



Raj Saha

[cloudwithraj.com](http://cloudwithraj.com)

 Cloud With Raj

   cloudwithraj

 [linkedin.com/in/rajdeep-sa-at-aws/](https://linkedin.com/in/rajdeep-sa-at-aws/)

---

Instructor Bio:

Sr. Solutions Architect @ 

Bestselling Udemy/Pluralsight author

Tech Advisor of crypto startup

Public speaker and guest lecturer

Author of multiple official AWS blogs

YouTuber with 30K subscribers

Previously - Distinguished Cloud Architect @Verizon

*Opinions are my own*

Copyright © Rajdeep Saha - All Rights Reserved

# SECTION1: SYSTEM DESIGN BASICS

# Microservices

# The monolith

“...a **single-tiered software application** in which the user interface and data access code are combined into a **single program** from a **single platform**.”

- Wikipedia

# Monolith is not the bad guy!

## Pros:

- At first...
  - Simple
  - No over-engineering
- Single code base
- Resource efficient at small scale

## Cons:

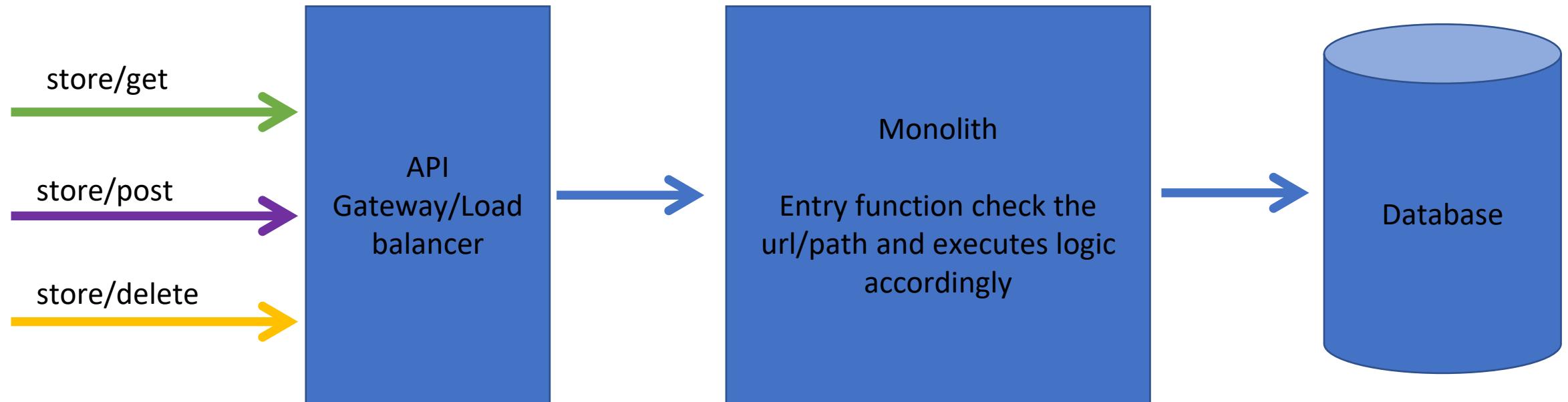
- Modularity is hard to enforce as app grows
- Scaling is a challenge
- All or nothing deployment
- Long release cycles
- Slow to react to customer demand

