Amazon API Gateway

Paul Lu

Sr. Solution Architect

Agenda

Overview of Serverless

What is Amazon API Gateway

API Types

Amazon API Gateway Features

Best Practices

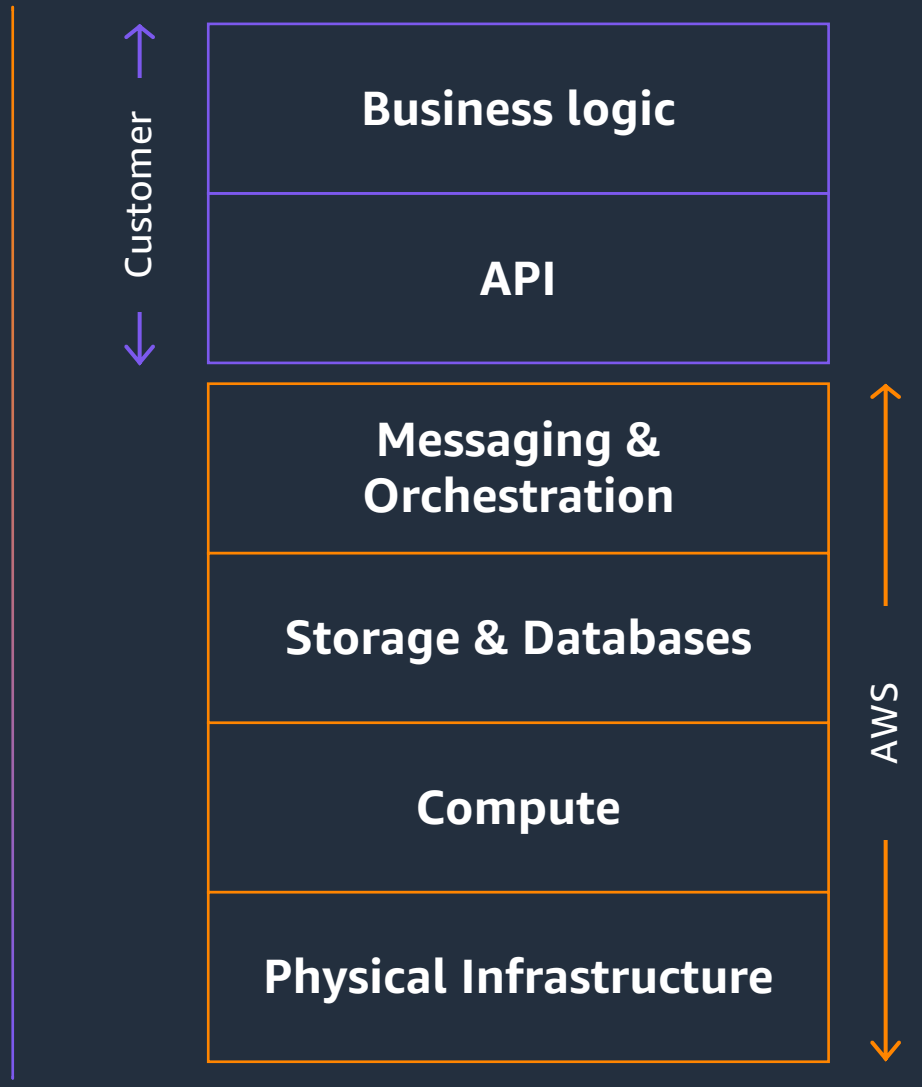
Overview of Serverless

What does the future look like?

ALL THE CODE YOU EVER WRITE IS BUSINESS LOGIC

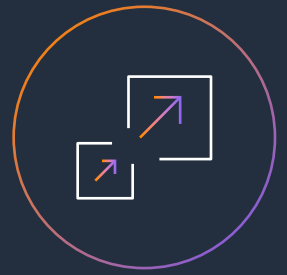
What is Serverless?

Serverless services simplify the management and scaling of cloud applications by shifting undifferentiated operational tasks to the cloud provider so **development teams can focus on writing code** that solve business problems



