref: '#/components/messages/unsubscribe'

    subscribe:
      description: Messages that you receive from the API
      operationId: sendMessage
      message:
        oneOf:
          - $ref: '#/components/messages/pong'
          - $ref: '#/components/messages/heartbeat'
```

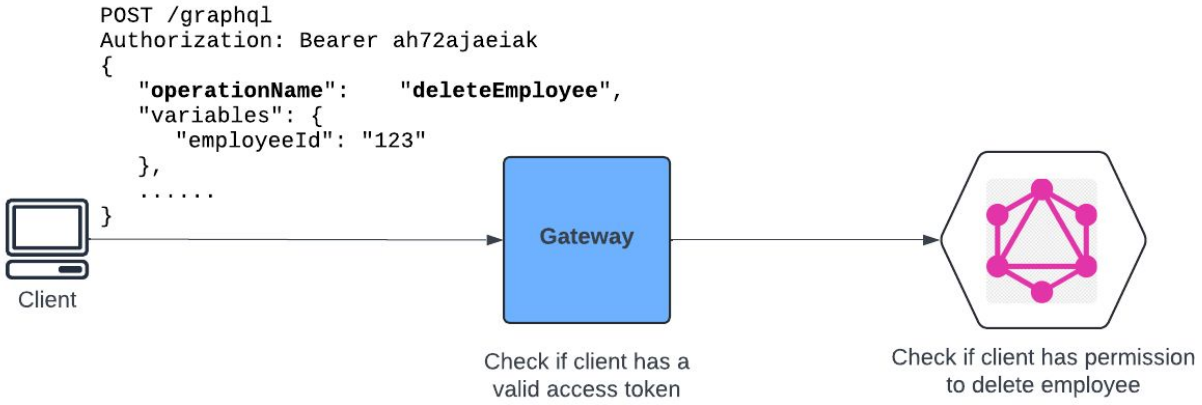
Securing REST APIs.

- OAuth2.0 is the widely used protocol for securing REST APIs.
- An API Gateway can perform most of the AuthN/AuthZ of REST APIs.
  - I.e. HTTP resources (`DELETE /employees/{empId}`) directly correspond to API functions.
- However, data entitlements are better performed at the API/service level itself.
  - I.e. I can access employee data but only if they belong to departments foo, bar.



Securing GraphQL APIs.

- Authentication for GraphQL APIs can be achieved using OAuth2.0 access tokens (JWTs).
- Unlike REST, GraphQL does not expose data or actions on resource paths and HTTP verbs.
  - Performing authorization on GraphQL therefore requires introspecting the message payload.



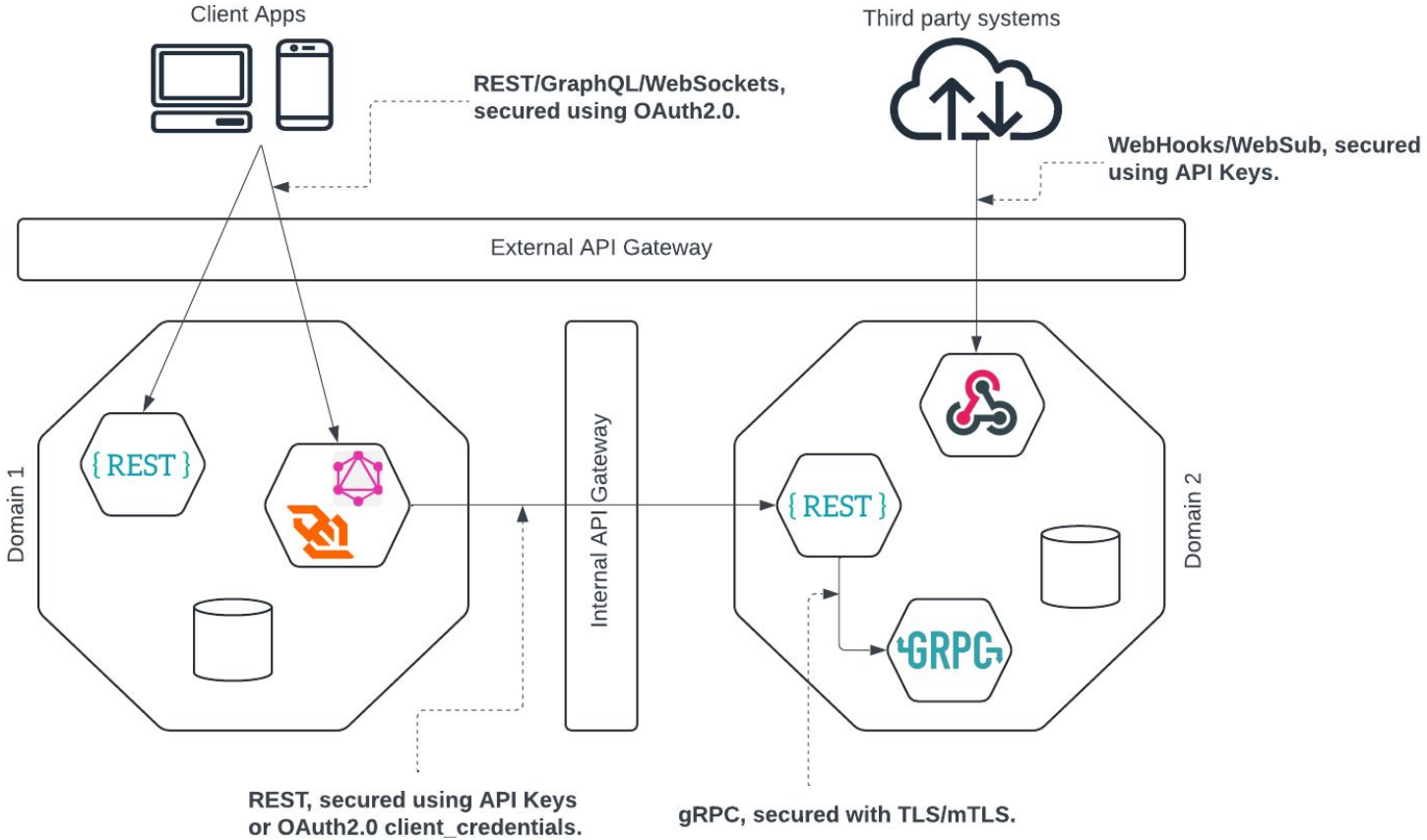
Rate-limiting GraphQL APIs.

- Rate limits are used to protect APIs from being exhausted.
- With GraphQL, limiting the number of requests by itself isn't sufficient.
  - Since clients have the freedom to execute any query, even a single request may result in exhaustion of the API.
  - Two such common issues are due to nested queries and too many nodes.
- Rate limiting GraphQL therefore require special solutions such as limiting query depth and applying node limits.

```
query GetUserDetails {
    user(id: 1) {
        name department {
            name manager {
                name department {
                    name manager {
                        . . . . .
```

```
query {
   viewer {
      repositories(first: 50) {
         edges {
            repository:node {
               name
               issues(first:10) {
                  totalCount
```

APIs in Action



The Kubernetes (declarative) API.

- Think GraphQL, you just declare the desired state, not the steps to execute.
- Similarly, Kubernetes has a REST API that is declarative in nature.
  - You specify the desired state. Kubernetes controllers makes sure the system is brought to that desired state.
  - **`kubectl apply -f service.yaml`**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-web-service
spec:
  selector:
    app: my-web-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

Question Time!



Thank You!