# Can you use API with Monolith?

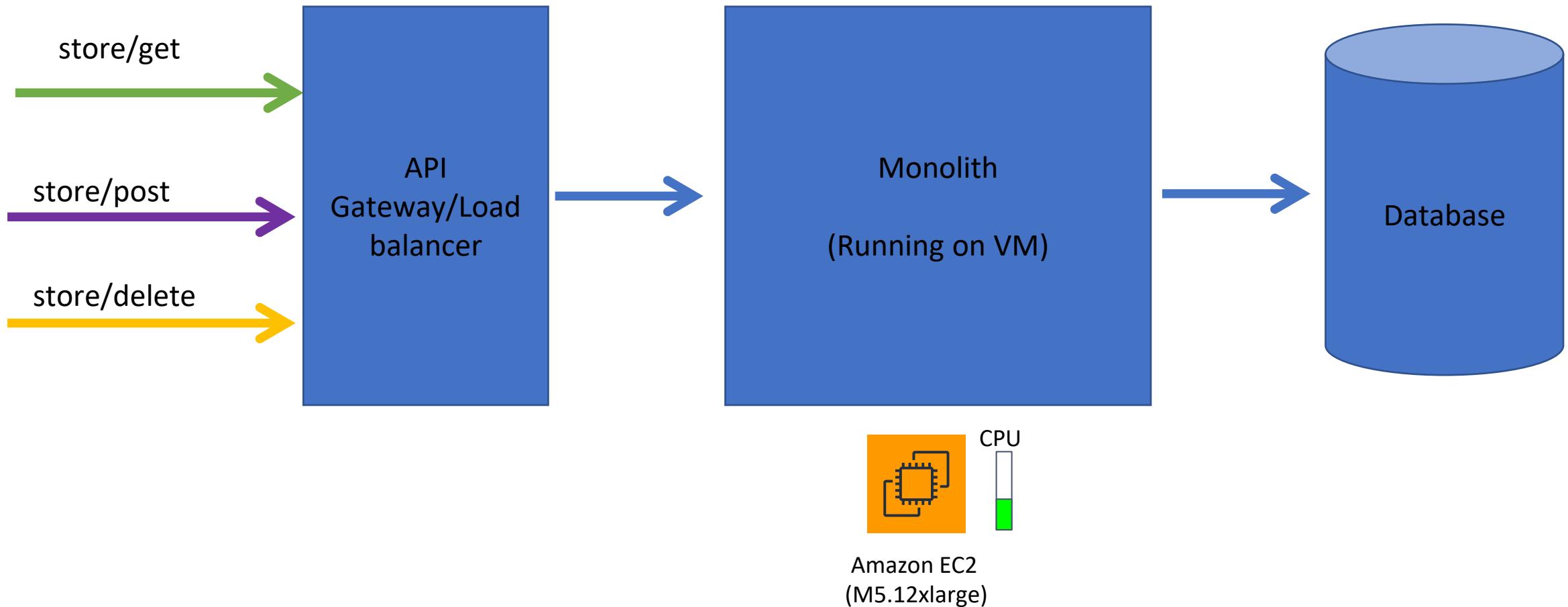
# Absolutely

APIs does NOT equal microservices

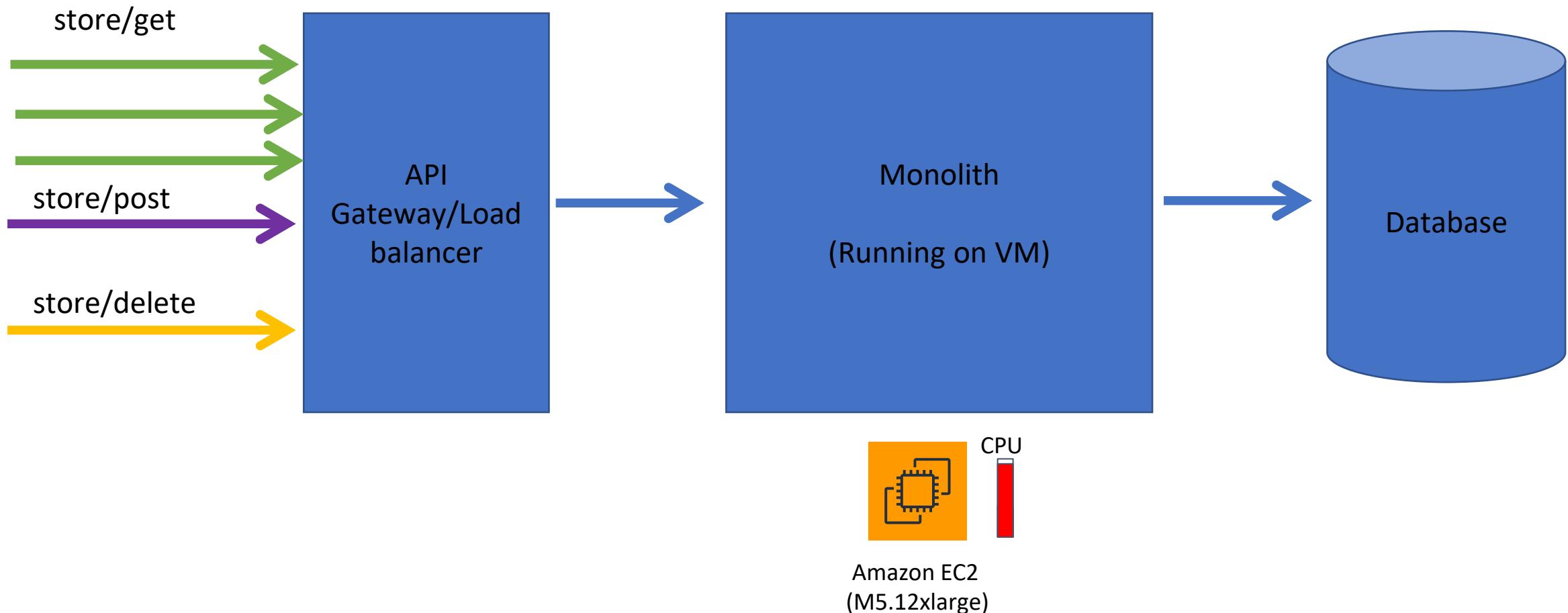
# APIs in Monolith



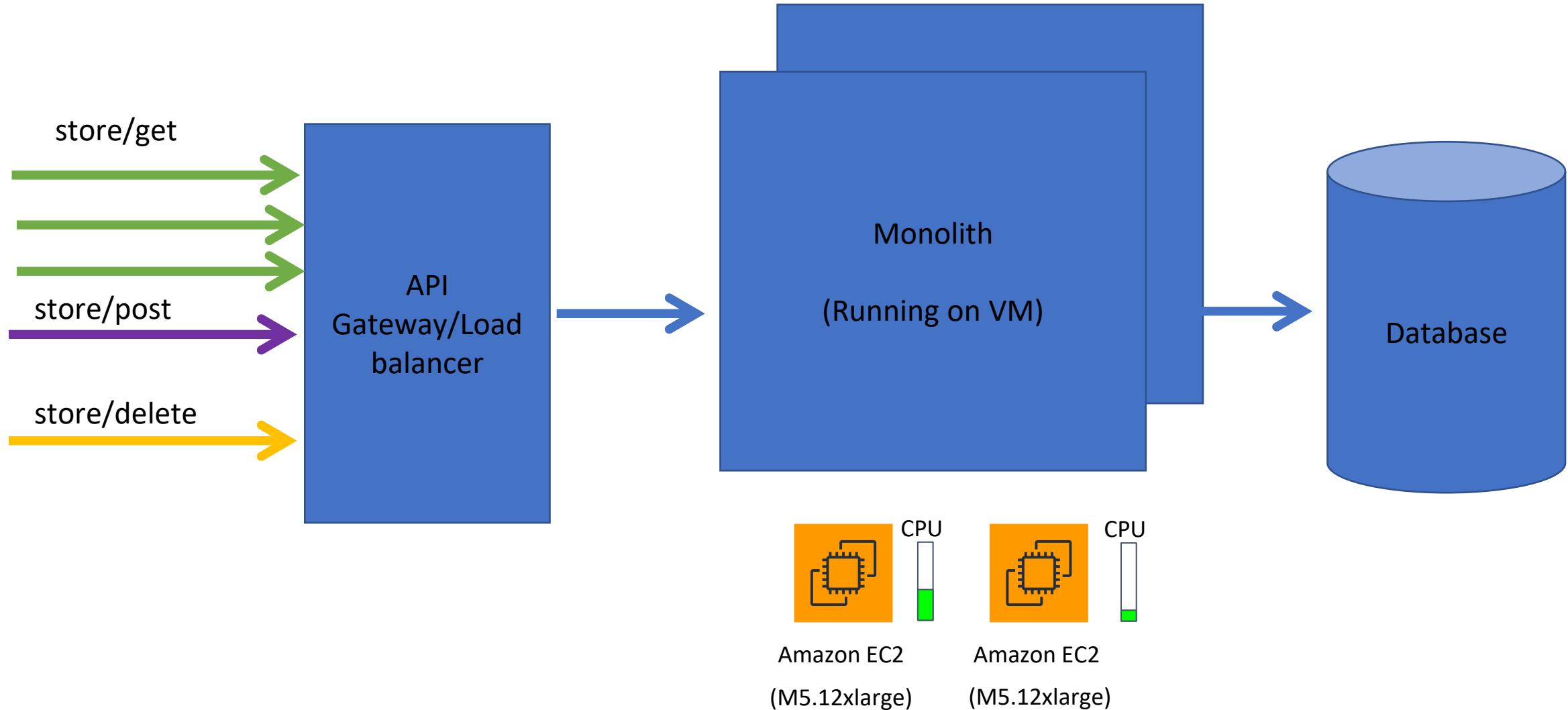
# Issue of Scaling



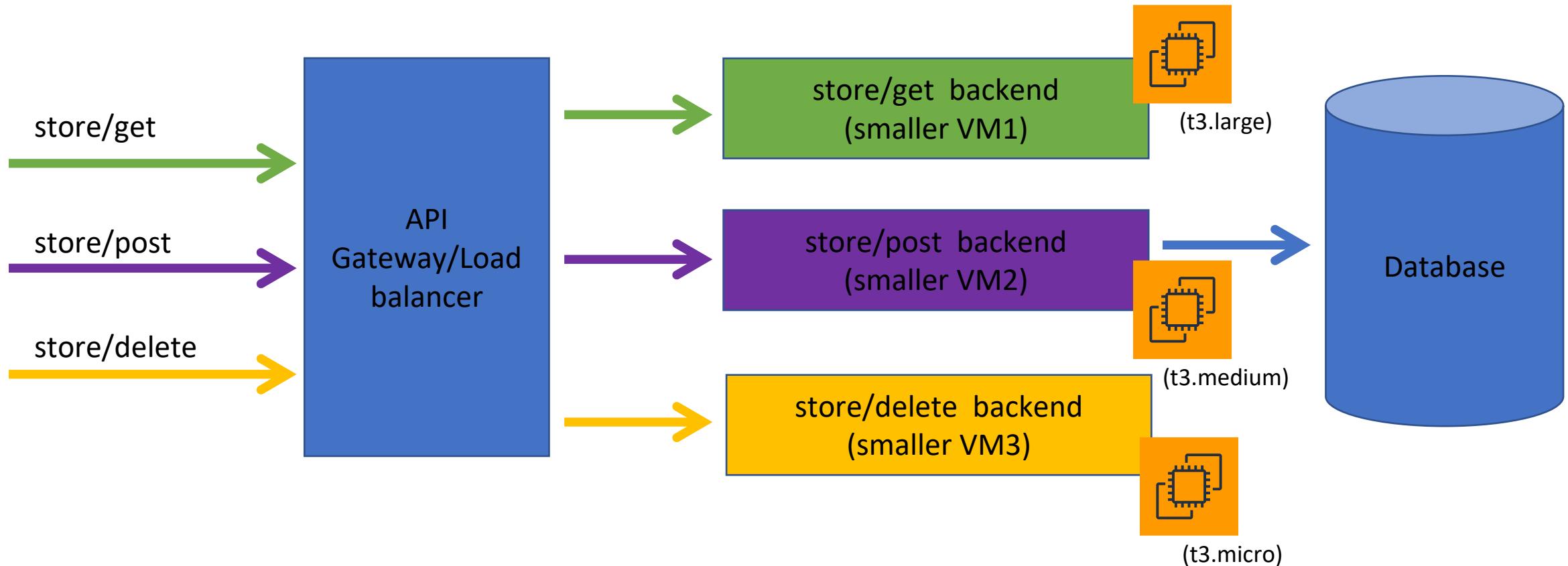
# Issue of Scaling



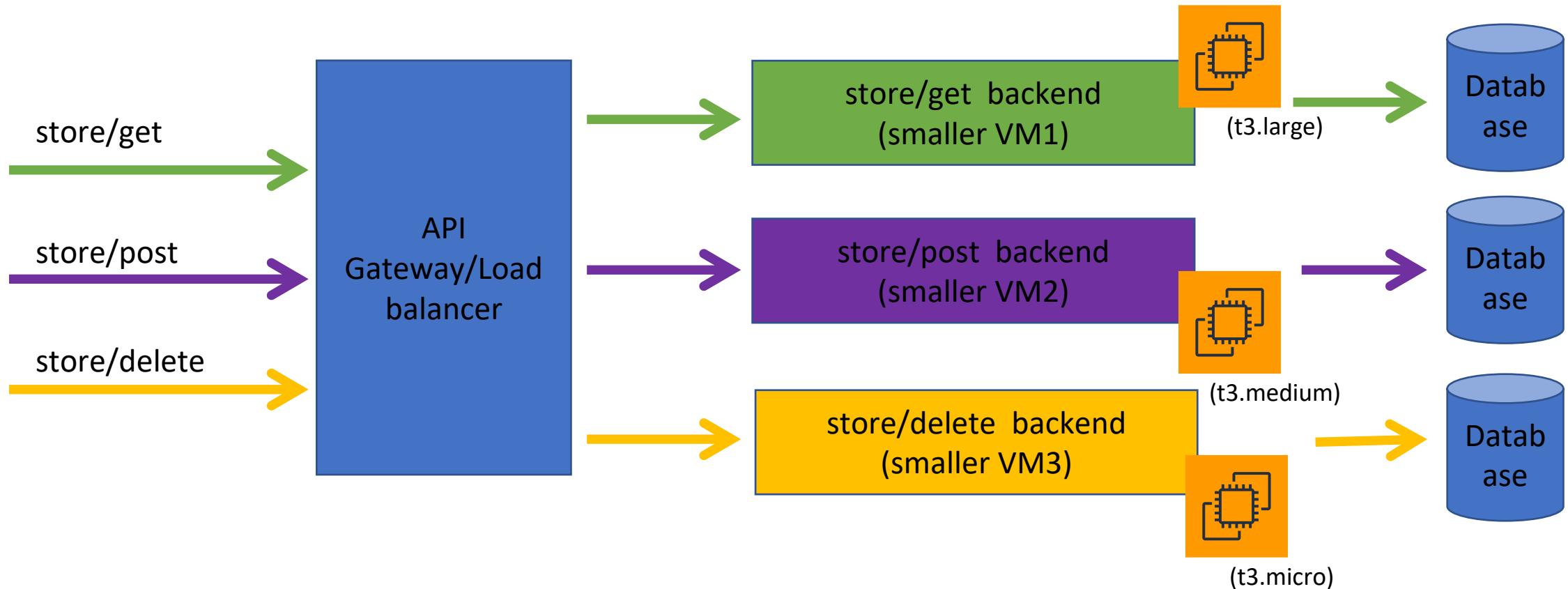
# Entire Monolith Need to Scale



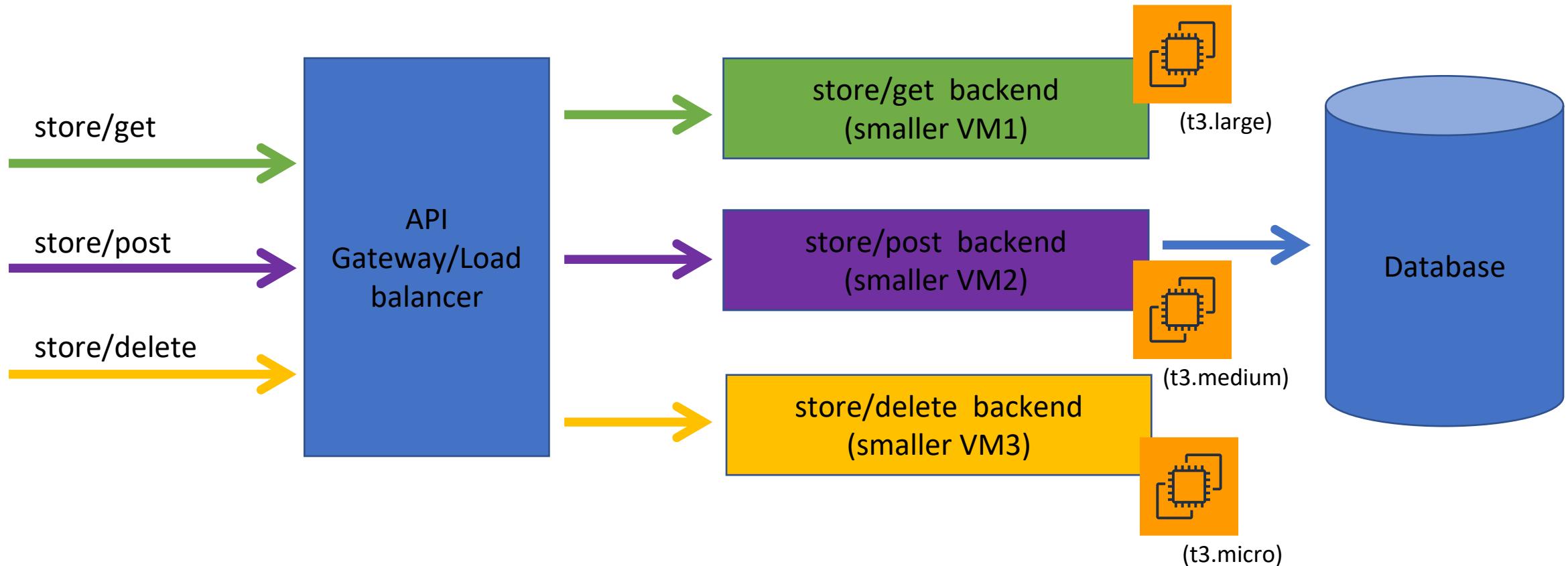
# APIs in Microservice



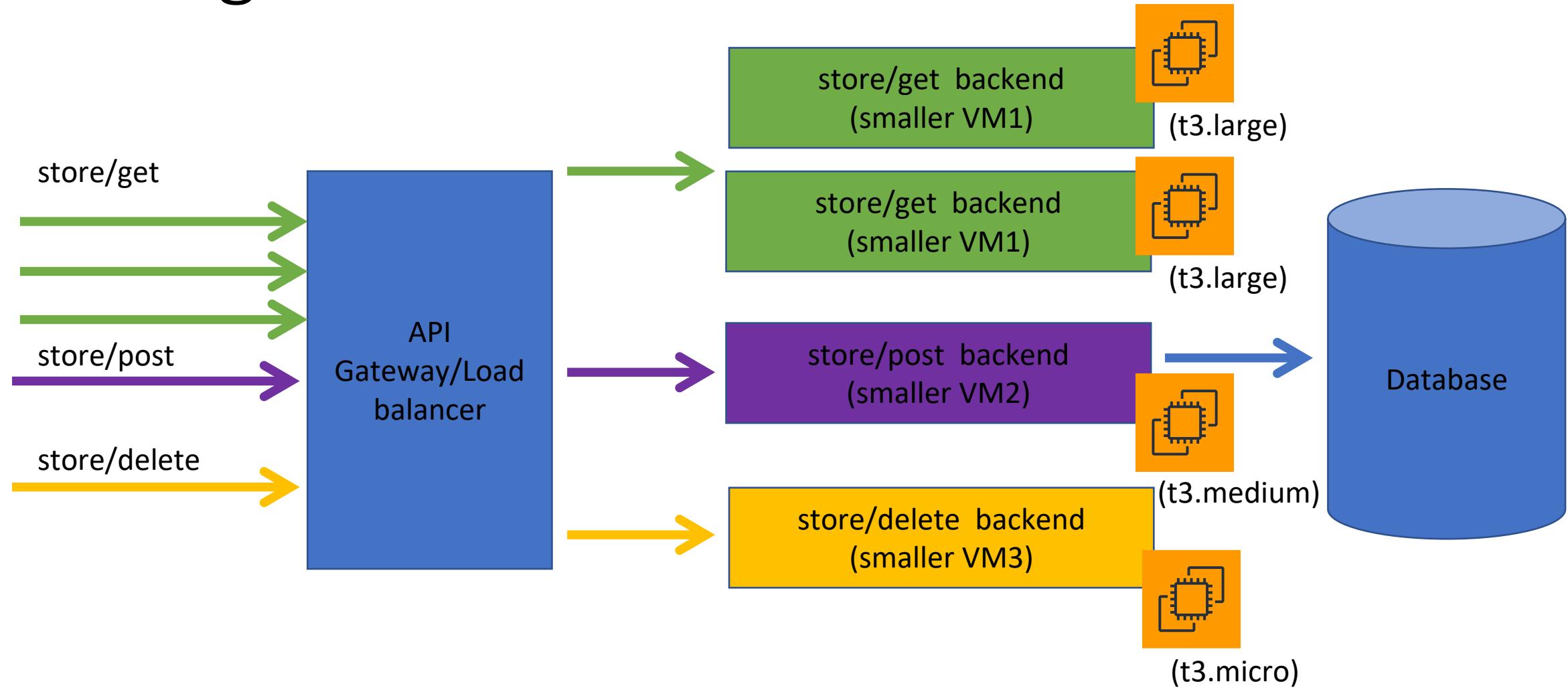
# APIs in Microservice



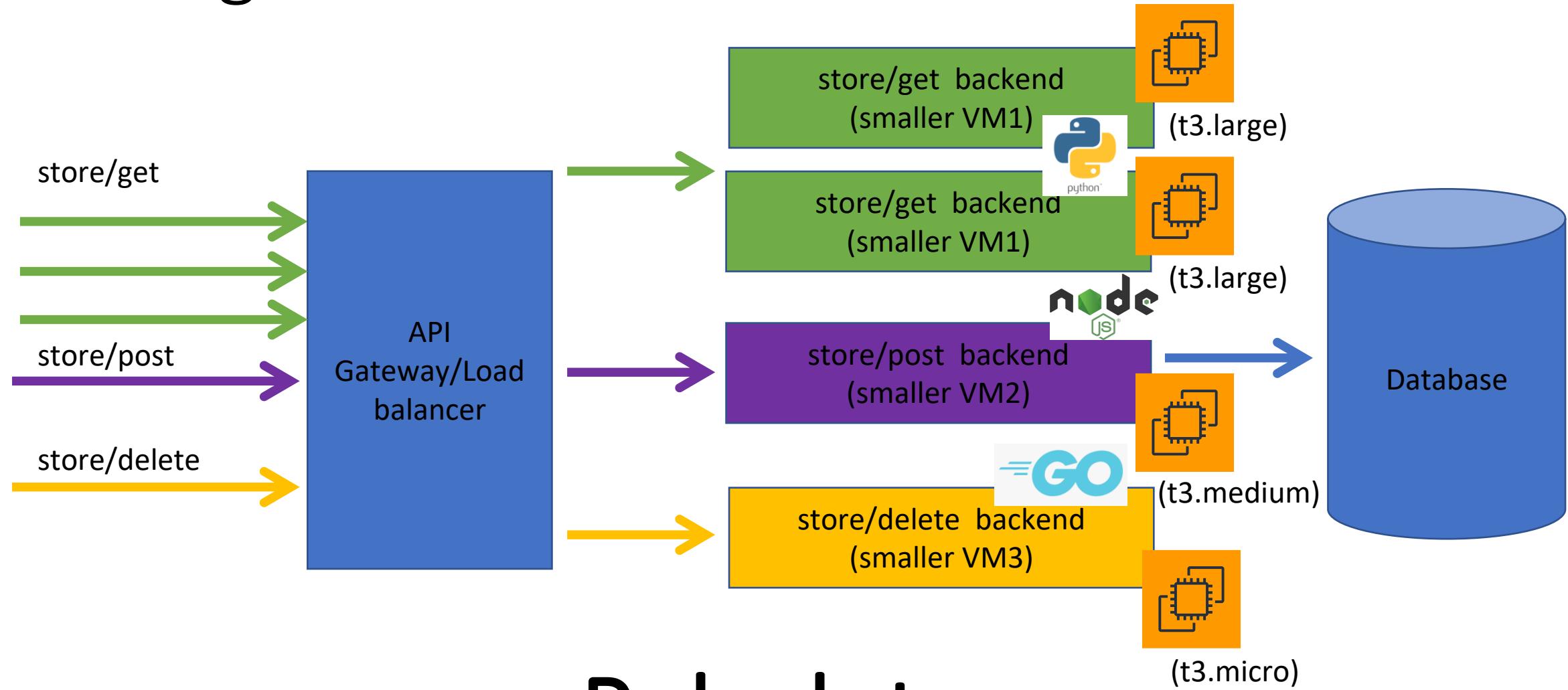
# APIs in Microservice



# Scaling APIs in Microservice



# Scaling APIs in Microservice



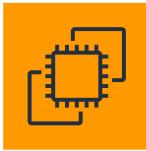
# Characteristics of microservice architectures

- Independent
  - Scaling
  - Governance
  - Deployment
  - Testing
  - Functionality

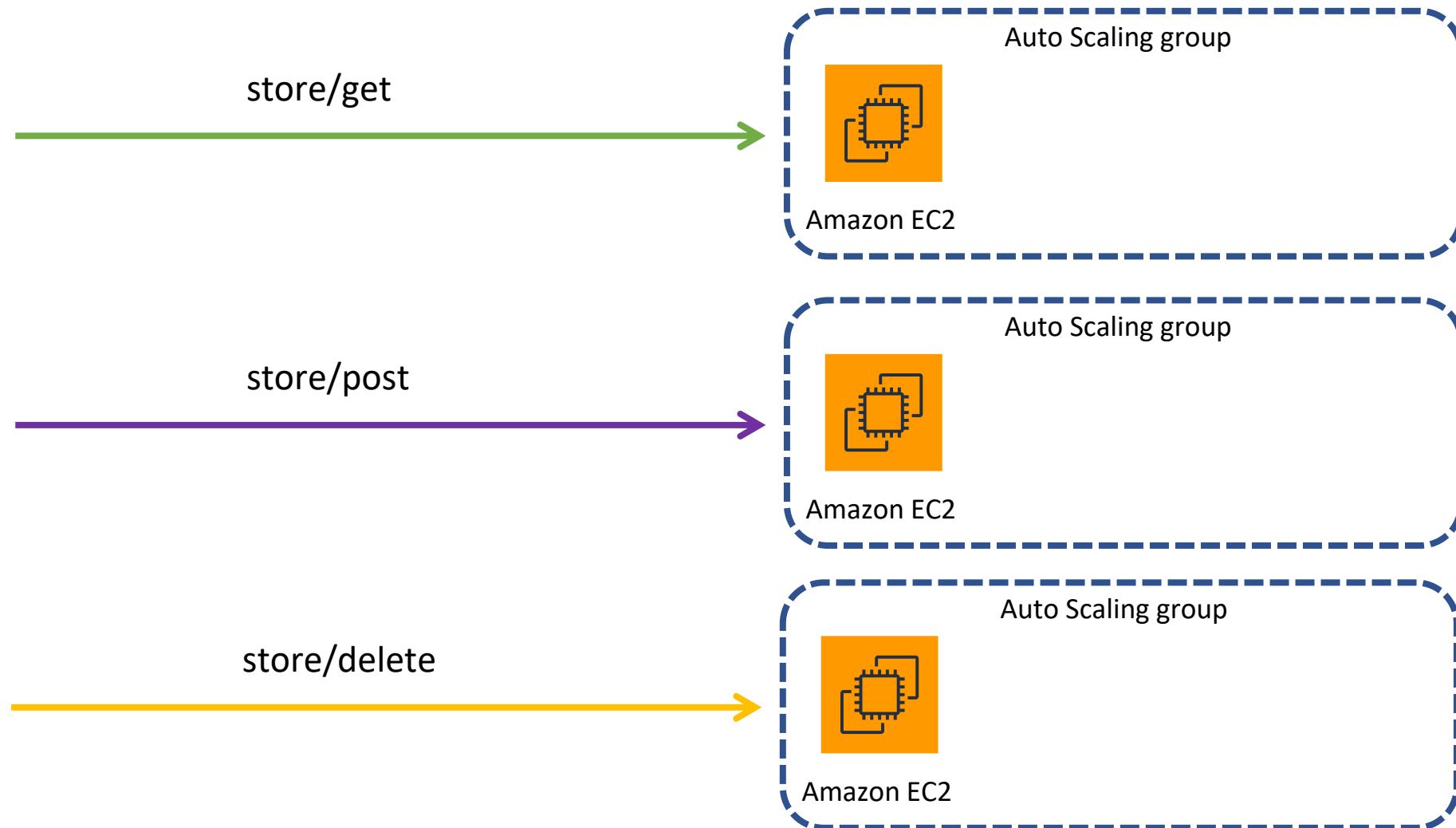
Important - Not required to follow every characteristic