Why serverless

No infrastructure provisioning, no management



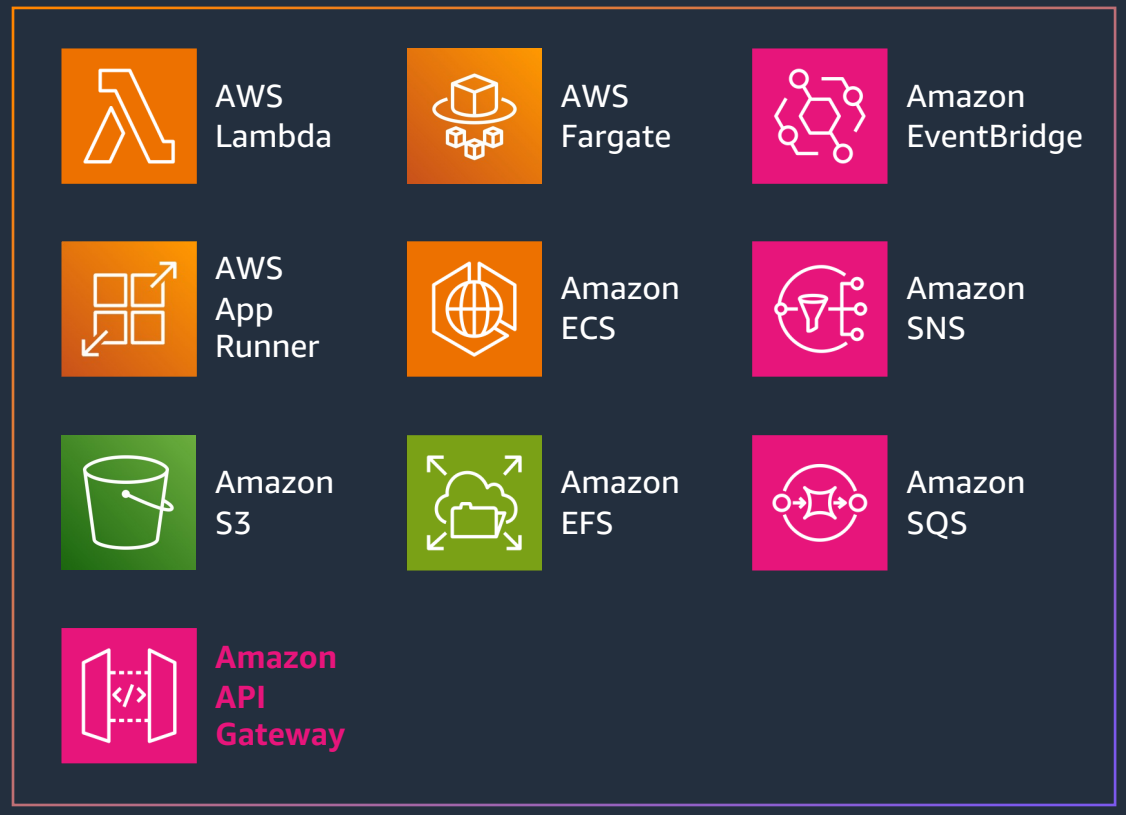
Automatic scaling

Pay for value

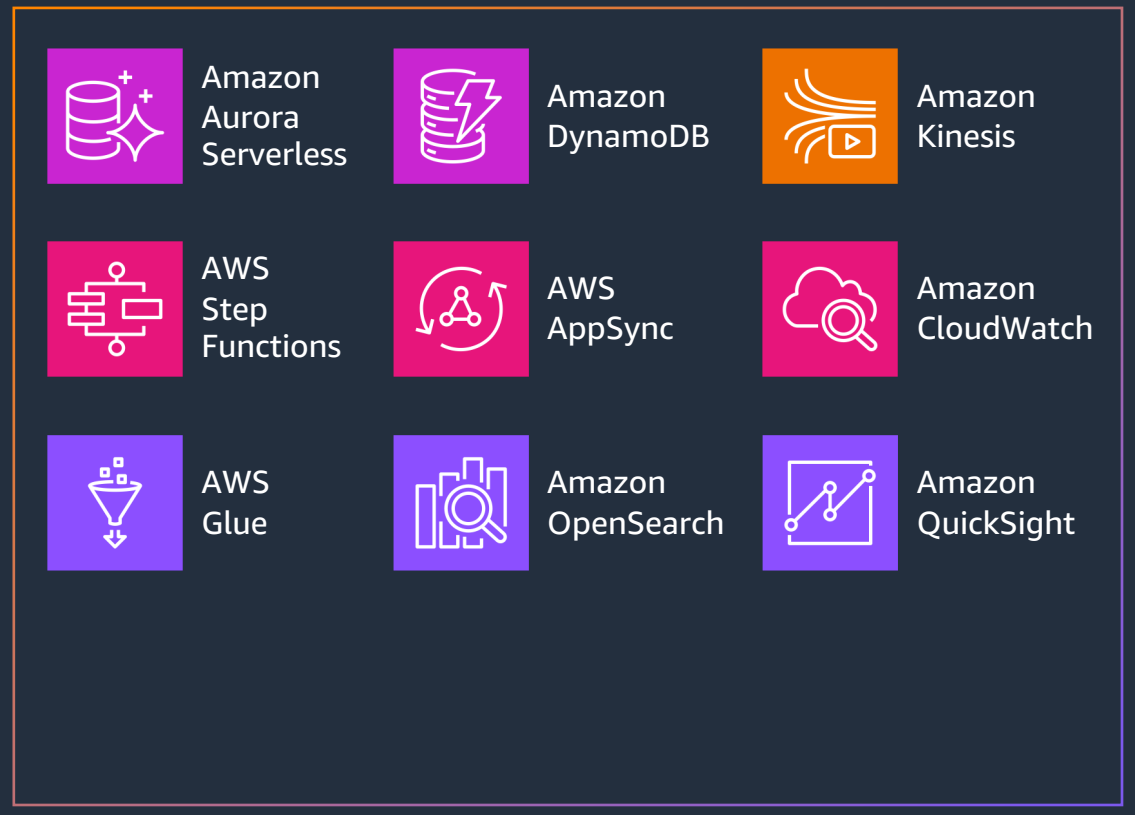
Highly available and secure

AWS serverless spectrum

From Primitives to…
(Examples: compute, containers, buses)



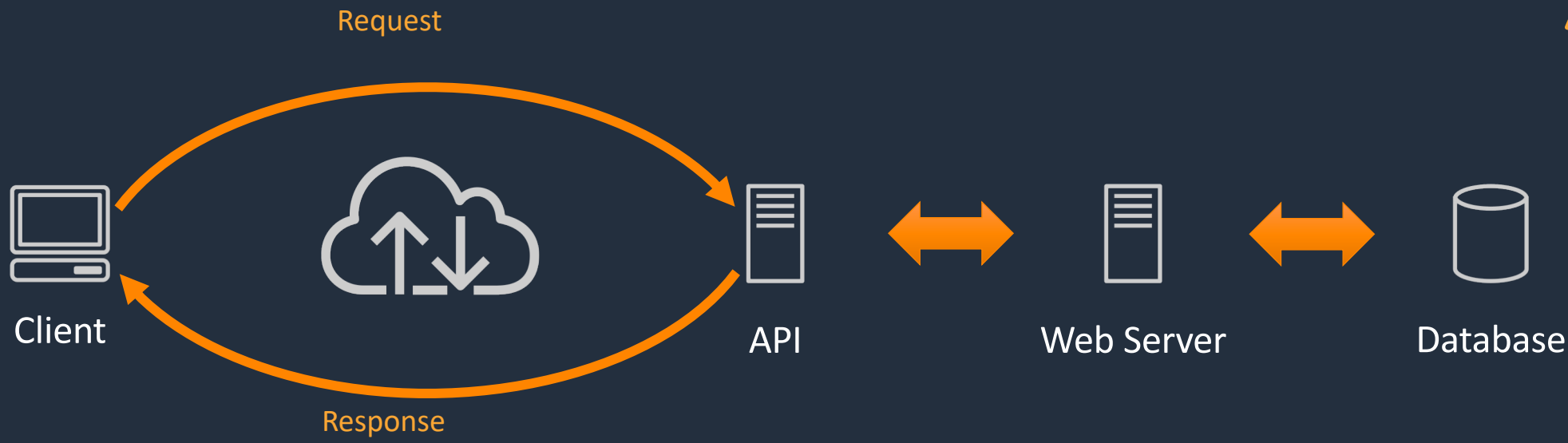
…Peripherals
(Examples: databases, analytics, workflows)



What is Amazon API Gateway?

Application Programming Interface (API)

> In building applications, an API simplifies programming by abstracting the underlying implementation and only exposing objects or actions the developer needs.