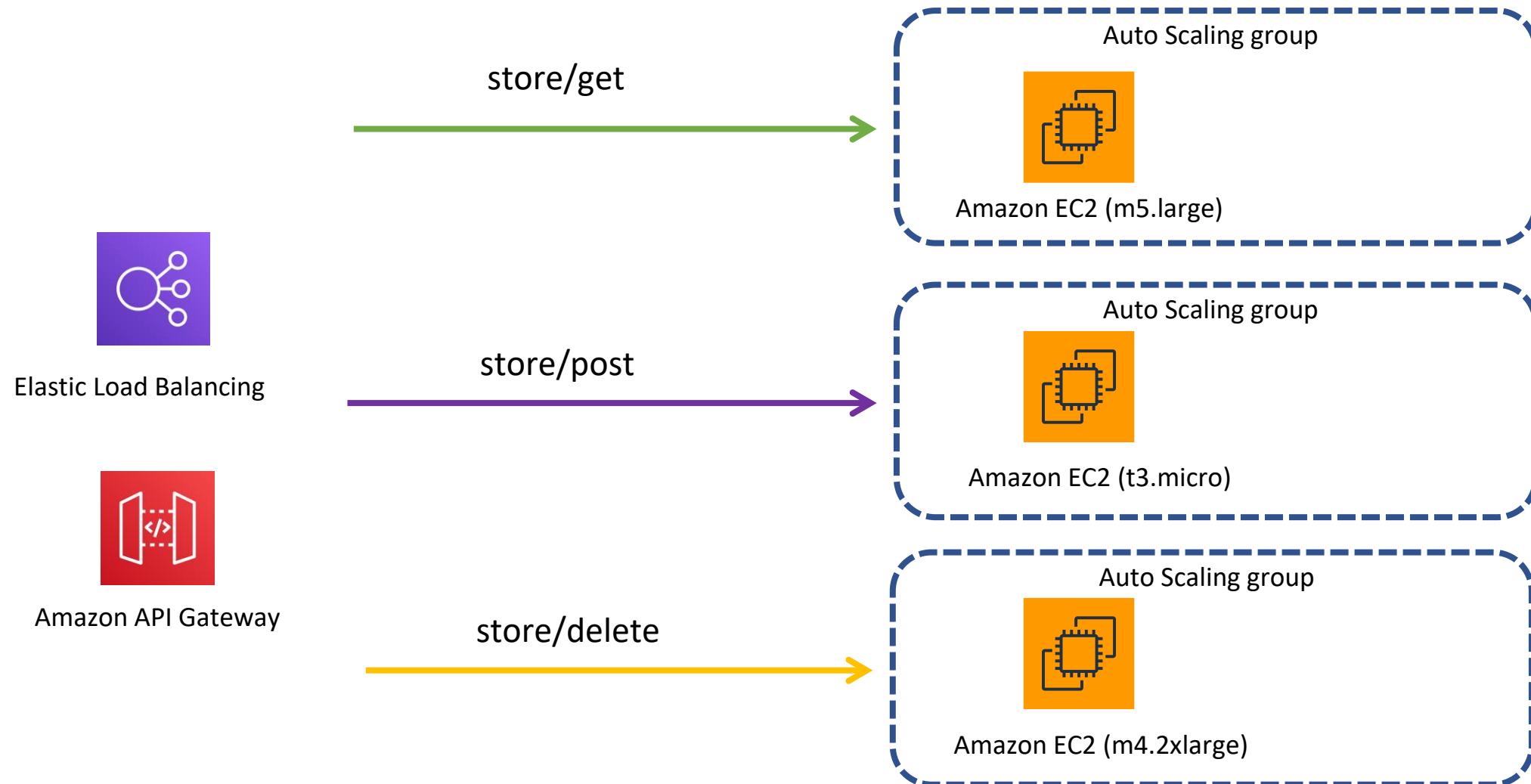
# Deploying microservices in AWS

# The answer to everything!



Amazon EC2



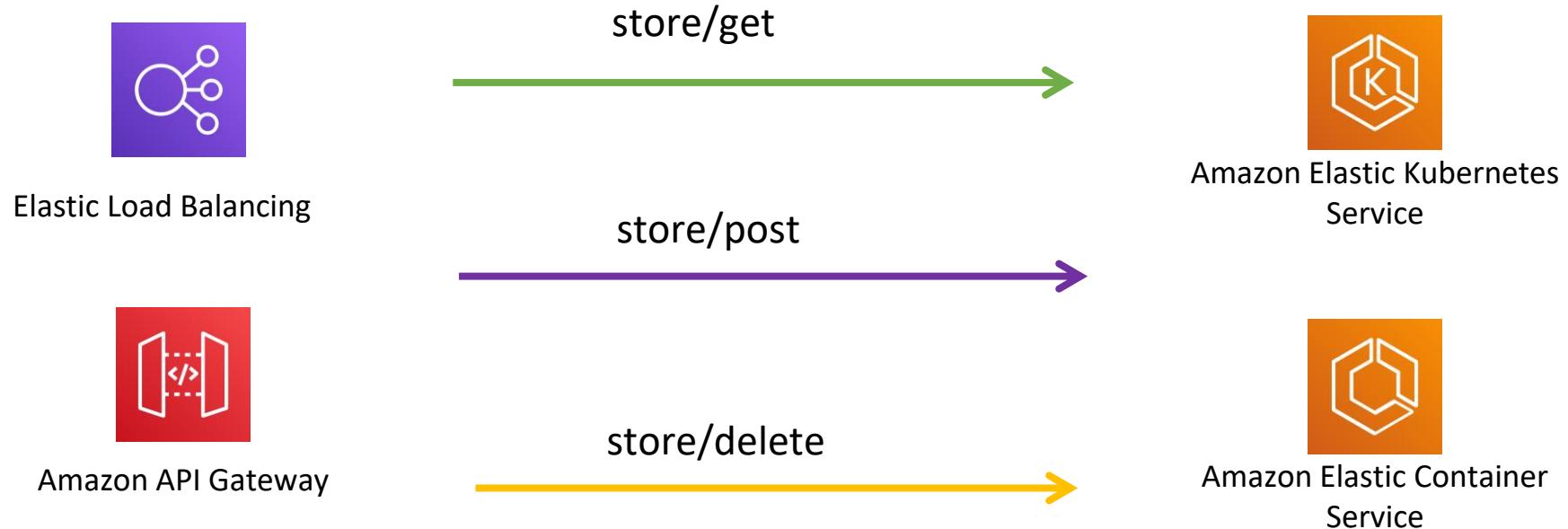


# Serverless



Lambda scales automatically

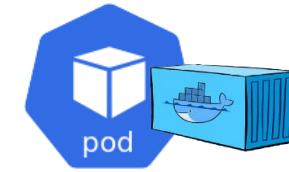
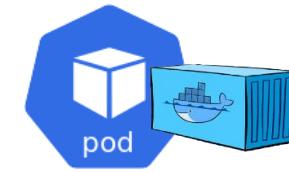
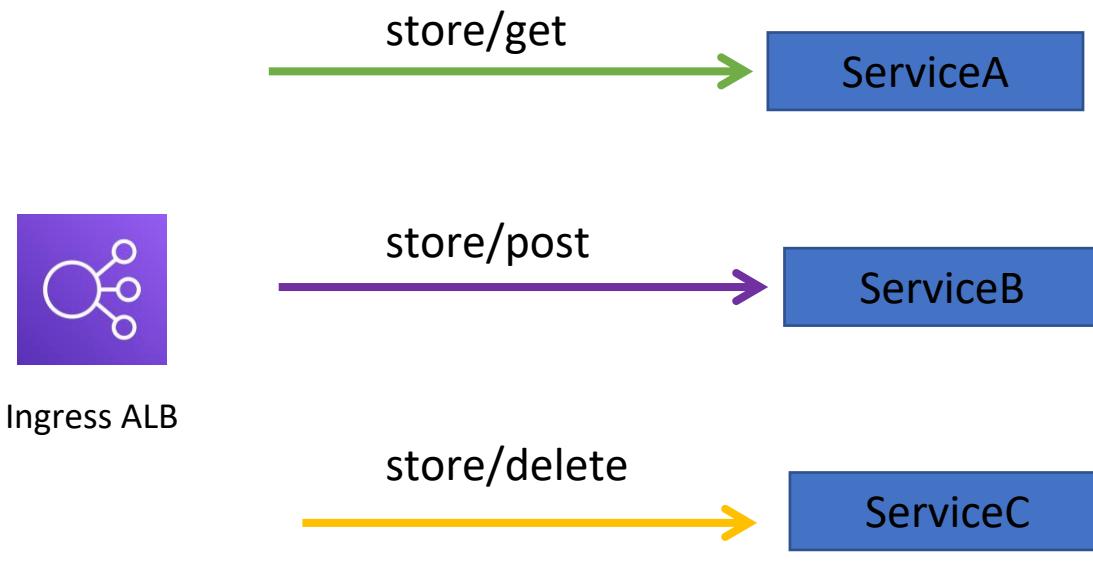
# Container