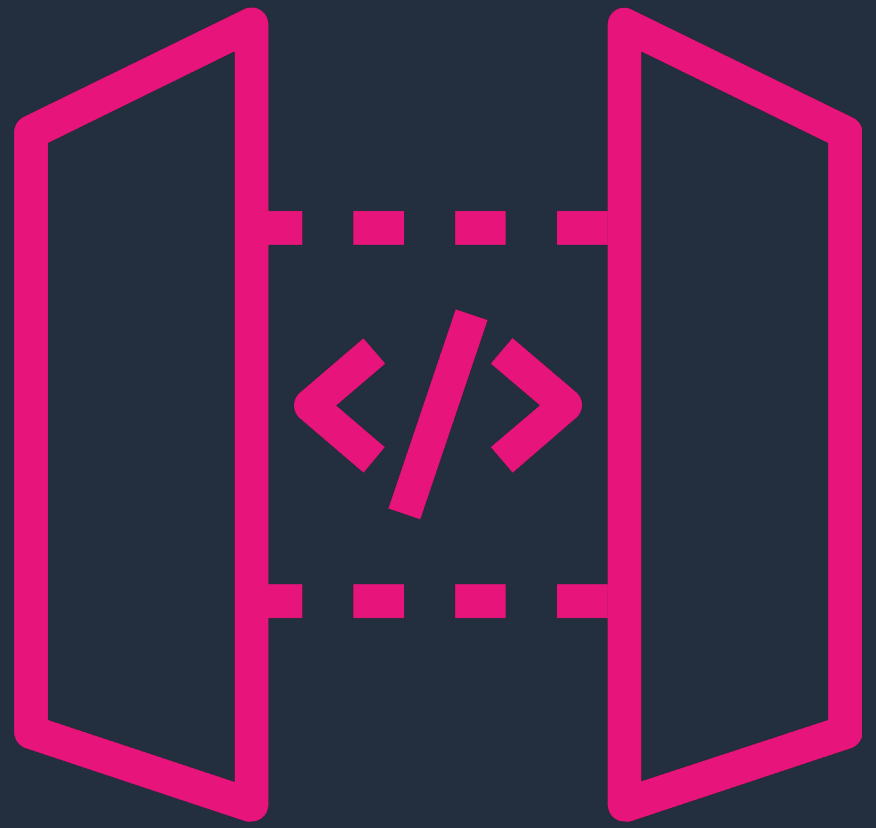
Web-based companies and services offer APIs for developers to use, such as:

- Social Networks – Facebook, Twitter, etc
- Payment Processing – Amazon Pay, PayPal, etc

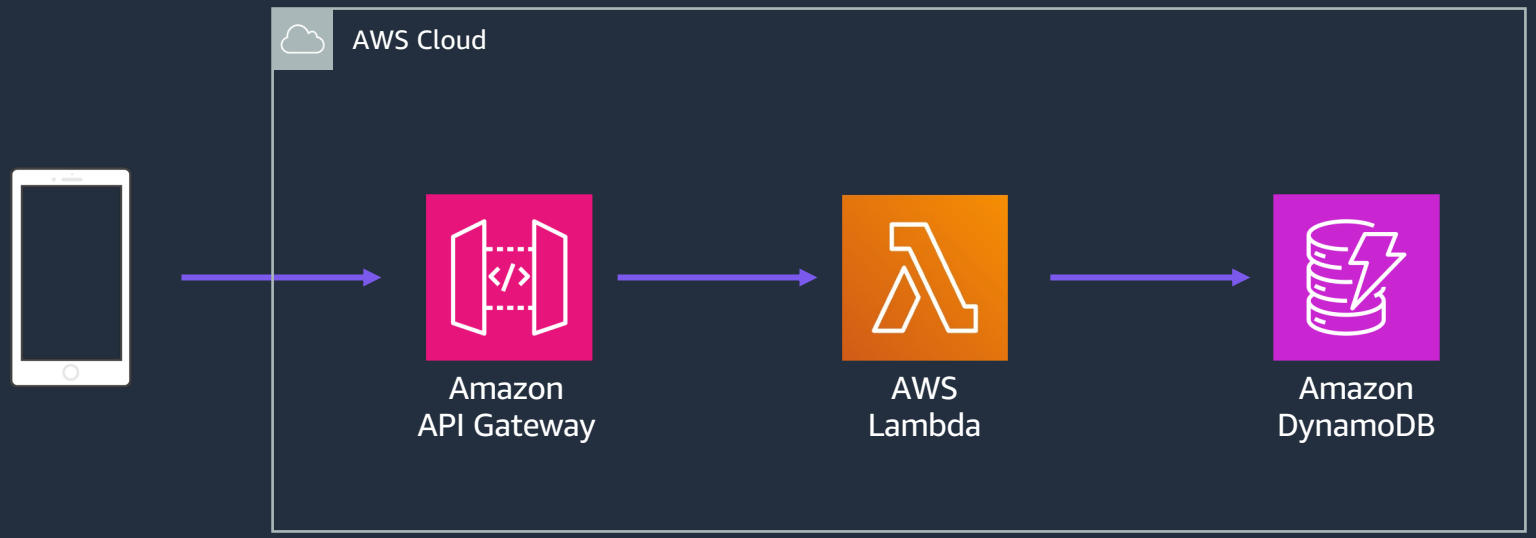
https://en.wikipedia.org/wiki/Application_programming_interface

Amazon API Gateway

Amazon API Gateway is a fully managed (serverless) service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.



API Gateway is a front door…



…which handles common concerns enabling developers to focus on business logic

- Throttling
- Caching
- Authorization
- API Keys
- Usage Plans
- Request/Response Mapping

Endpoint types

Edge-Optimized [REST]