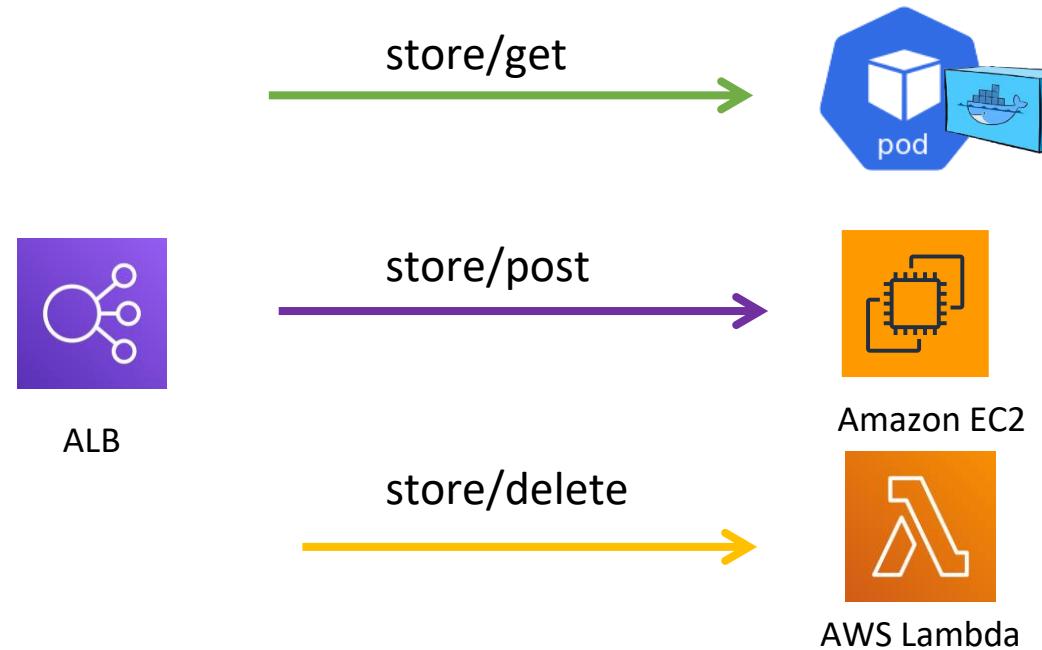
# Kubernetes



Amazon Elastic Kubernetes  
Service



# Mix and Match!

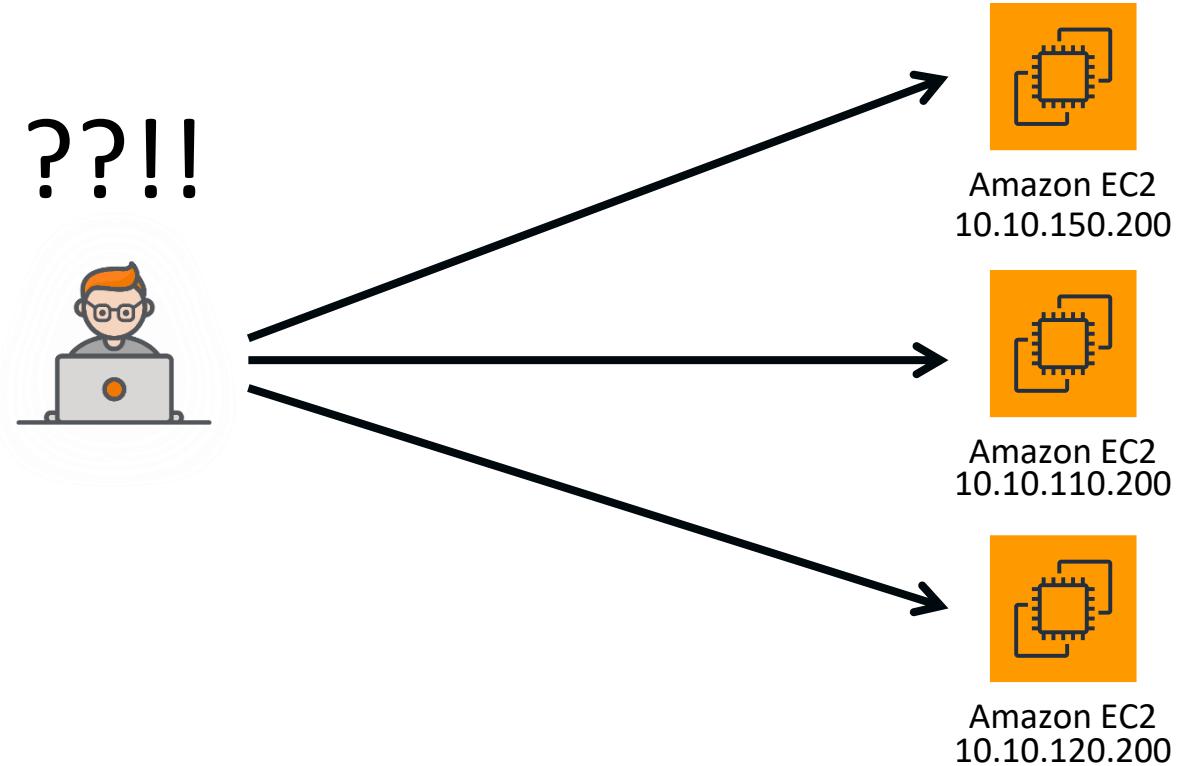


# Load Balancer

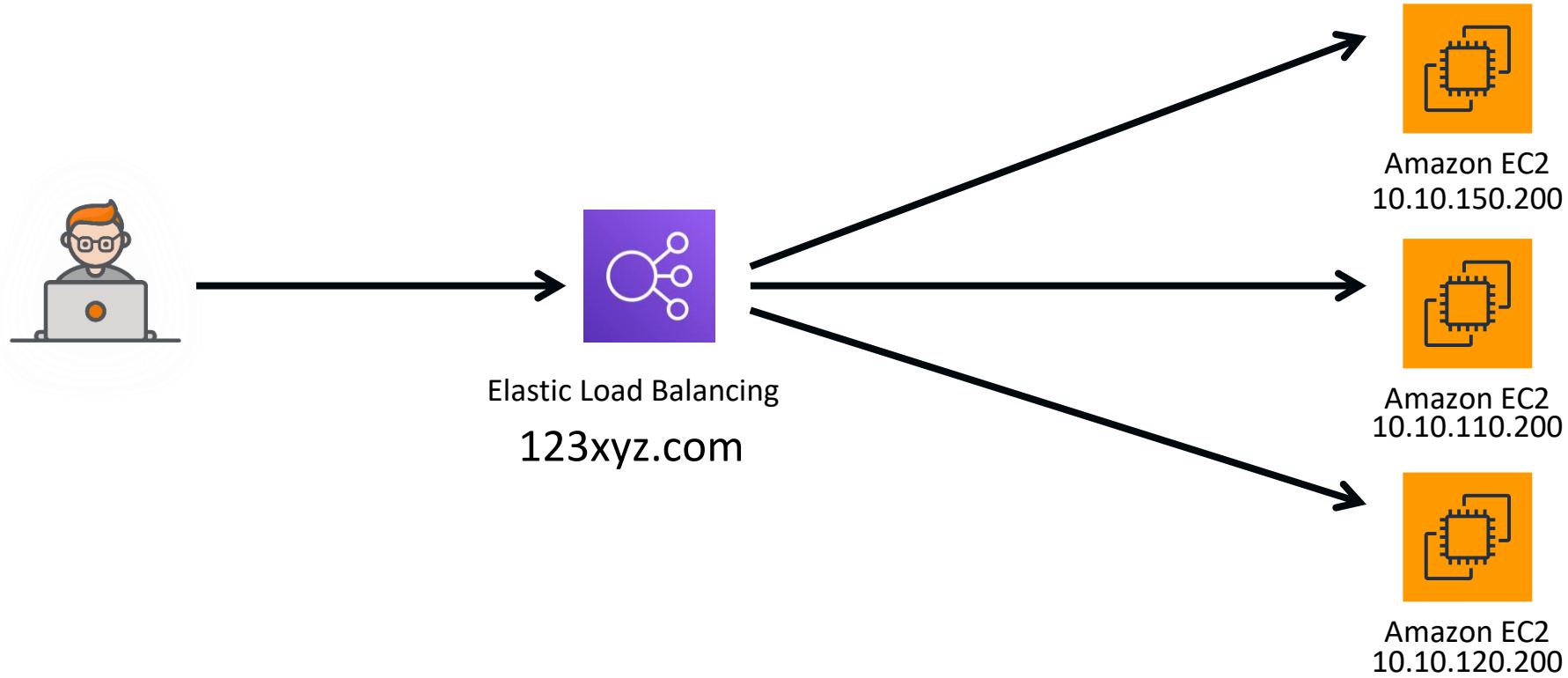
# Your Application



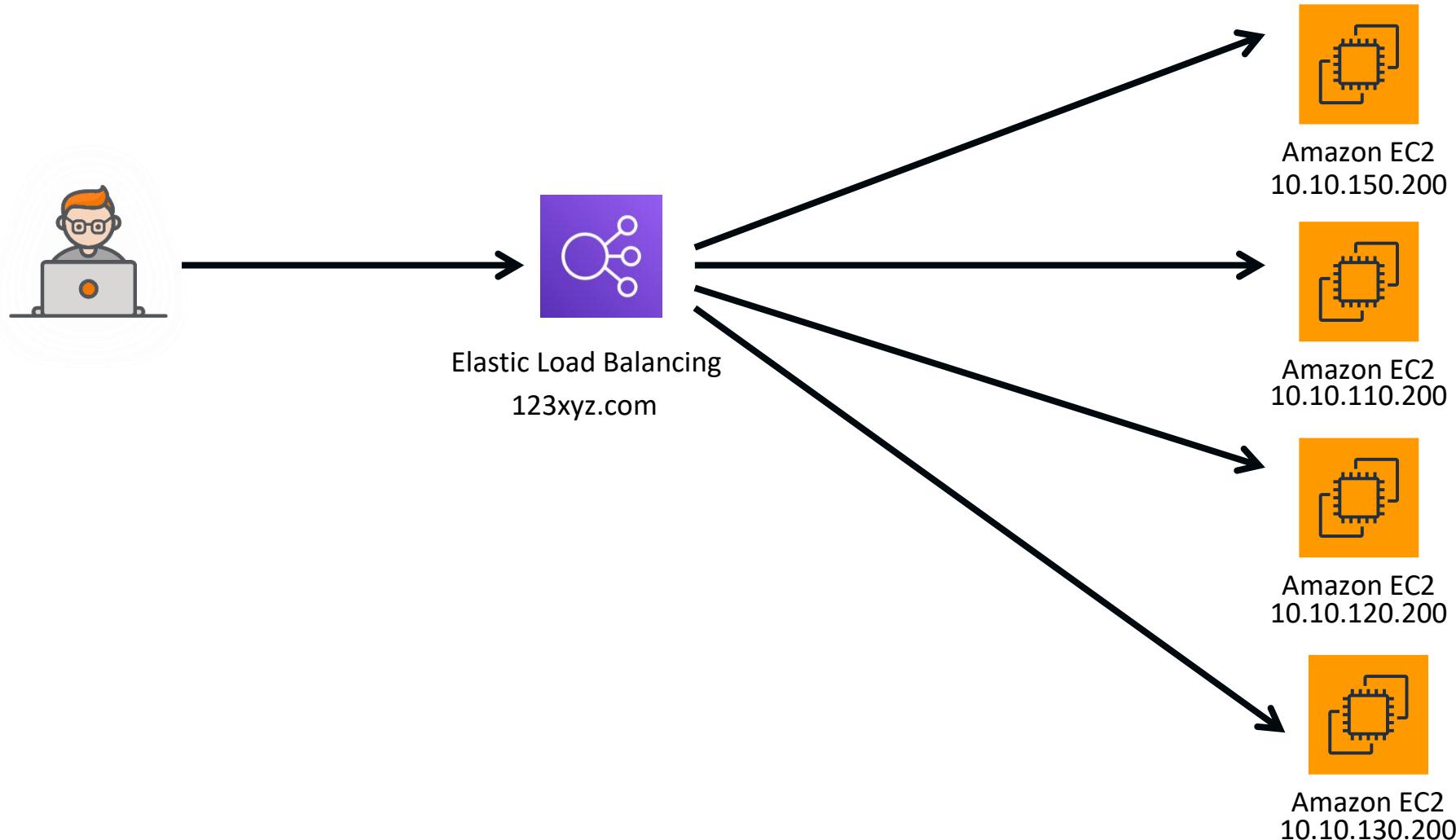
# Your Application



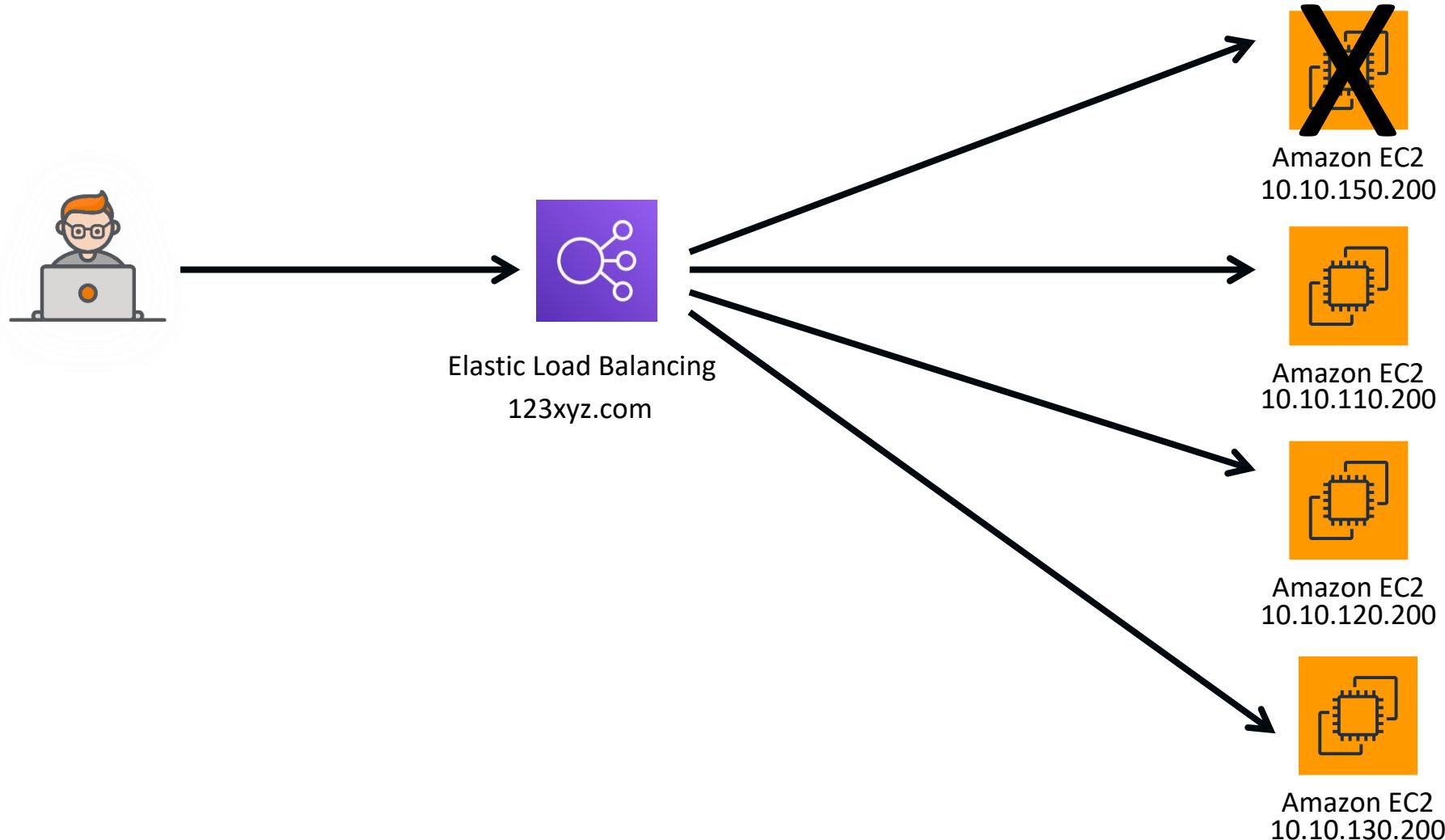
# Your Application



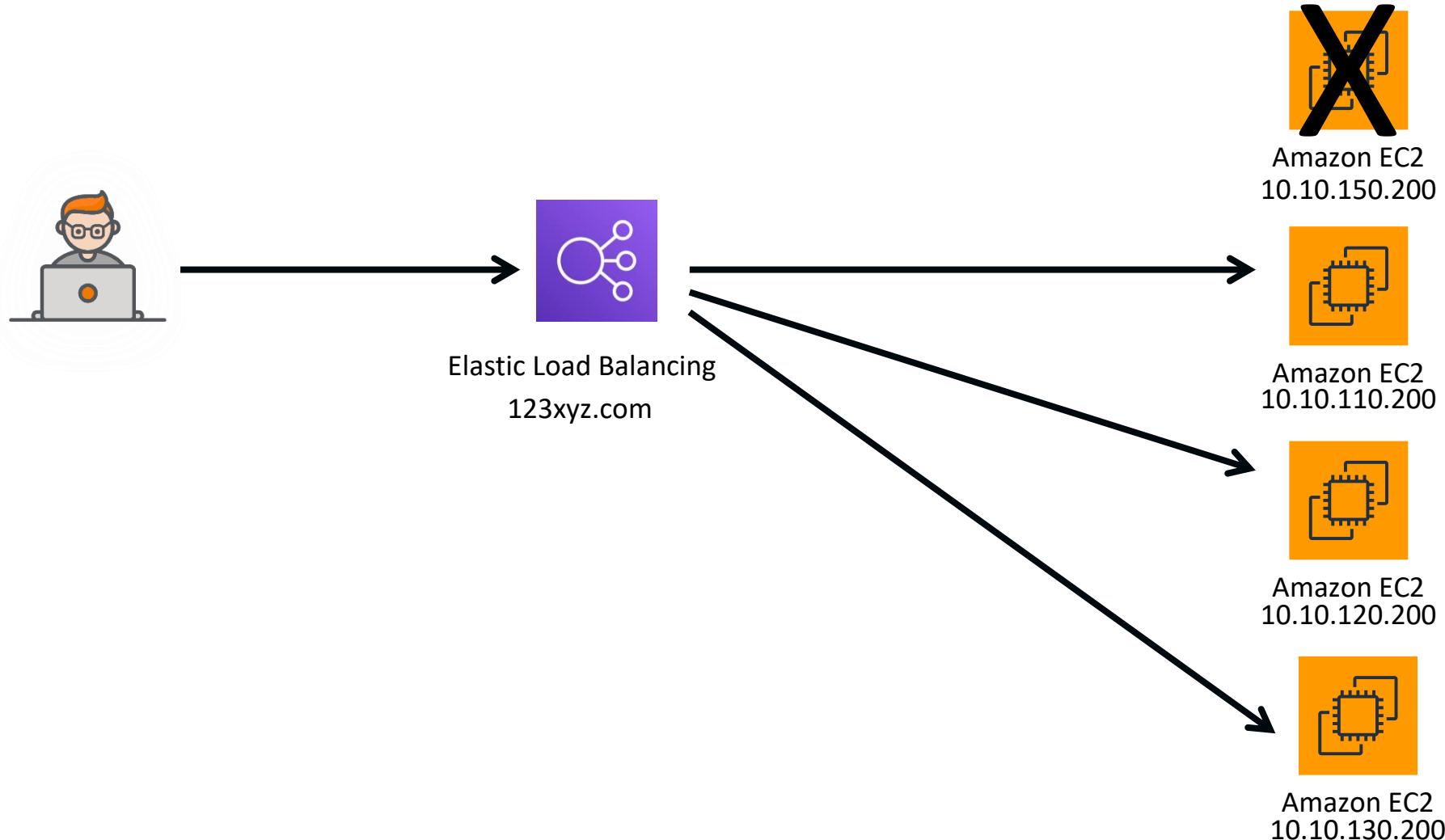
# Your Application



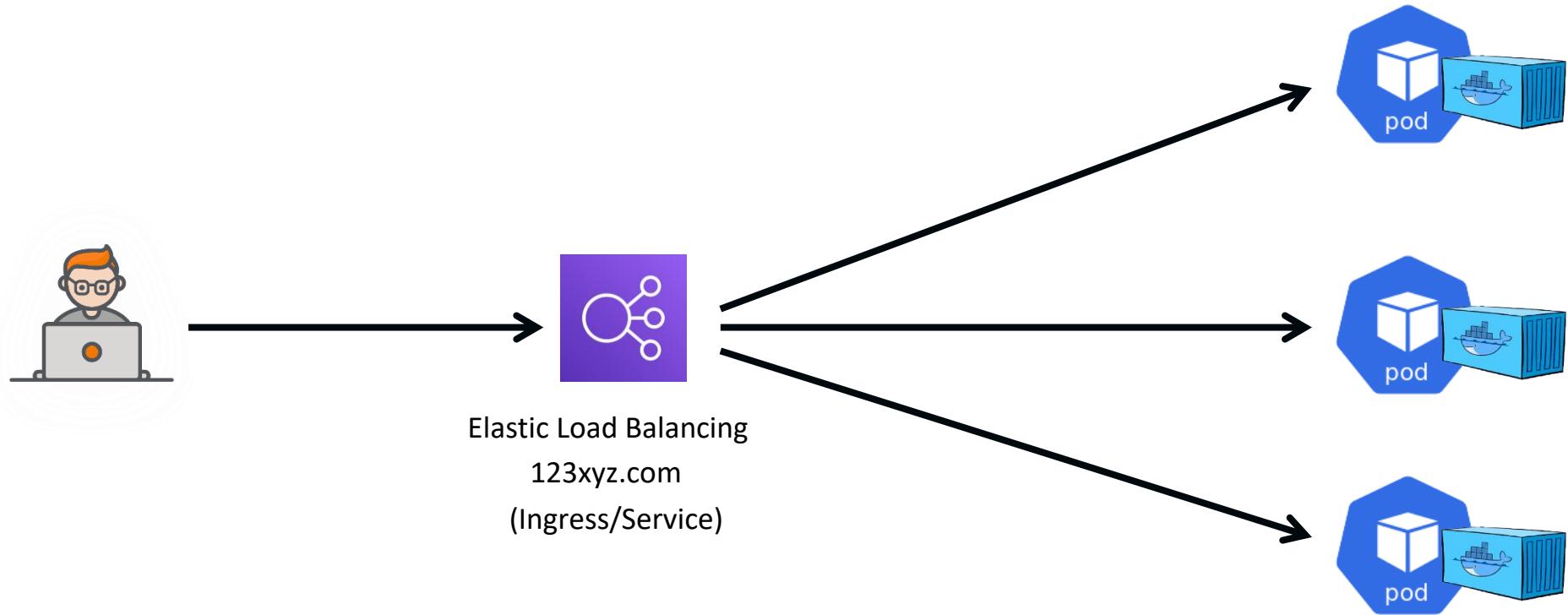
# Your Application



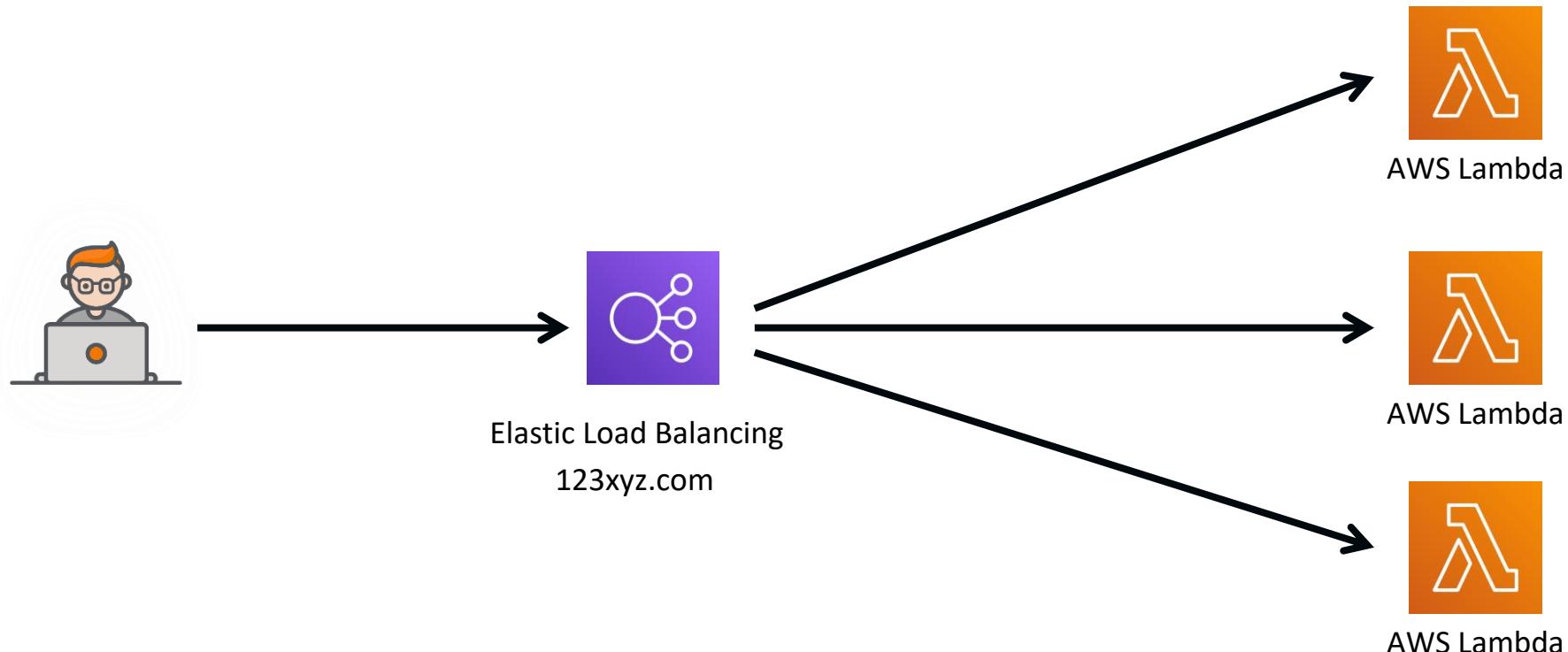
# Your Application



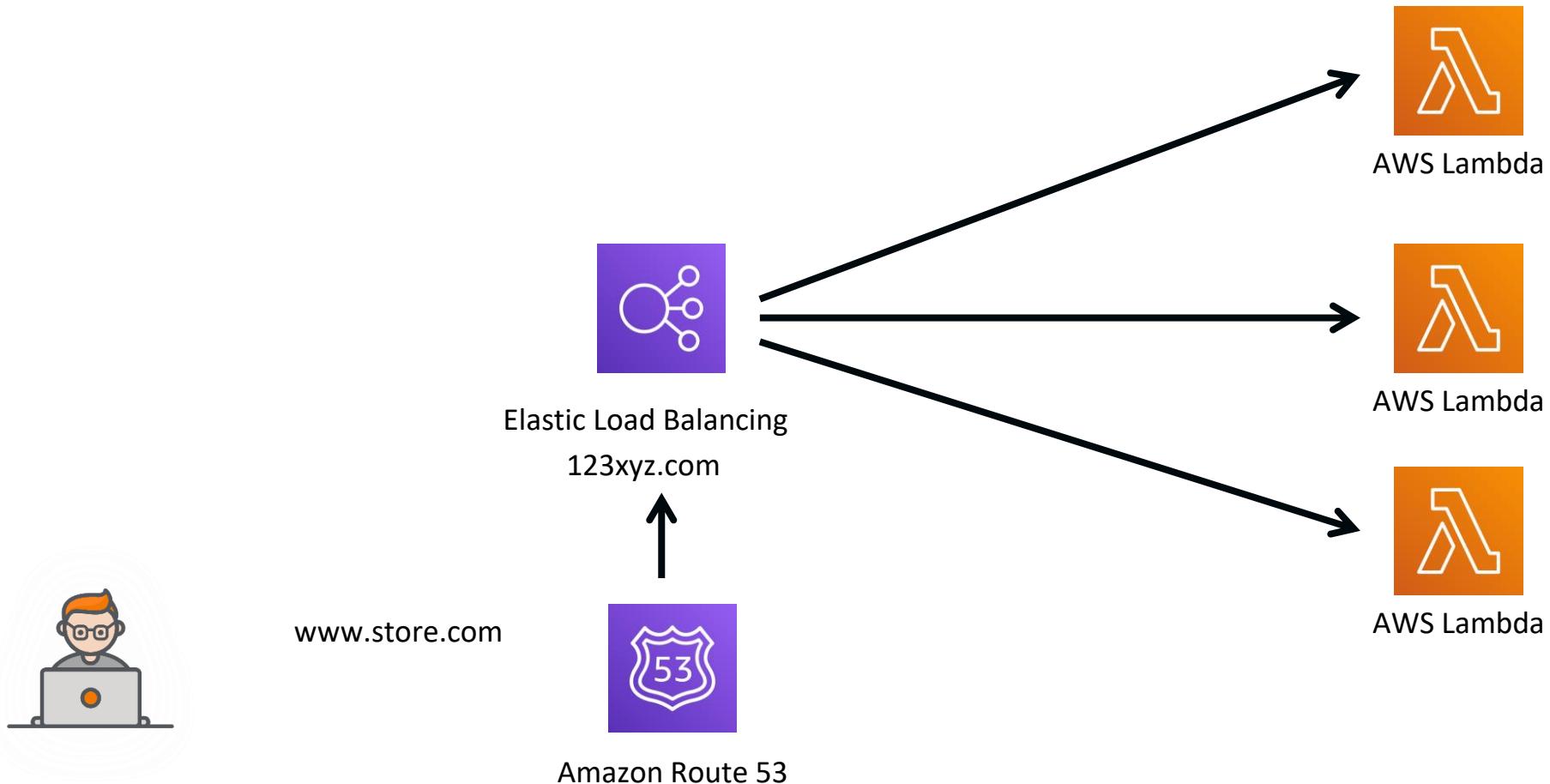
# Your Application



# Your Application



# Your Application



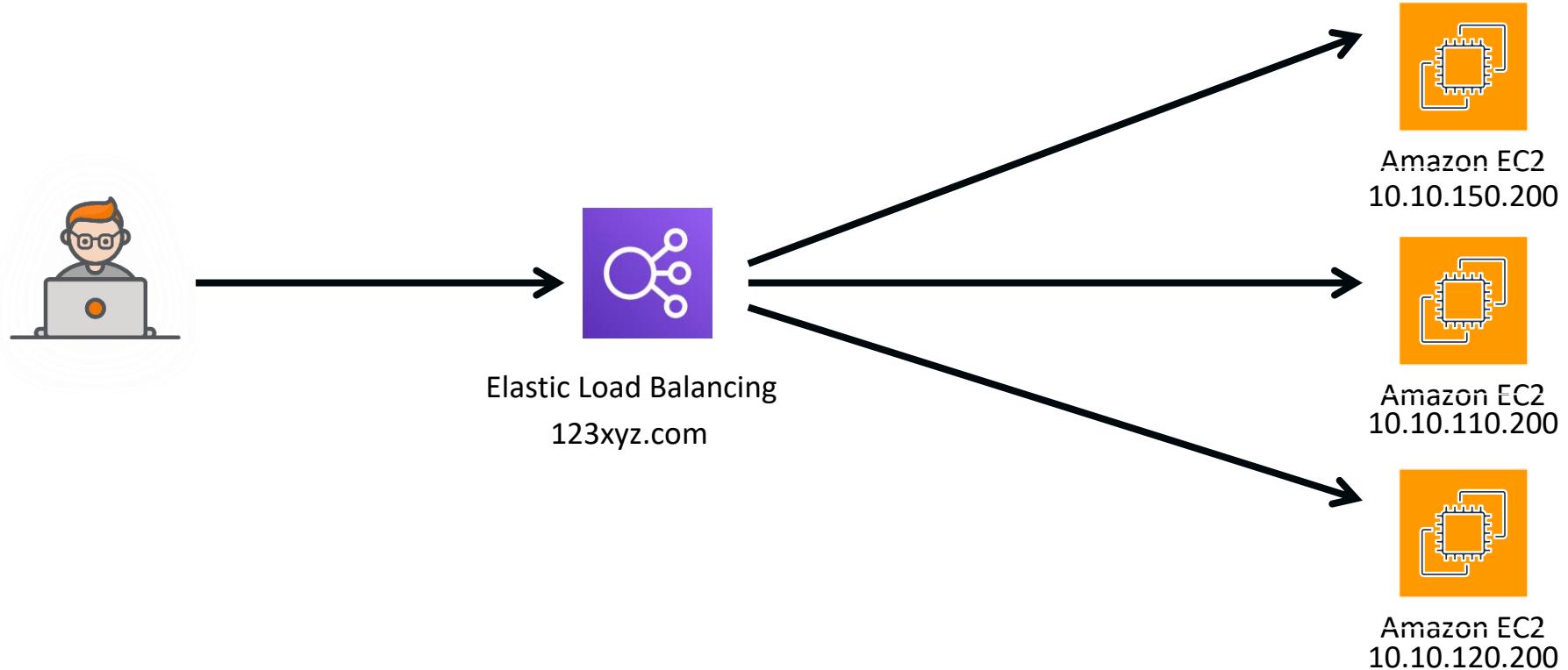
# Load Balancer

- Automatically distributes incoming traffic across multiple targets
- Monitors health of targets
- Integrates with SSL
- “Elastic”