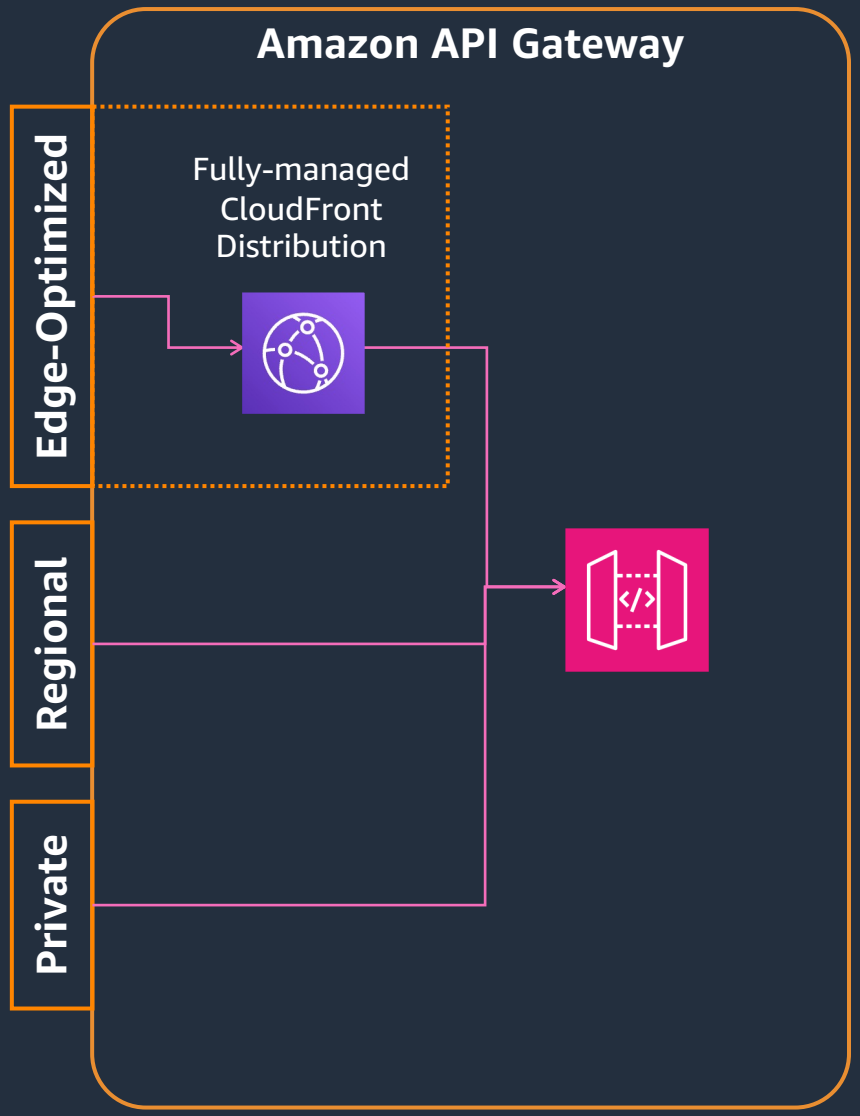
- Utilizes CloudFront to reduce TLS connection overhead (reduces roundtrip time)
- Designed for a globally distributed set of clients

Regional [BOTH]

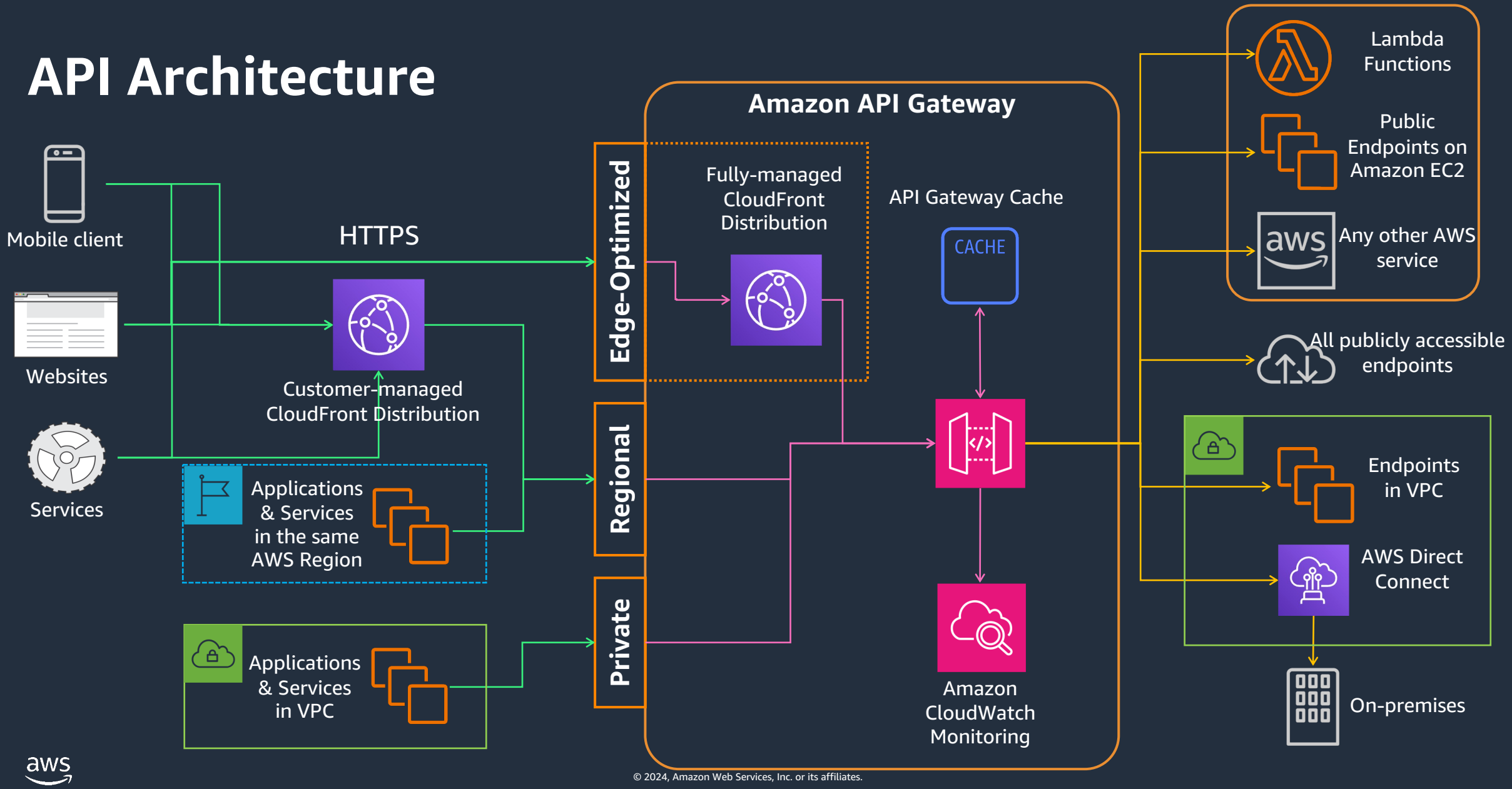
- Recommended API type for general use cases
- Designed for building APIs for clients in the same region

Private [REST]

- Only accessible from within VPC (and networks connected to VPC)
- Designed for building APIs used internally or by private microservices



API Architecture

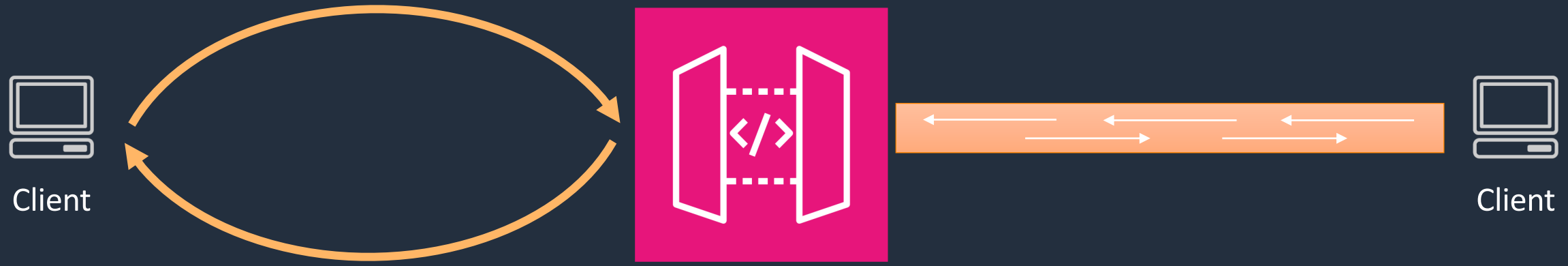


API Types

Supported Protocols

RESTful APIs

WebSocket APIs



- Request / Response
- HTTP Methods like GET, POST, etc
- Short-lived communication
- Stateless

- Serverless WebSocket
- 2 way communication channel
- Long-lived communication
- Stateful

RESTful APIs: When should you use REST vs HTTP?

- Two flavors: REST API and HTTP API
- REST API is more feature rich
- HTTP API is built from the ground up:
  - Faster – up to 60% faster
  - Lower cost – up to 71% less expensive
  - Easier to use

Amazon API Gateway Features

Authorization

Types of authorization

- Mutual TLS
- IAM
- Lambda Authorizers

+

- WAF
- Cognito User Pools
- Resource Policies
- JWT

Throttling and Usage Plans

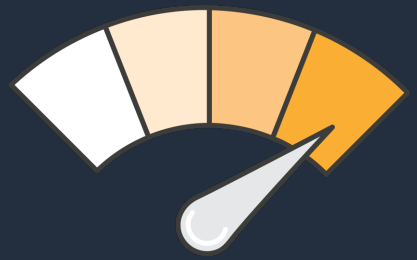
Rest only

Throttling and usage plans

- Protect backend systems
- Prevents one customer from consuming all your backend system’s capacity
- Let’s you decide how to allocate capacity among your API consumers with quotas and request rates. Example:

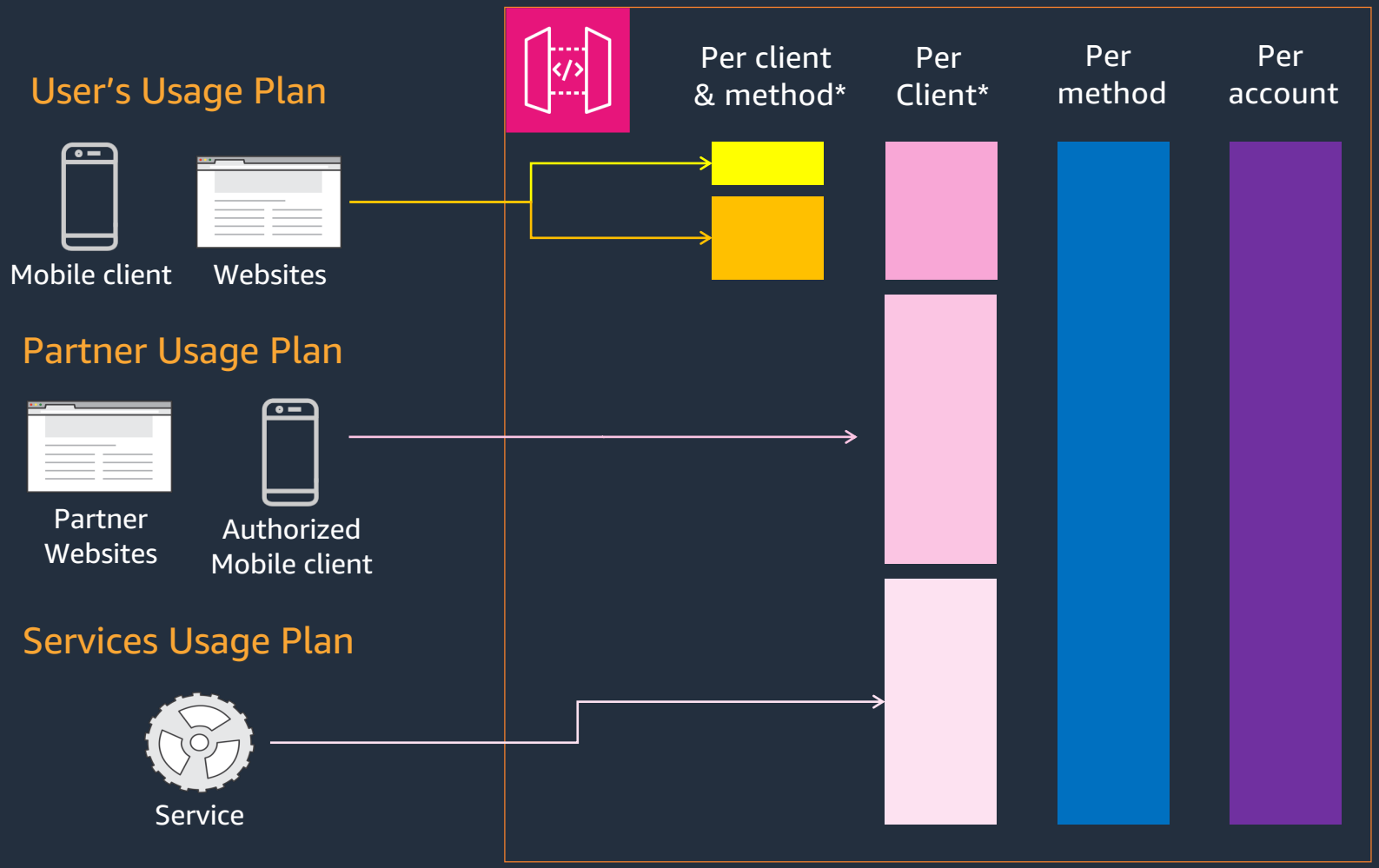
Professional plan users:
10 RPS, up to 100 calls / day

Enterprise plan users:
500 RPS, no limit on calls / day



Four levels of Throttling

* Requires Rest & API Keys and Usage Plans



Validation and Transformation

Rest only

Validation & Transformation

- Validations
  - The required request parameters in the URI, query string, and headers of an incoming request are included and non-blank.
  - The applicable request payload adheres to the configured JSON schema request model of the method.
- Transformations
  - Transform request to match expected format from backend
  - Transform response from API to match expected format for front end
  - Override request and response parameters based on message body
  - Override response status code

Stages and Versioning

Stages

API Gateway enables you to set stage variables, allowing the same API to point to different backends.

Your APIs are versioned and can be rolled back.

- APIs are deployed to staging environments.

  You choose what to name them.