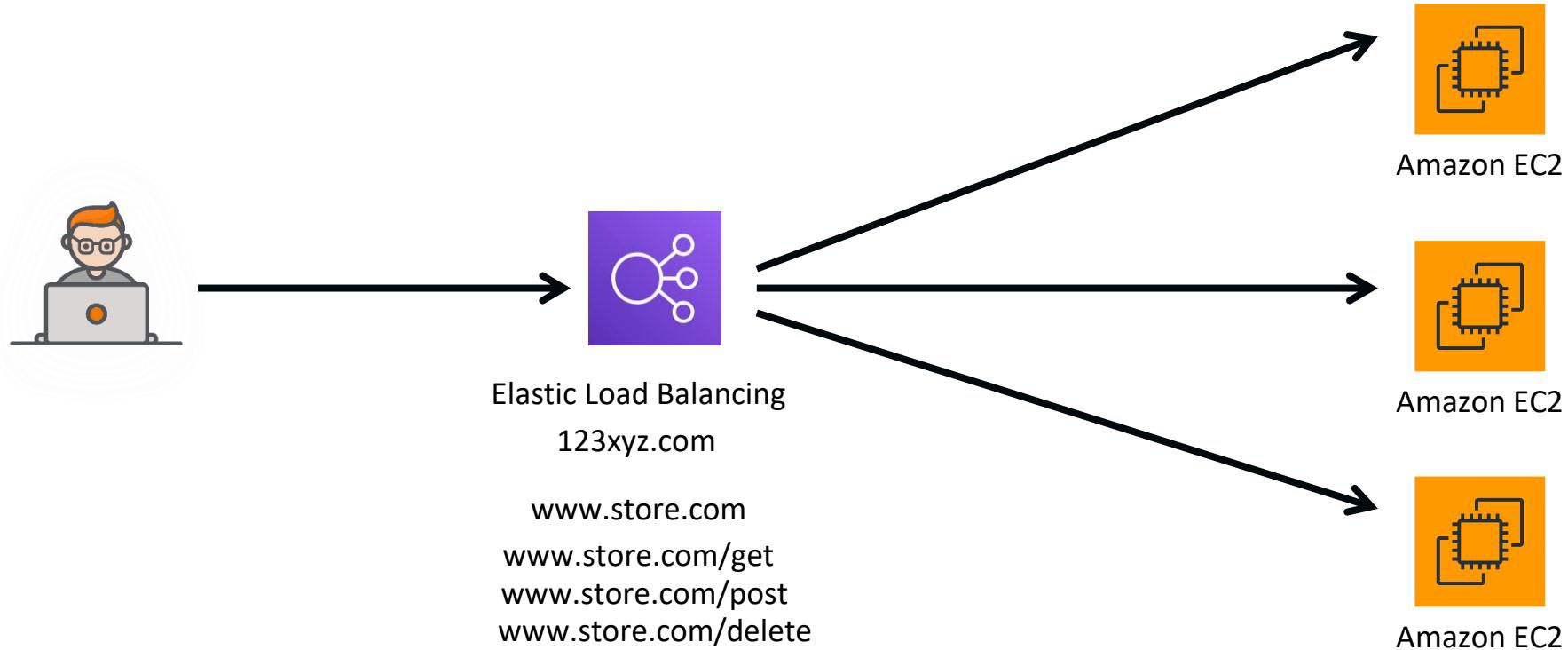
# Types of Load Balancer

- Application Load Balancer
- Network Load Balancer

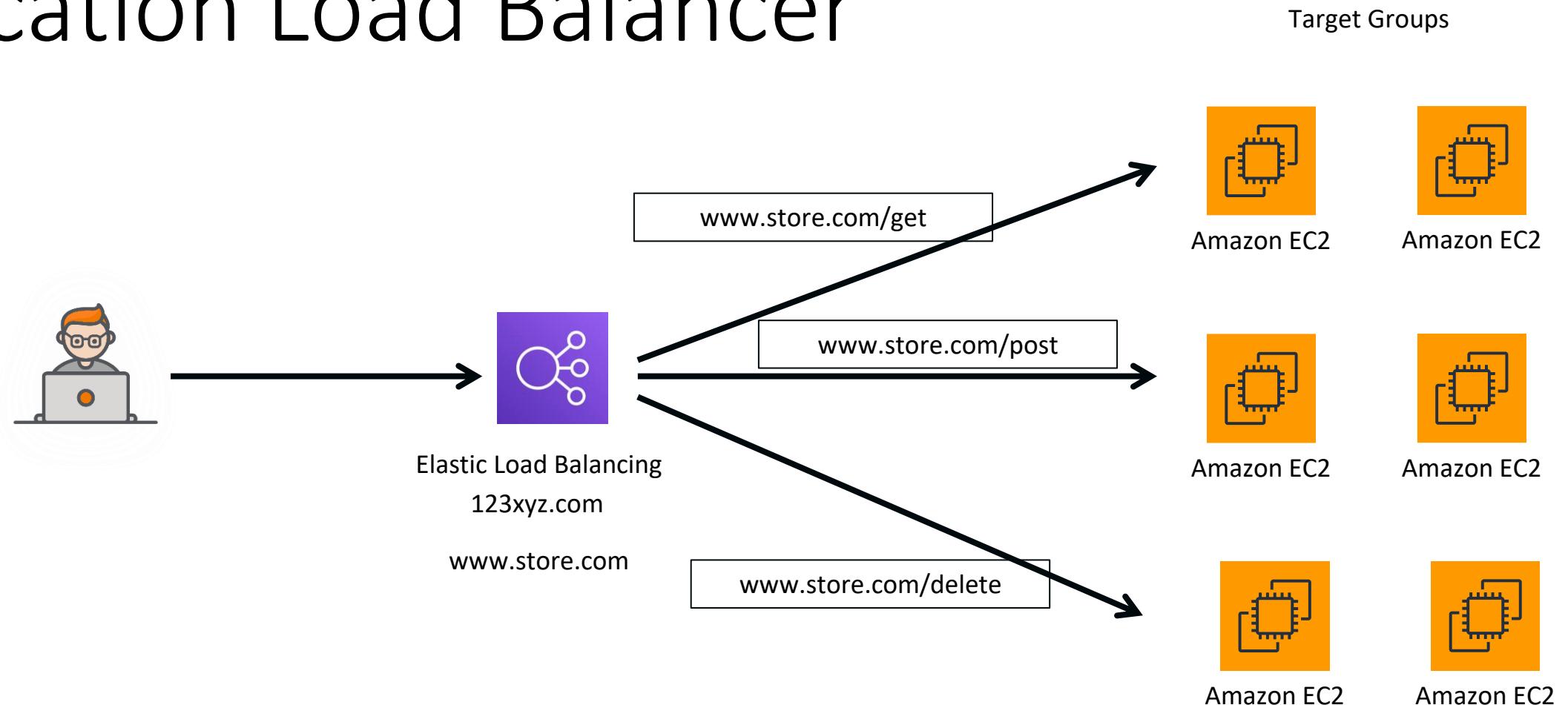
# Application Load Balancer



# Application Load Balancer



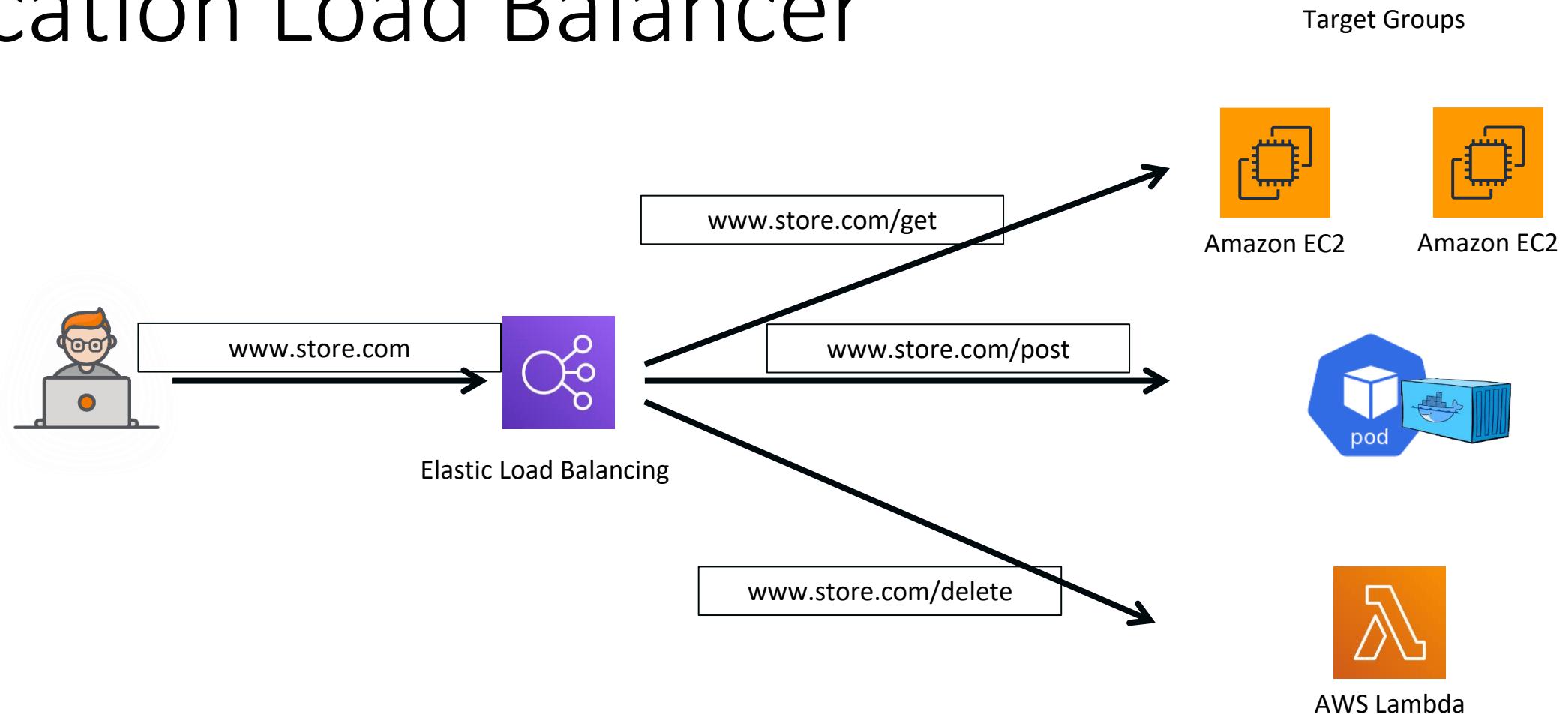
# Application Load Balancer



# Application Load Balancer

- Operates on OSI Layer 7
- Routes traffic based on url path
- Validates and terminates SSL
- Sticky session

# Application Load Balancer



# Network Load Balancer

- Operates on OSI Layer 4
- Routes traffic based on protocol and port of incoming traffic
- SSL passthrough

# ALB or NLB?

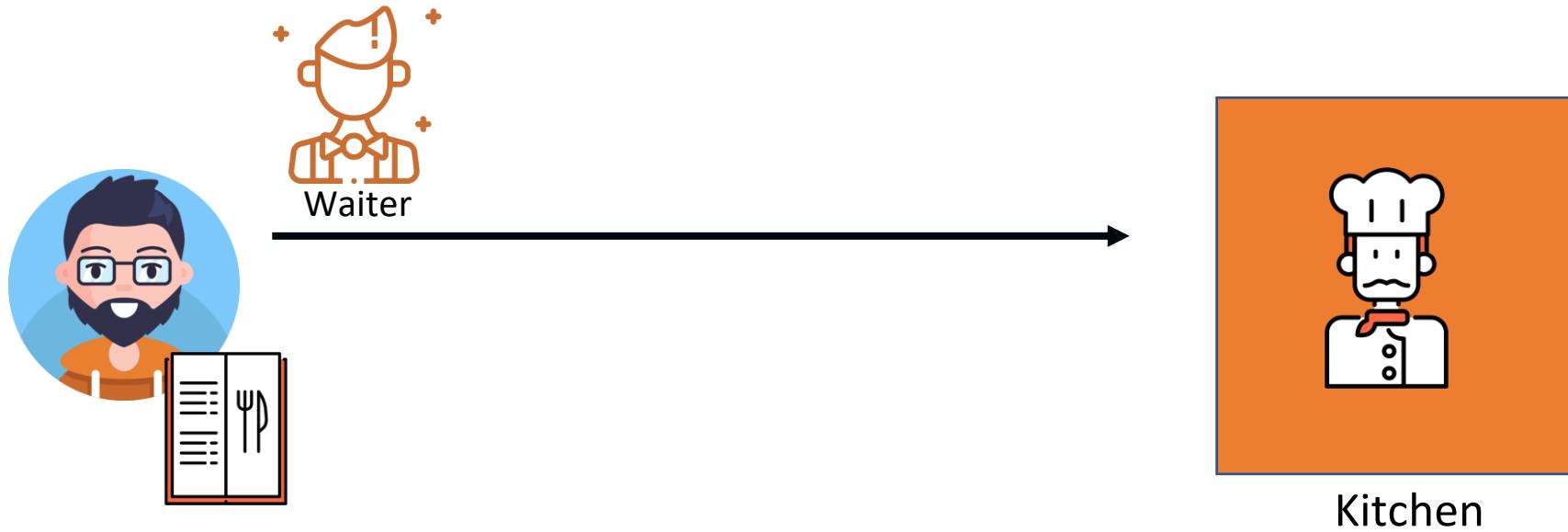
- NLB handles spiky traffic better
  - ALB handles consistent high traffic better
- NLB exposes static IP address
  - ALB needs Global Accelerator
- Influenced by choices
  - API Gateway REST API Private integration with NLB with Private Link
  - NLB supports EC2 instance and IP address as backend target group
  - ALB supports EC2, IP address, and Lambda

# API – What and Why?

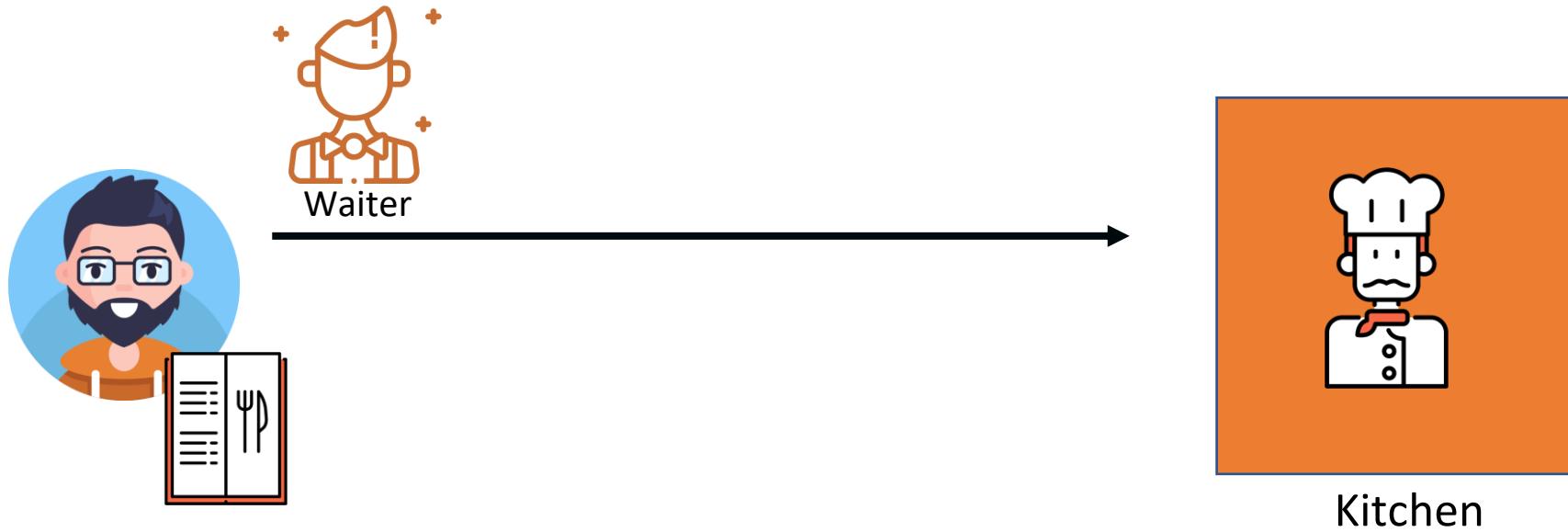
# Wiki Definition

An **application programming interface (API)** is a connection between [computers](#) or between [computer programs](#). It is a type of software interface, offering a service to other pieces of [software](#).<sup>[1]</sup> A document or standard that describes how to build or use such a connection or interface is called an *API specification*. A computer system that meets this standard is said to *implement* or *expose* an API. The term API may refer either to the specification or to the implementation.

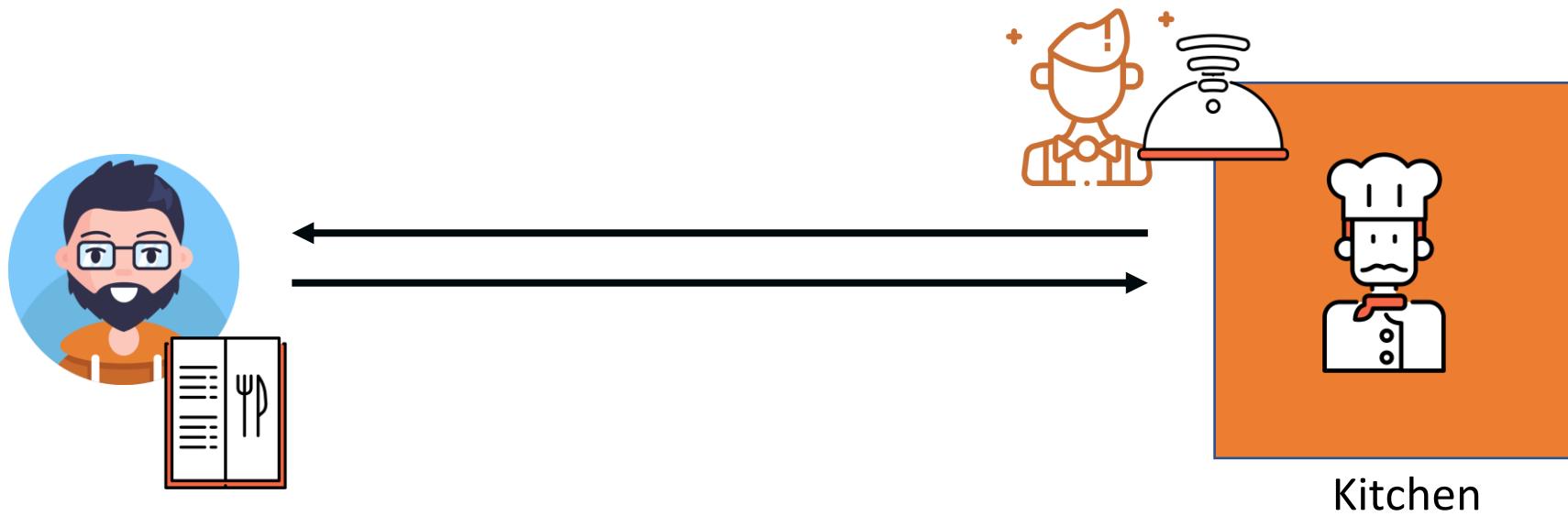
# Real World Example



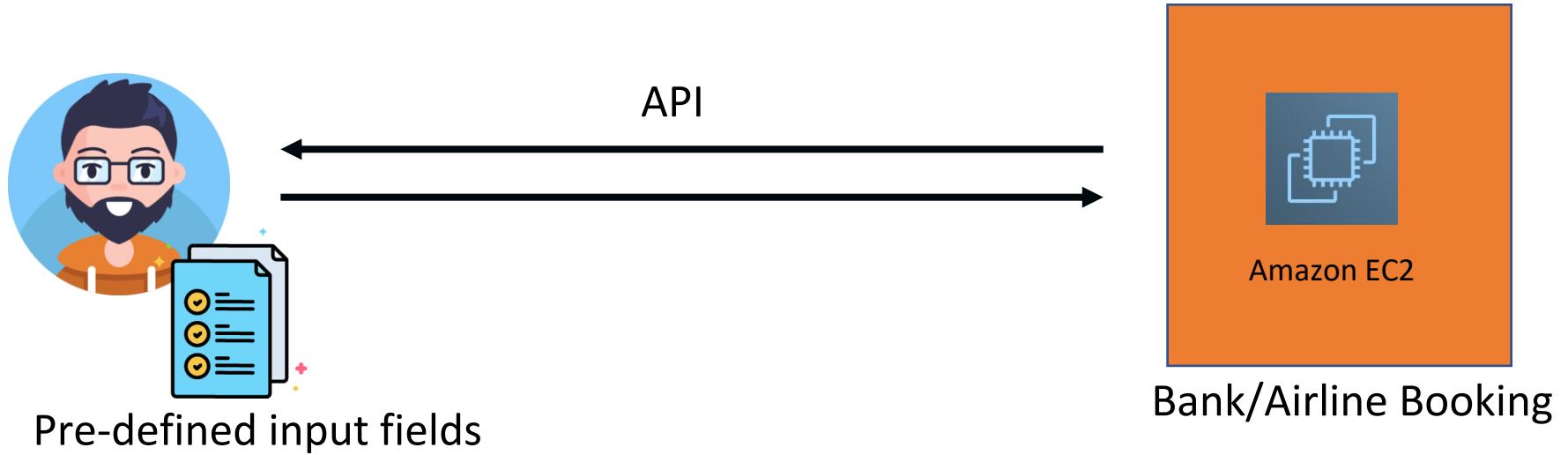
# Real World Example



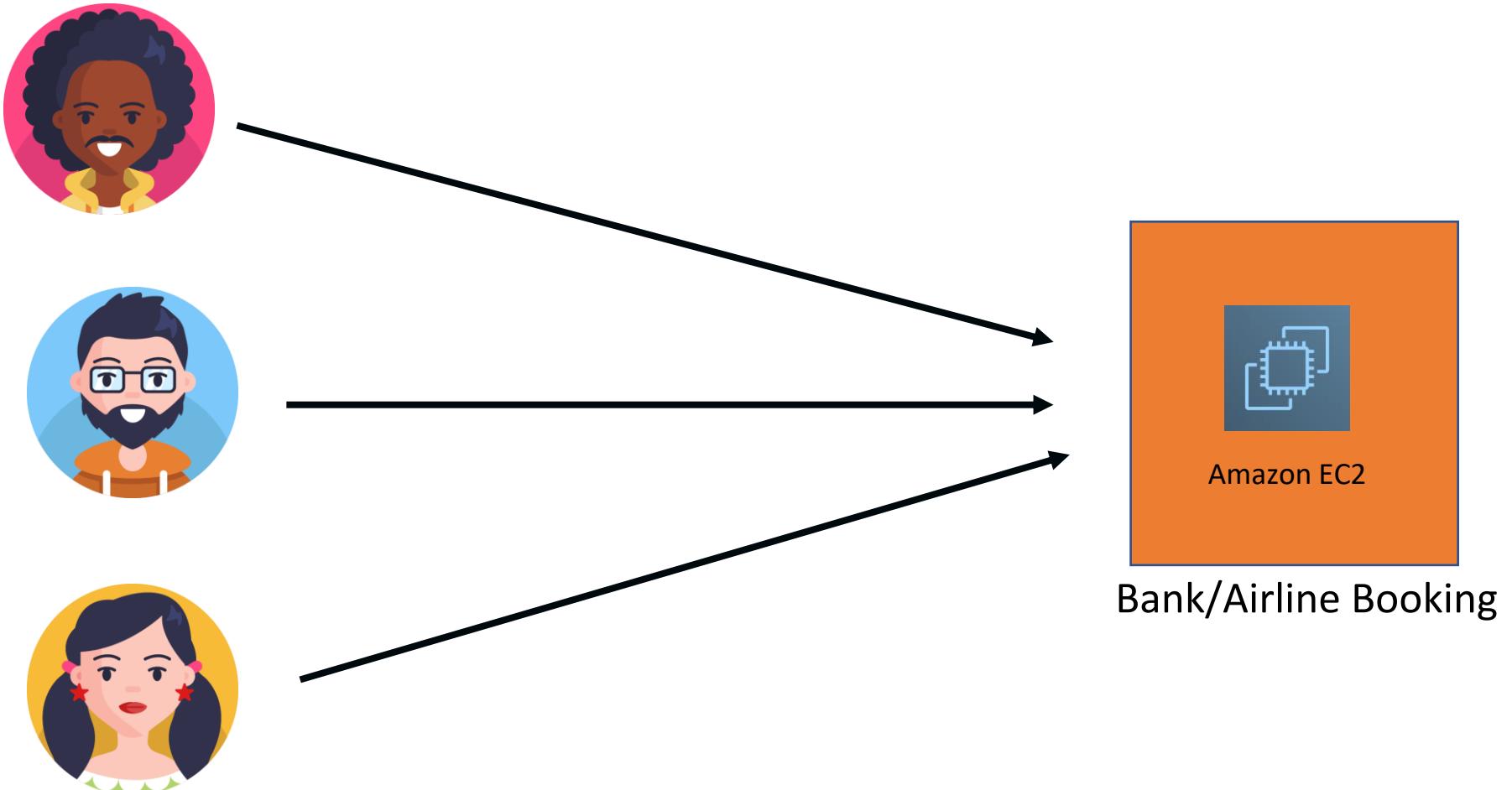
# Real World Example



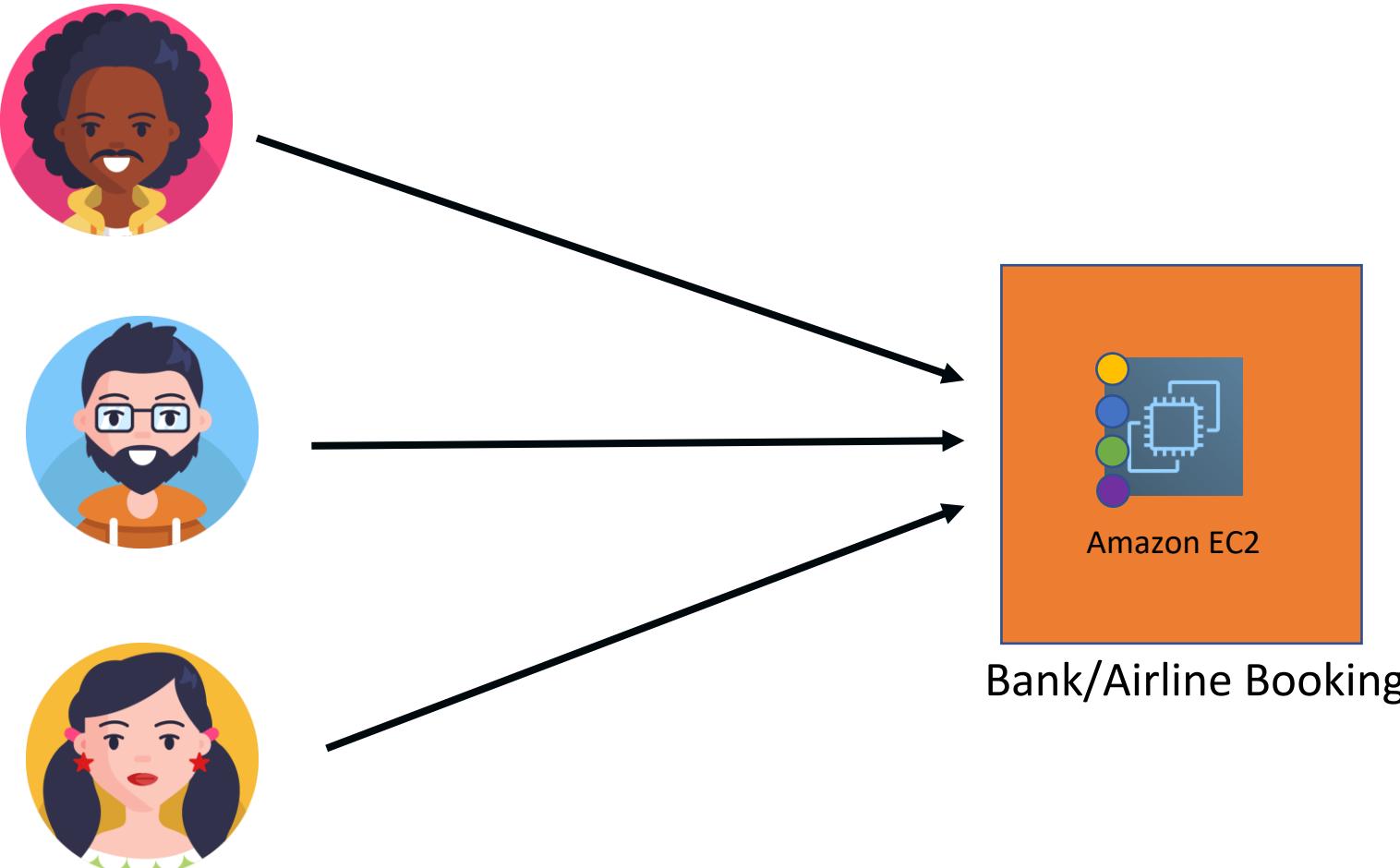
# Back to IT



# But Why?

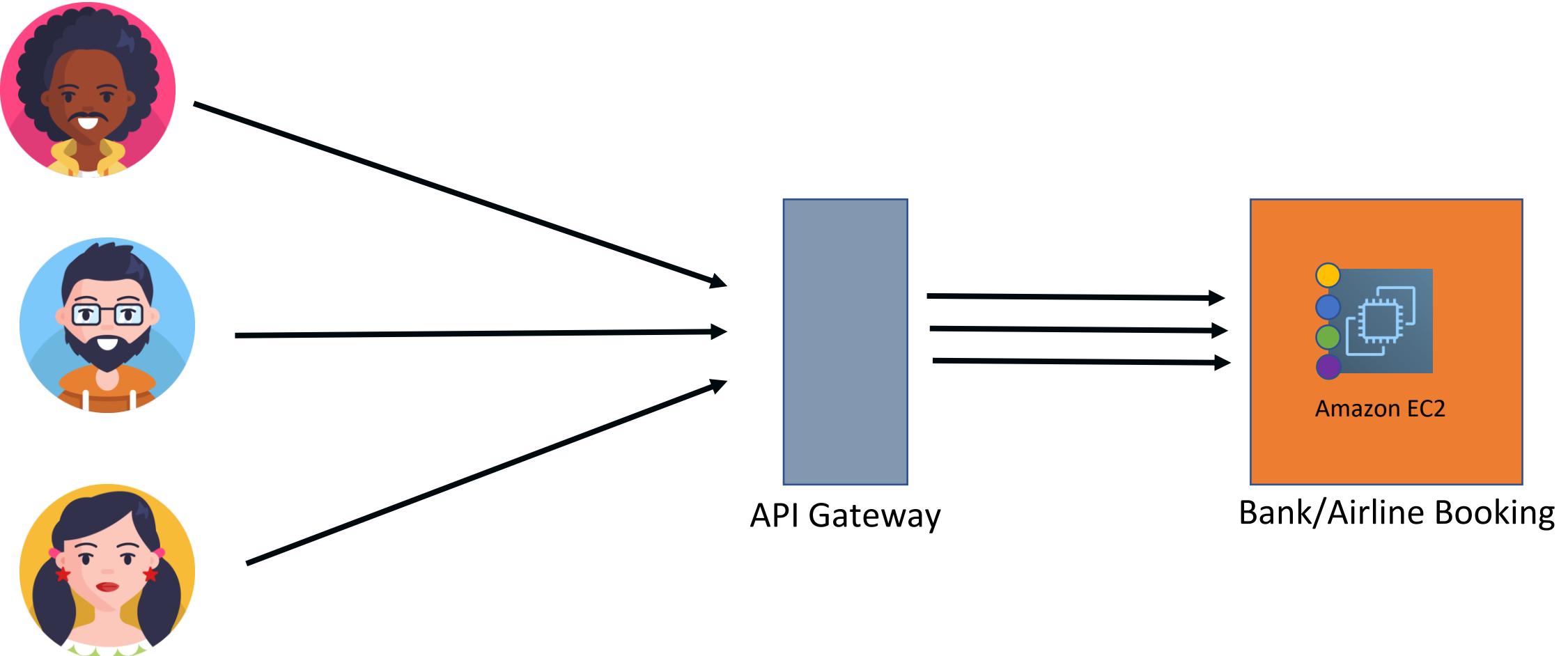


# But Why?

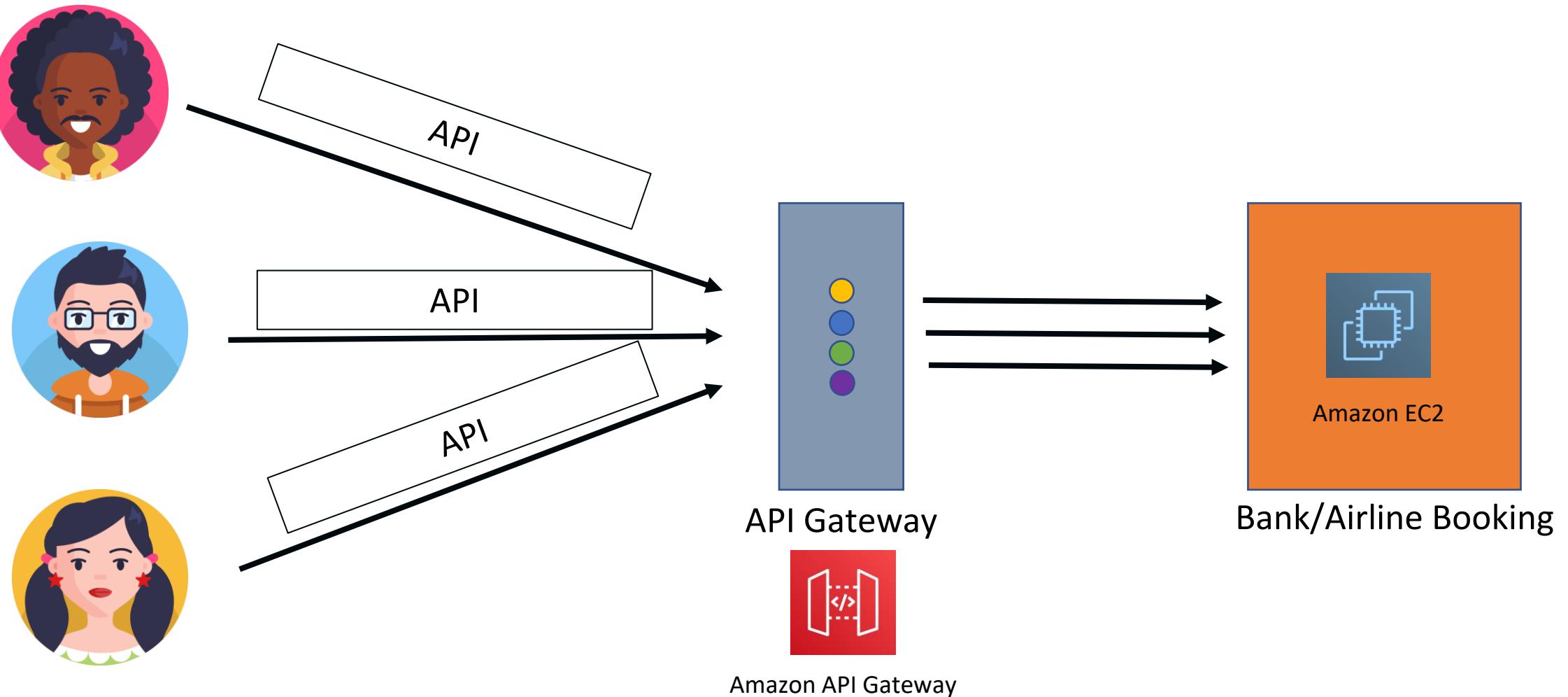


- Traffic management
- Load balancing
- Specific input/output needs
- AuthN/Z

# But Why?



# AWS Implementation

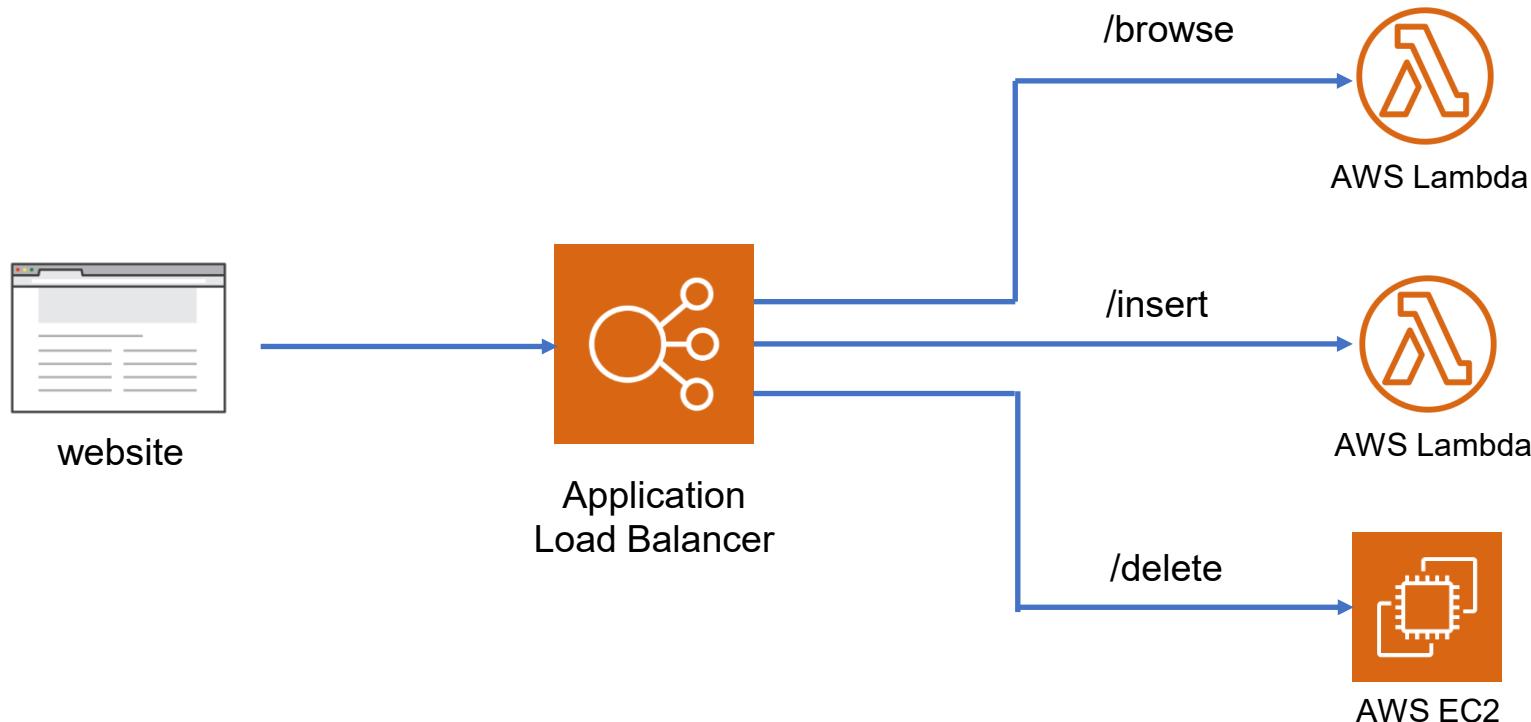


# ALB vs. API Gateway

# Application Load Balancer (ALB)

- Automatically distributes incoming traffic across backend targets
- Layer 7 load balancer
- Infrastructure managed by AWS, highly available, elastic