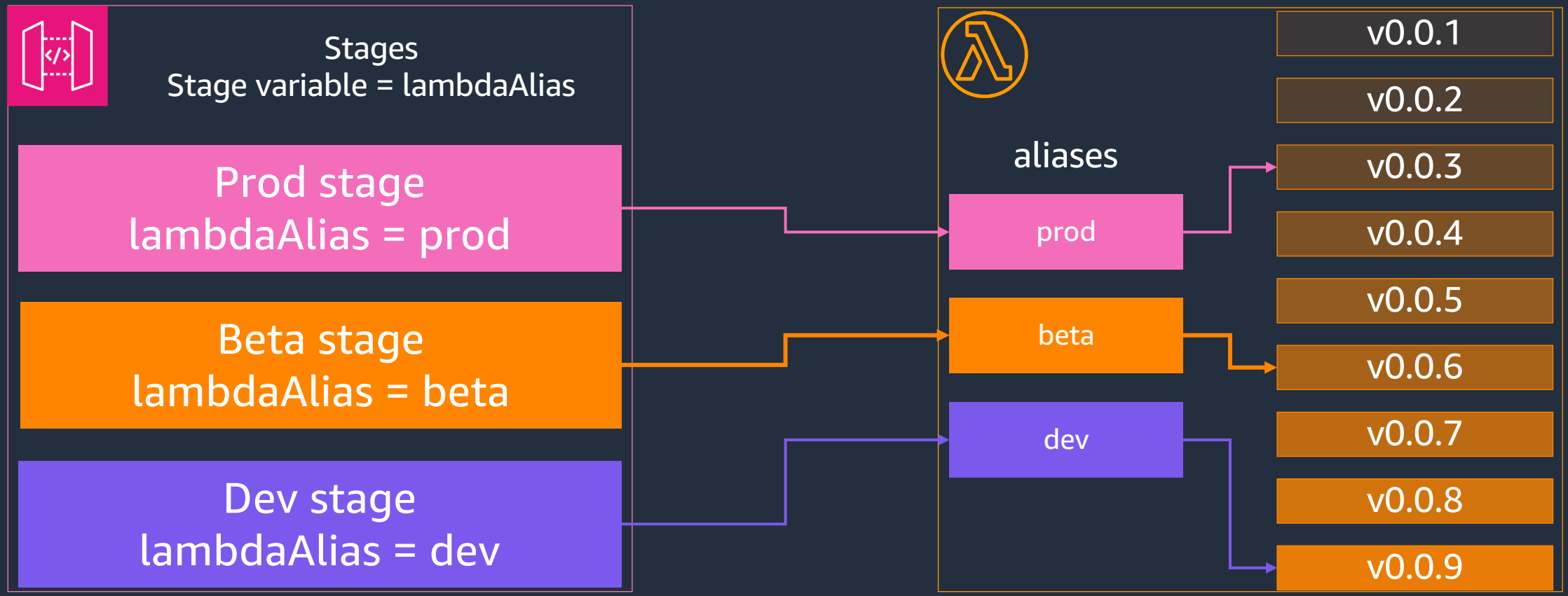
- For example, these environments:

  Dev (e.g., example.com/dev)

  Beta (e.g., example.com/beta)

  Prod (e.g., example.com/prod)

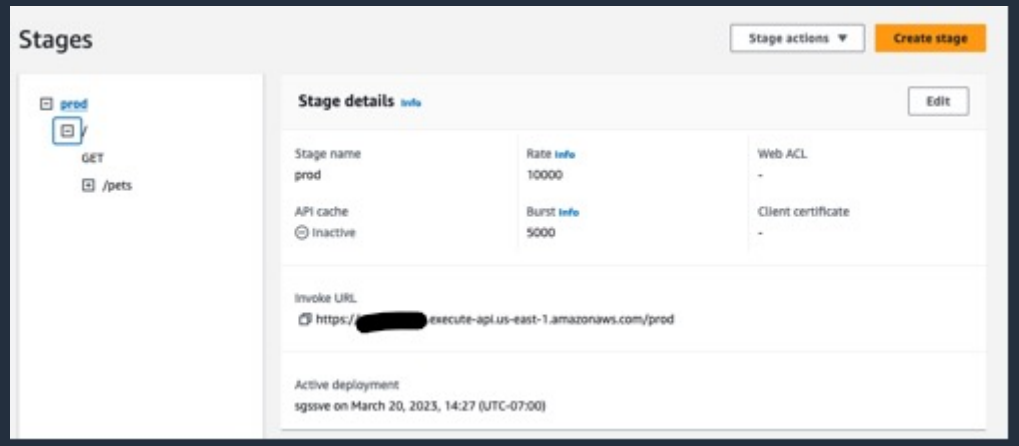
Stages API Gateway

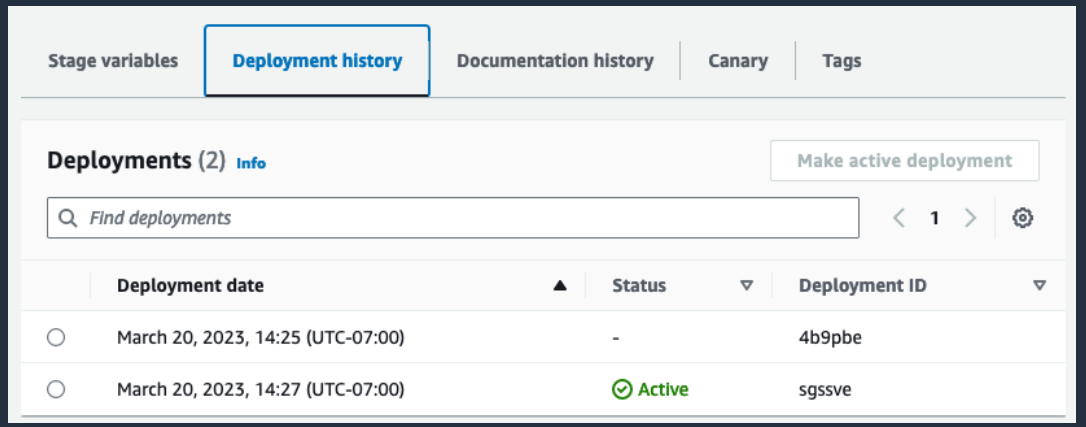


API Versioning [REST]