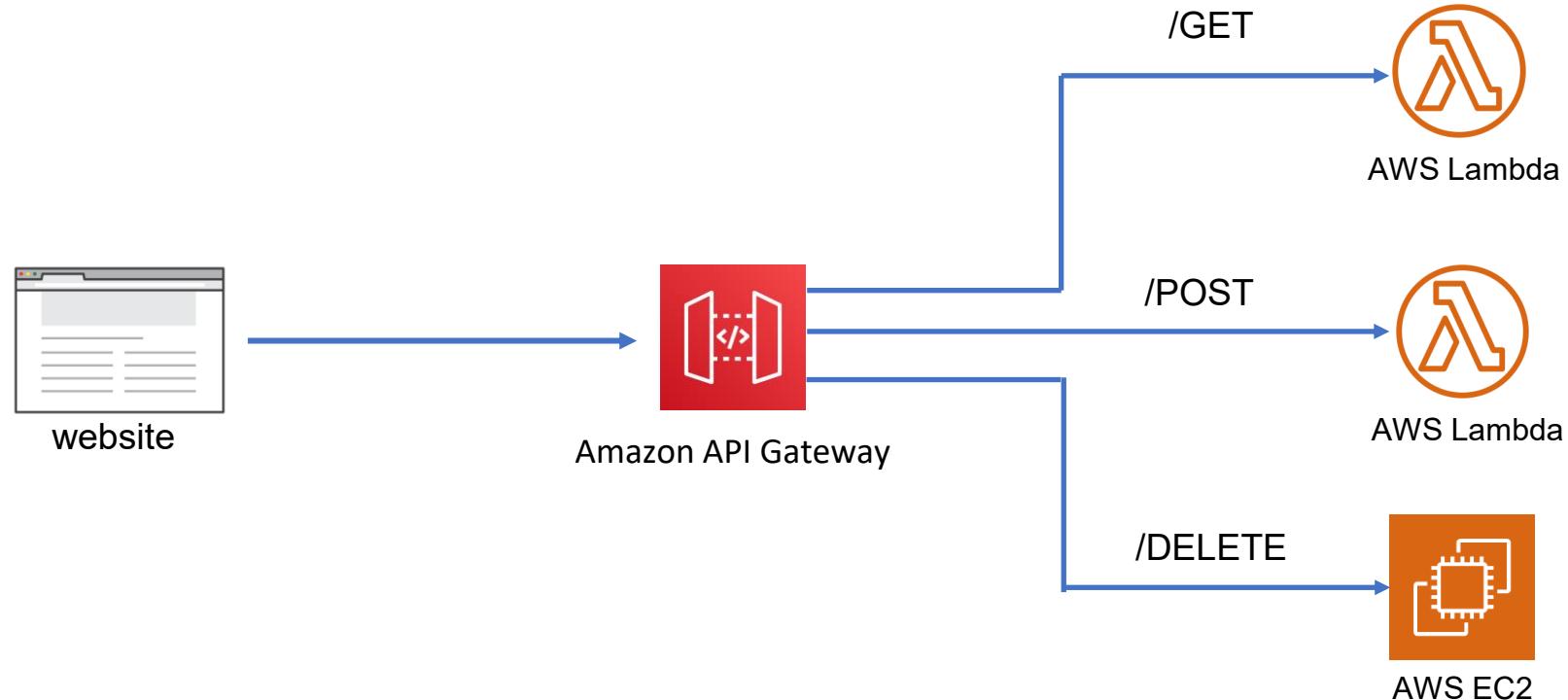
# ALB Integration



# API Gateway

- Fully managed and serverless API service from AWS
- Automatically scales up and down
- Infrastructure managed by AWS, highly available, elastic

# API Gateway Integration



# API Gateway



Can implement rate limiting, bursting for APIs

Integrate with AWS WAF for protection

Not possible to get a static IP address for endpoint

Accepts HTTPS traffic

Able to do request validation, request/response mapping

Able to handle spiky traffic (default rate – 10k rps, 5k burst rate)

Able to integrate with Lambda from different region, even different AWS account

# ALB



No rate limiting, bursting capability

Integrate with AWS WAF for protection

Possible to get a static IP address for load balancer endpoint

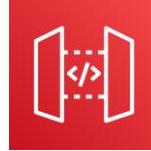
Accepts HTTP, HTTPS traffic

Not able to do request validation, request/response mapping

Delay during spiky traffic, pre-allocate LCUs to avoid delay (charged extra)

ALB is a regional service

# API Gateway



Able to export/import APIs cross API platforms using swagger, Open API Spec 3.0

Have extensive AuthN/Z integration – API Key, IAM, Cognito User Pool, Cognito Identity Pool, external IdP

Able to cache responses

Timeout limit 30 seconds

Integrates with almost all AWS services

No health check available

Pay per use

# ALB



No direct method to import/export rules for cross platforms

Integration with any OIDC compliant IdP (Cognito, LDAP etc.)

Not able to cache responses

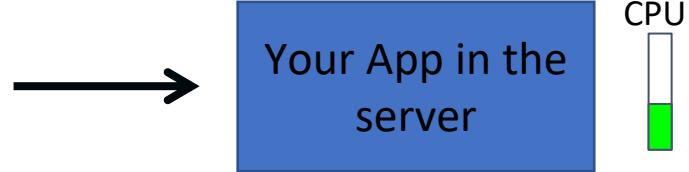
Timeout limit 4000 seconds

Use EC2, Lambda, IP addresses as backend  
Health check available

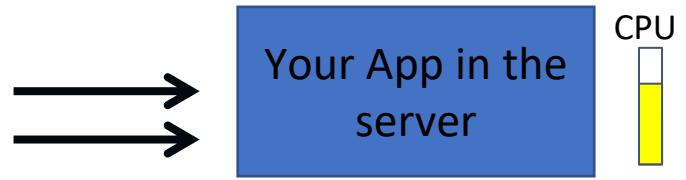
Pay for idle

# Scaling – Vertical vs Horizontal

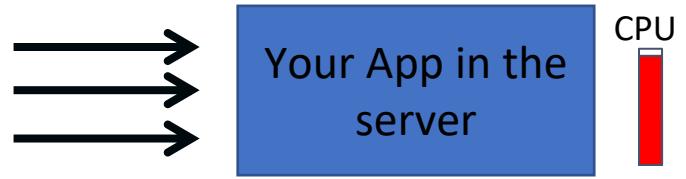
# Regular Application



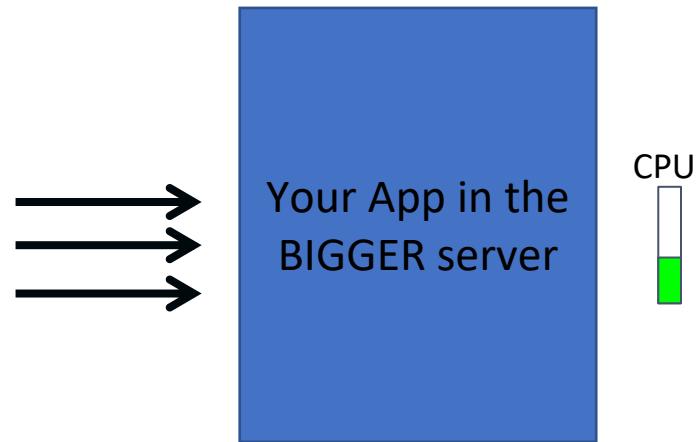
# Regular Application



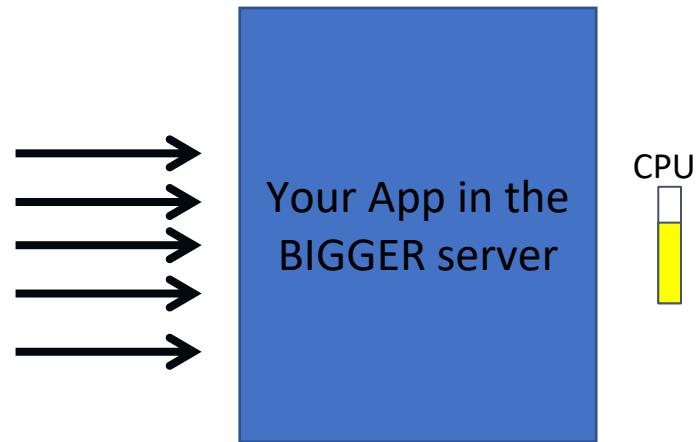
# Regular Application



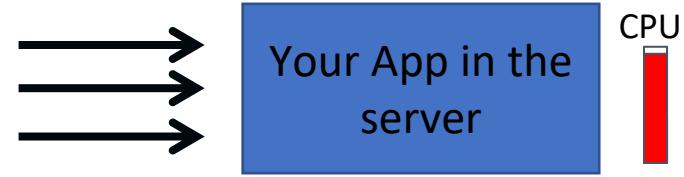
# Vertical Scaling



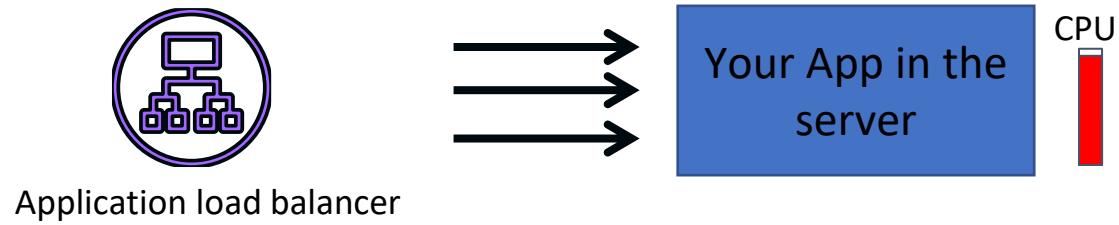
# Vertical Scaling



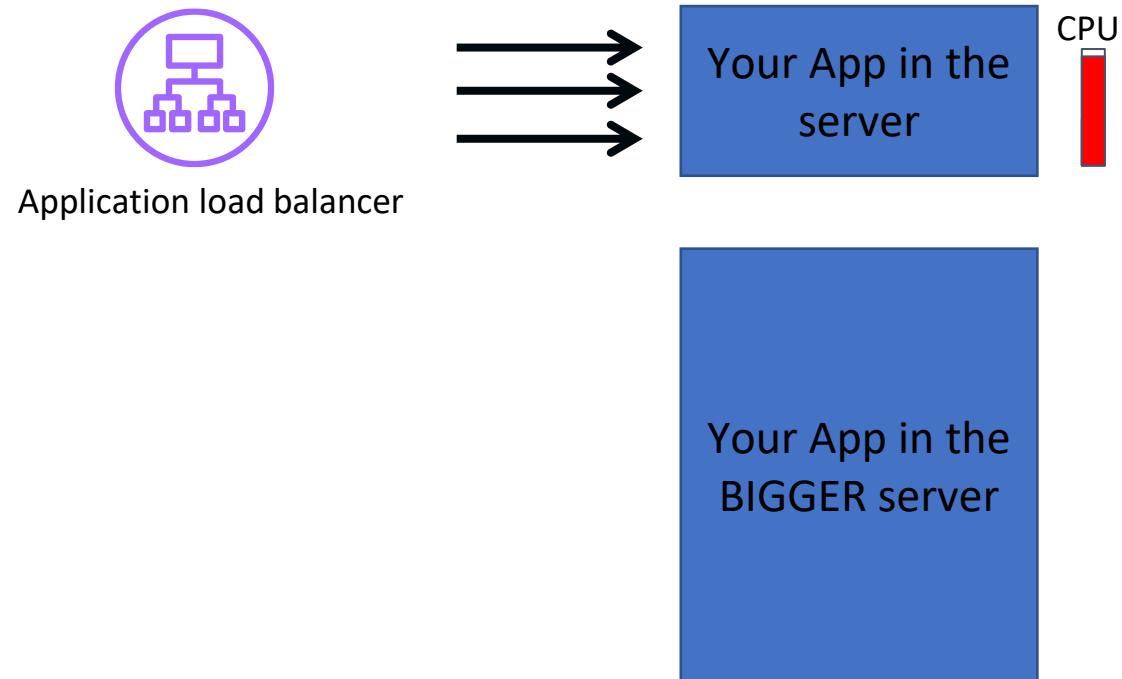
# Vertical Scaling Deep Dive



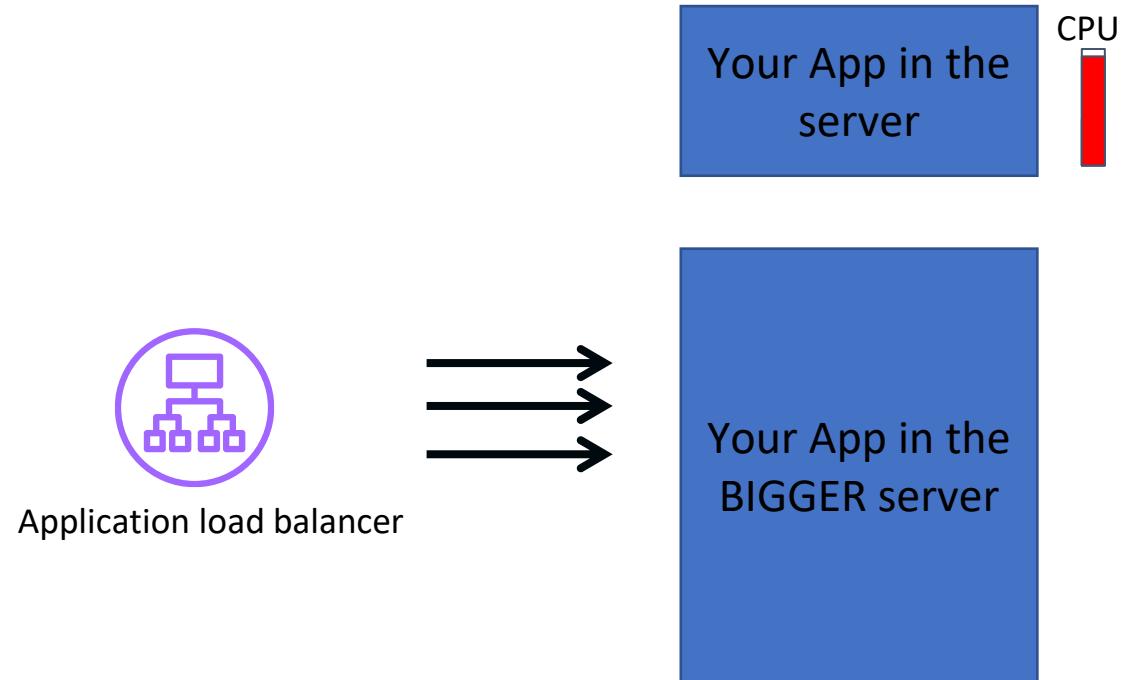
# Vertical Scaling Deep Dive



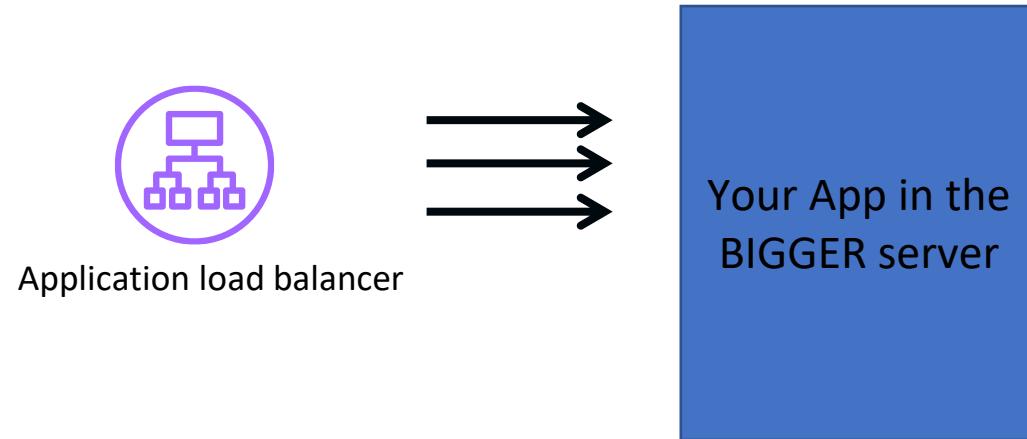
# Vertical Scaling Deep Dive



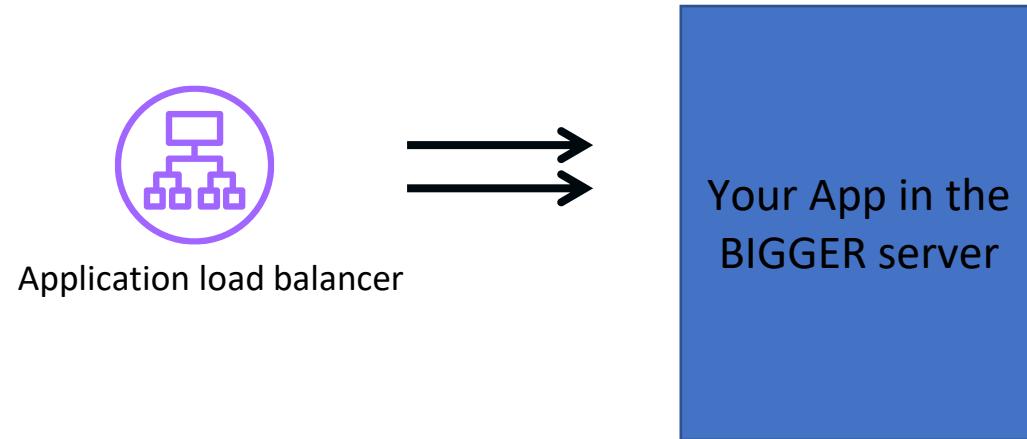
# Vertical Scaling Deep Dive



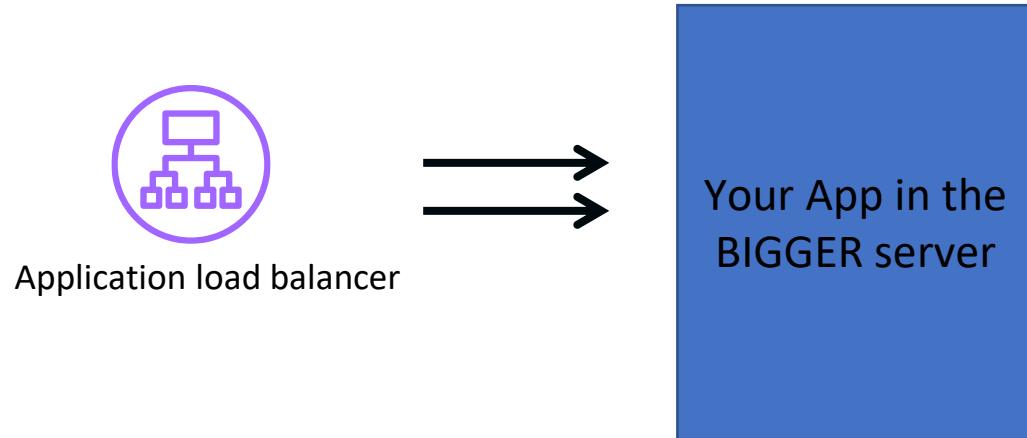
# Vertical Scaling Deep Dive



# Vertical Scaling Deep Dive

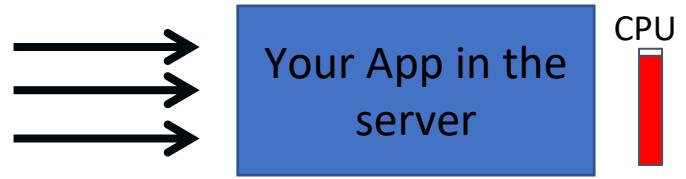


# Vertical Scaling Challenges

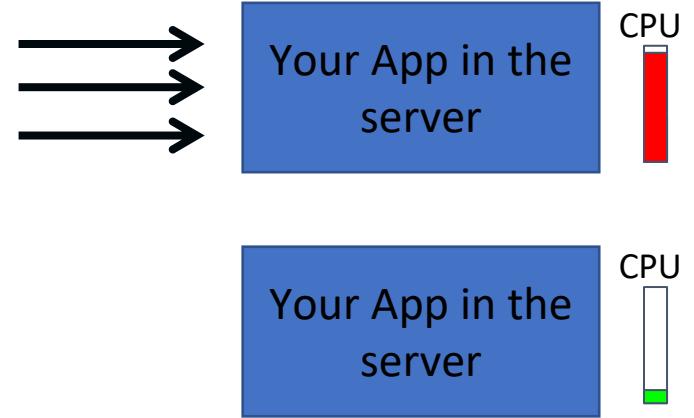


- Scaling up/down takes longer
- Chance of missing transactions during scaling cutover
- Limited scaling
- Expensive