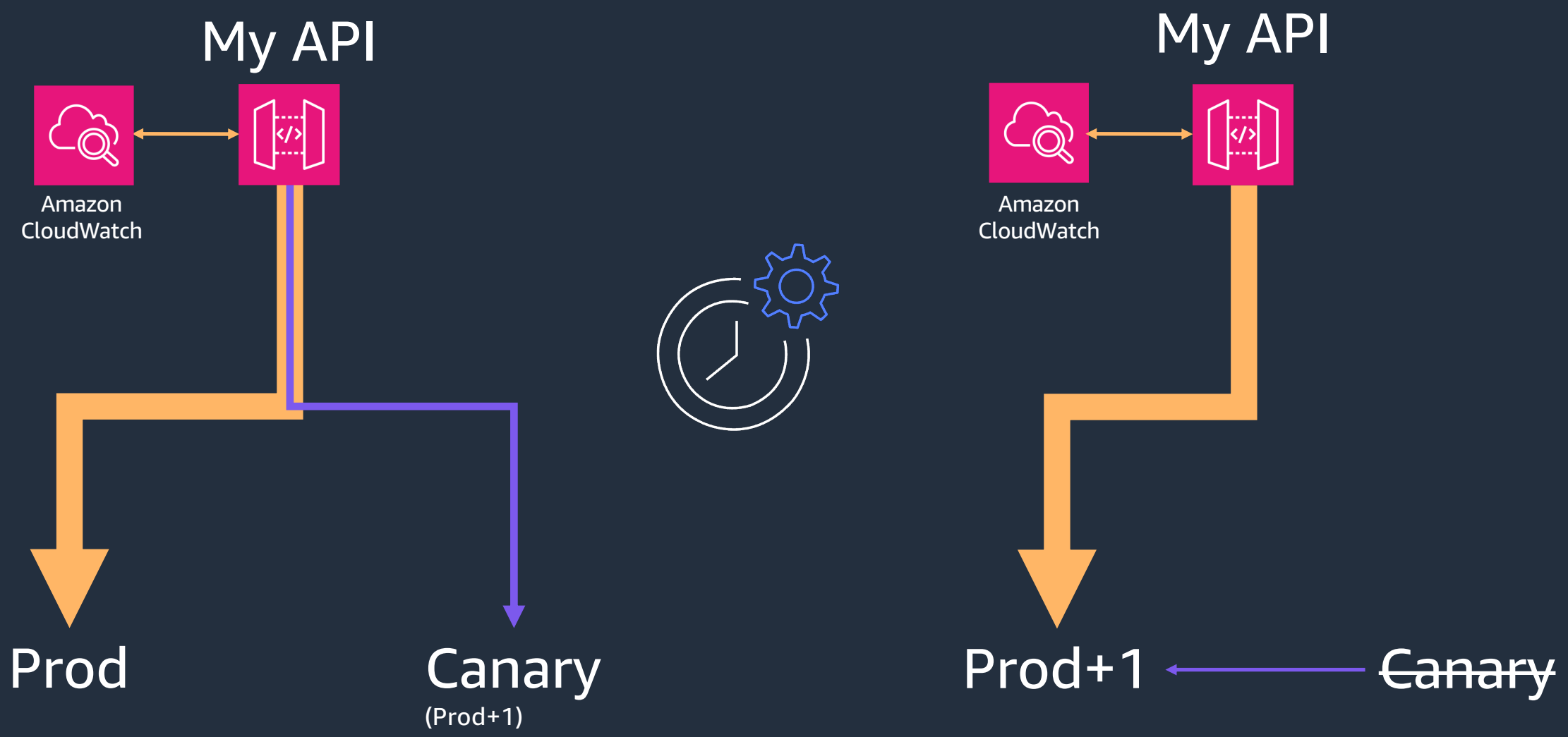
We support versioning inside the API Console allowing to easily roll back to a previous API version and deploy it.

- Easily roll back to different snapshots of the same API (“Deployments”).
- Each Stage points to a Deployment. You can update the Stage to point to a previous Deployment.
- Permits Canary Deployments.





Canary Releases [REST]



Observability

Logging

- Execution Logs [REST]
  - Two levels of logging, ERROR and INFO
  - Optionally log method request/body content
  - Set globally in stage, or override per method
  - CloudWatch Logs

- Access Logs
  - Customizable format for machine parsable logs (CLF, JSON, XML, CSV)
  - CloudWatch Logs OR Kinesis Firehose

Metrics

- Built-in
- REST
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, Cache Hit Count, Cache Miss Count
- HTTP
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, DataProcessed
- WebSocket
  - Connect Count, Message Count, Integration Error, Client Error, Execution Error, Integration Latency
- Custom
- Create Custom Metrics via Metric Filter out of logs

Important quotas

Important quotas

- Throughput: 10,000 Requests/second
- Max integration timeout: *30 seconds*
- Payload size: *10 MB*

- WebSocket connections: 500 connections/sec
- WebSocket connection duration: *2 hours*
- WebSocket message size: *128 KB*

More: https://docs.aws.amazon.com/apigateway/latest/developerguide/limits.html

Best Practices

Best Practices

- Ensure you apply security features.
- Do not hard code settings. Use stage variables.
- Set appropriate retention policy for CloudWatch Logs. REST API’s can emit logs to Kinesis Data Firehose
- Create an API per team/microservice and unify them with Custom Domain Name
- Use HTTP APIs if existing features are sufficient
- The console is for experimenting, use Infrastructure as Code

AWS Lambda

Run code without provisioning or managing servers

Introduction to AWS Lambda

Function-as-a-Service

Run code without provisioning or managing servers

Pay only for the compute time you consume

Automatically runs your code with high availability

Scale with usage

Lambda handles

- Load balancing
- Auto scaling
- Handling failures
- Security isolation
- OS management
- Managing utilization

AWS Compute Offerings

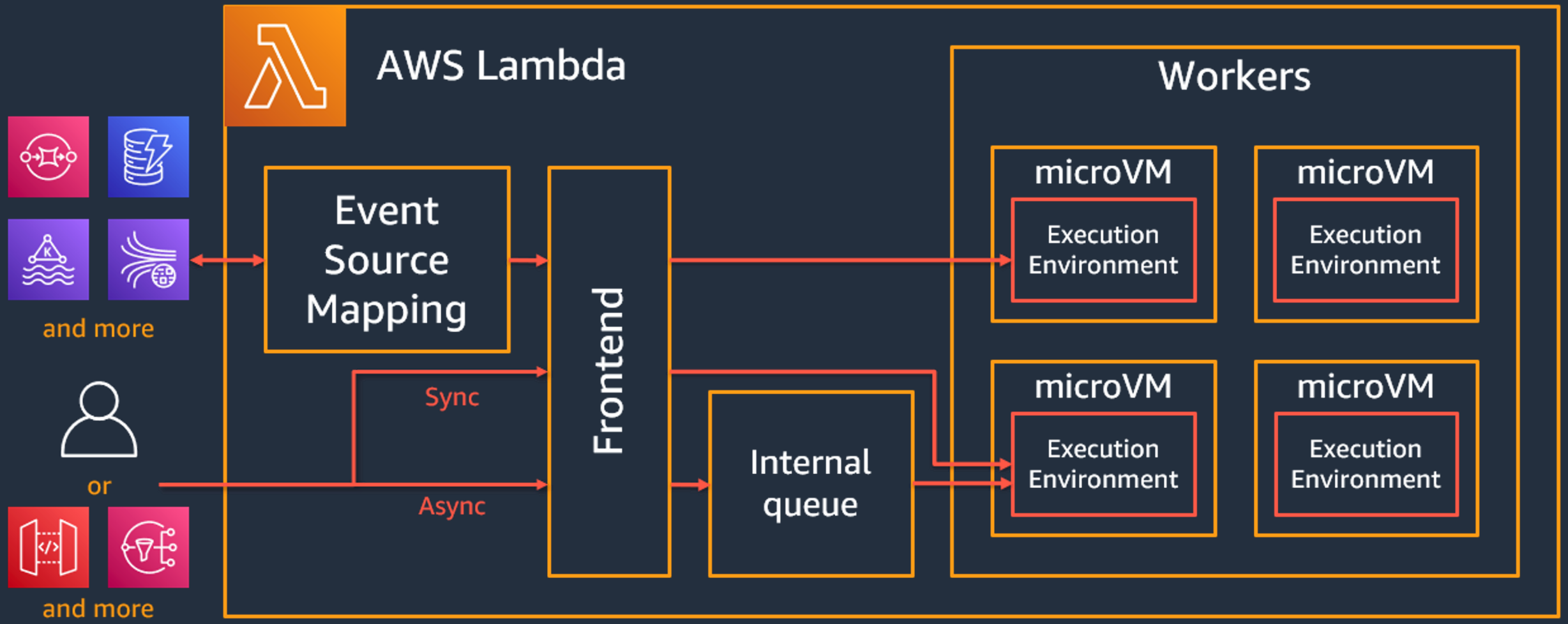
				
Service	**Amazon EC2**	**Amazon ECS**	**AWS Fargate**	**AWS Lambda**
Unit of scale	VM	Task	Task	Function
Level of abstraction	H/W	OS	OS	Runtime

AWS Compute Offerings

Service	 Amazon EC2	 Amazon ECS	 AWS Fargate	 AWS Lambda
How do I choose?	I want to configure servers, storage, networking, and my OS	I want to run servers, configure applications, and control scaling	I want to run my containers	Run my code when it's needed

Anatomy of a Lambda Function

AWS Lambda under the hood



Serverless applications

AWS Lambda

Serverless applications

Function

Node.js
Python
Java
C#
Go
Ruby
PowerShell
Runtime API

Serverless applications

Event source

Function

Changes in data state

Requests to endpoints

Changes in Resource state

Node.js
Python
Java
C#
Go
Ruby
PowerShell
Runtime API

aws

Serverless applications

Event source

Changes in data state

Requests to endpoints

Changes in Resource state

Function

Node.js
Python
Java
C#
Go
Ruby
PowerShell
Runtime API

Services

Anatomy of a Lambda Function

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda function Invocation

Context object

Methods available to interact with runtime information (request ID, log group, more)

```
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello World!')
    }
```

Monitoring and debugging Lambda functions

- AWS Lambda console includes a dashboard for functions
  - Lists all Lambda functions
  - Easy editing of resources, event sources and other settings
  - At-a-glance metrics
- Metrics automatically reported to Amazon CloudWatch for each Lambda function
  - Requests
  - Errors
  - Latency
  - Throttles

Lambda Execution Models

Lambda Permissions Model

Resource-based policies:

- Allow an AWS service to invoke your function on your behalf
- Grants usage permission to other AWS accounts
- Can enable sharing a function across an AWS Organization
- Support for Attribute-Based Access Control (ABAC)

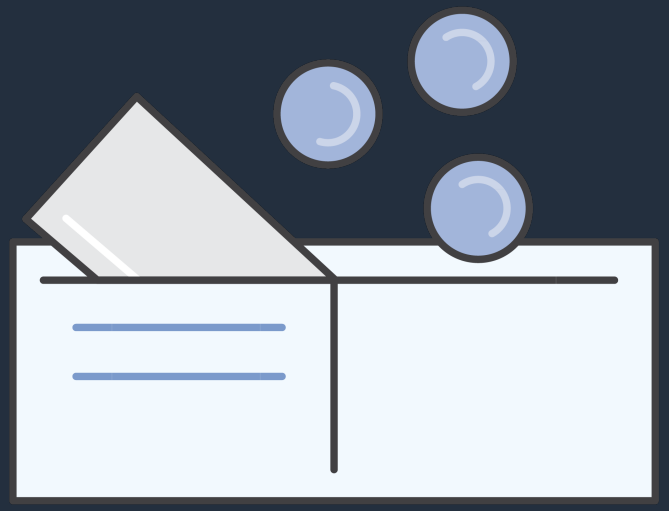
- For example, "Actions on Bucket X can invoke Lambda function A"

Execution role:

- An IAM role that grants the function permission to access AWS services and resources
- You provide this role when you create a function

- For example, "Lambda function A can read from DynamoDB table X"

Fine-grained Pricing



Free Tier

1M requests and 400,000 GB-sec of compute.

Every month, every customer.

- Buy compute time in 1ms increments
- Low request charge
- No hourly, daily, or monthly minimums
- No per-device fees
- Never pay for idle
- x86 or ARM execution using Graviton
- Pricing tiers for bulk usage

Common AWS Lambda use cases

Web Apps

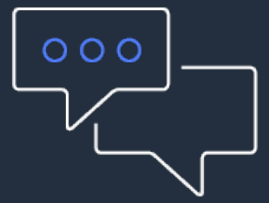
- Static websites
- Complex web apps
- Packages for Flask and Express

Backends

- Apps & services
- Mobile
- IoT

Data Processing

- Real time
- MapReduce
- Batch



Chatbots

- Powering chatbot logic

Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

aws

Customer use cases

The Seattle Times uses AWS Lambda for real time image data processing

Coca-Cola scaled from prototype to 10k machines in 100 days using AWS Lambda

Zillow uses AWS Lambda and Amazon Kinesis to load data into their data warehouse

AWS Lambda Best Practices

- Limit your function/code size
- 10,240 MB /tmp directory storage provided to each function
- Don’t assume function will reuse underlying infrastructure
  - But take advantage of it when it does occur
- You own the logs
  - Include details from service-provided context
- Create custom metrics
  - Operations-centric vs. business-centric
- Review Trusted Advisor Checks

Thank you!