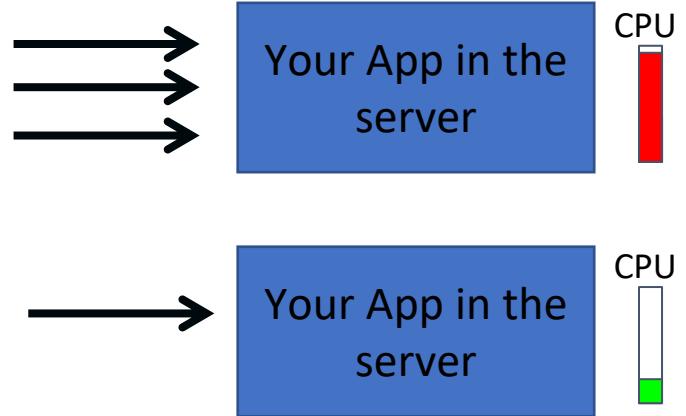
# Regular Application



# Horizontal Scaling



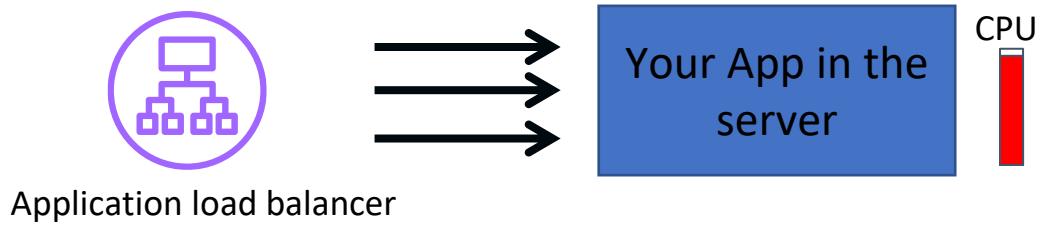
# Horizontal Scaling



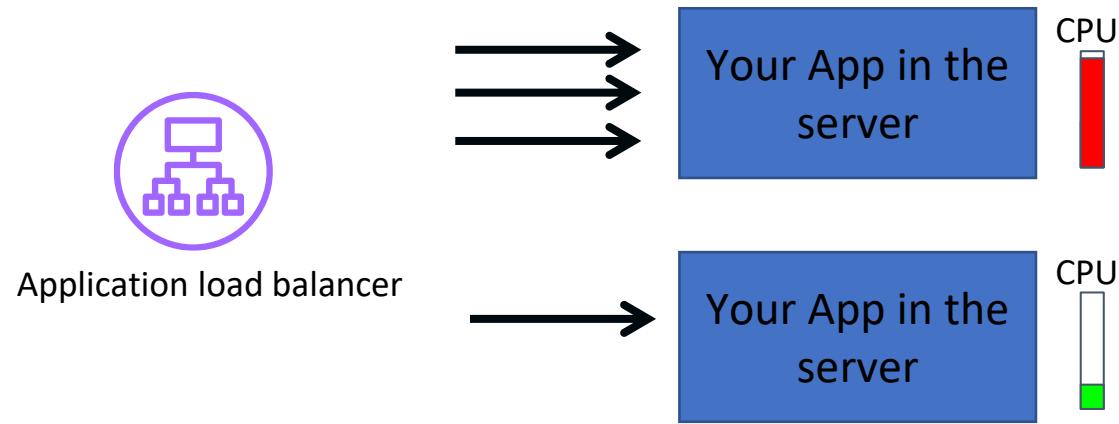
# Horizontal Scaling



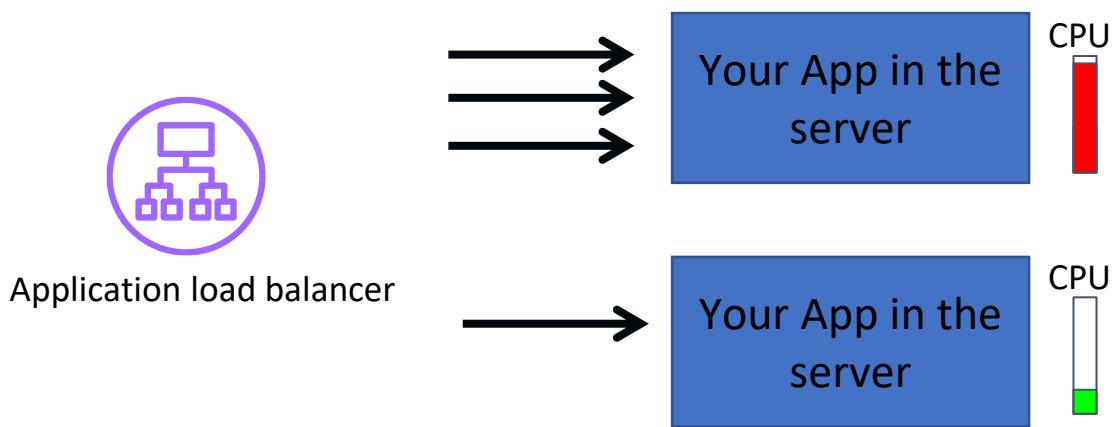
# Horizontal Scaling Deep Dive



# Horizontal Scaling Deep Dive



# Horizontal Scaling



- Scaling up/down faster
- Massively scalable
- Cost effective
- Legacy code needs to be refactored for horizontal scaling

# VM, Container, Serverless Scaling on AWS

# Vertical vs Horizontal Scaling on AWS

- EC2 vertical and horizontal scaling
- Container scaling
- Lambda scaling
- Fargate scaling

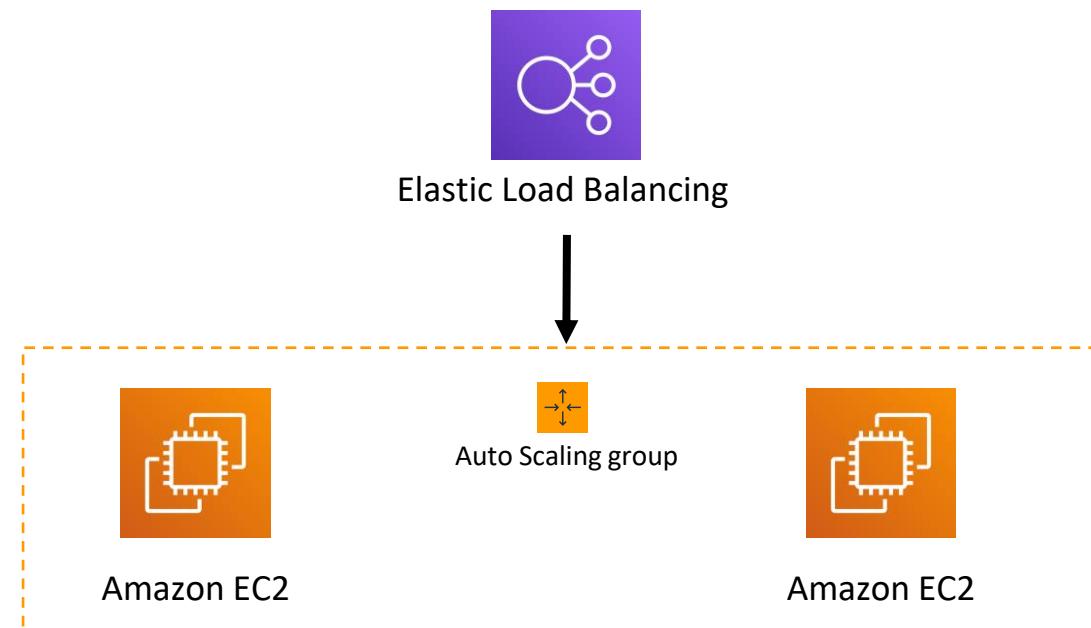
# Let's Whiteboard!

# Real World Interview Tips on Scaling

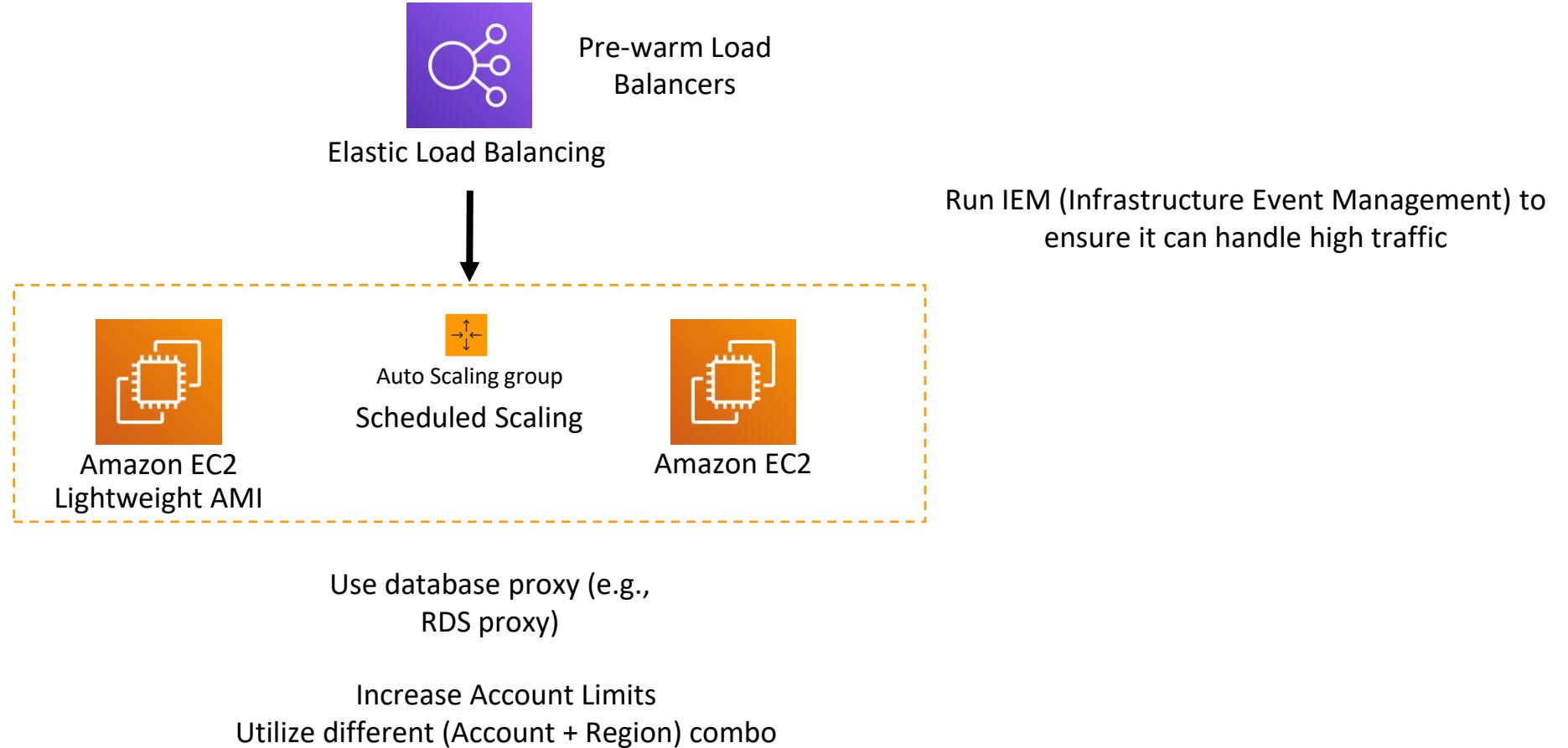
# How can you make your application scalable for a big traffic day?

Average Answer

Put the VMs in auto scaling group and use load balancer

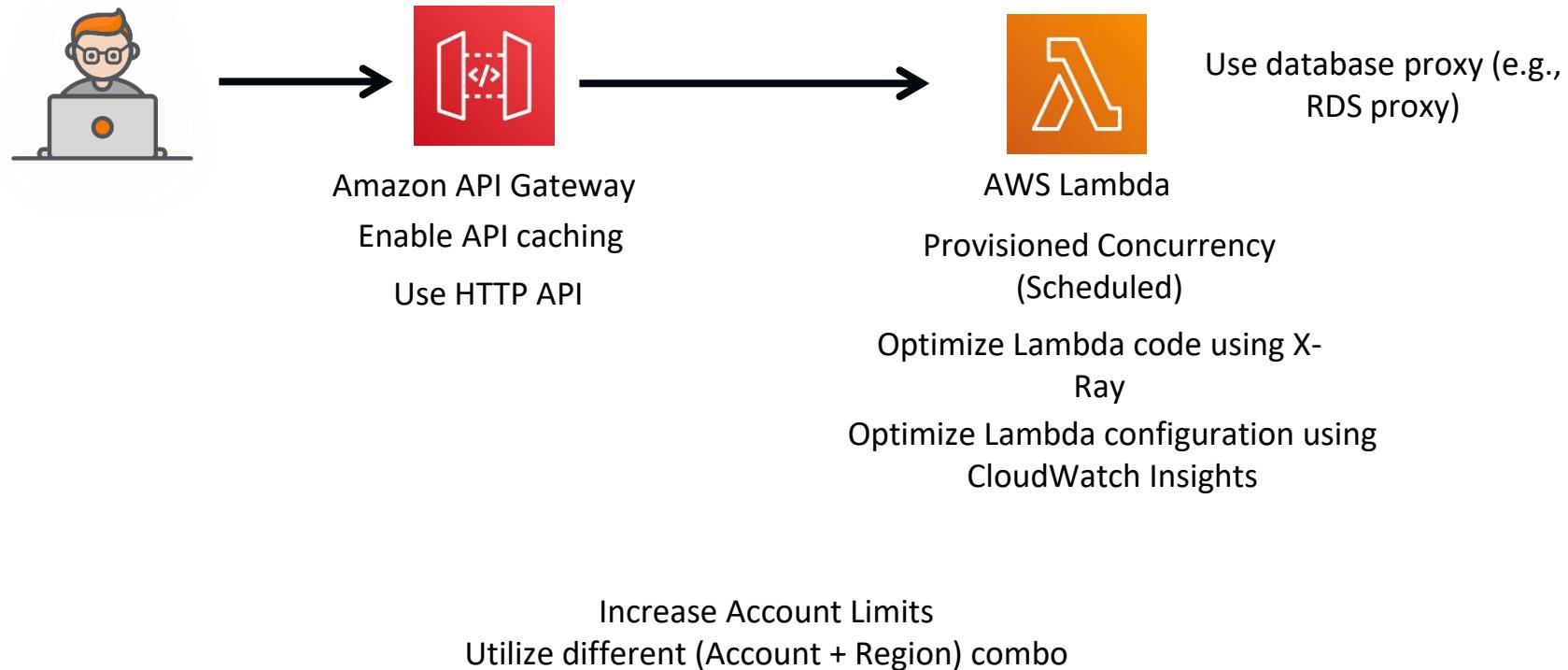


# GOOD Answer

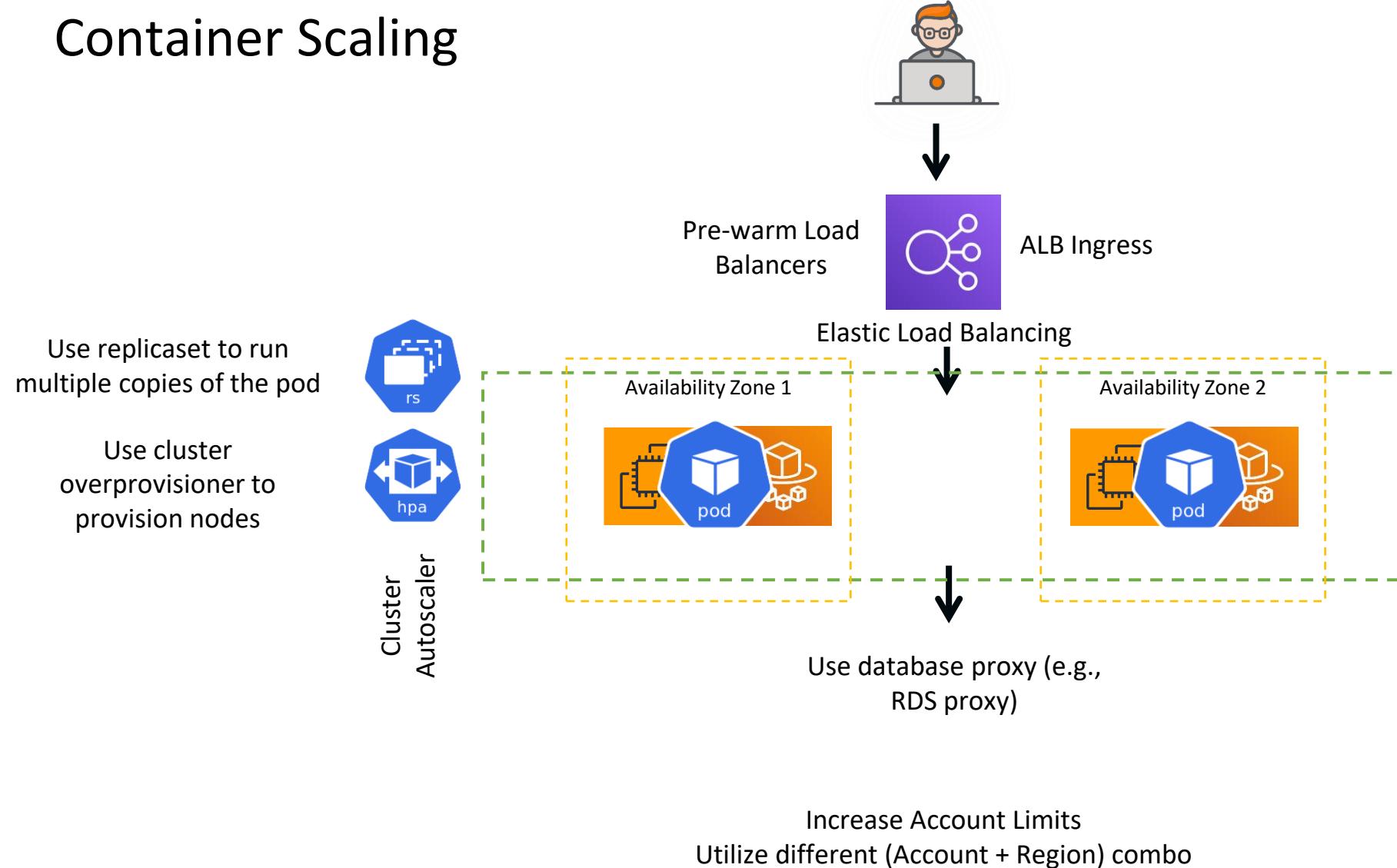


*Talk about breaking the app into microservices  
Going into Kubernetes or Serverless doesn't eliminate these challenges*

# Serverless Scaling

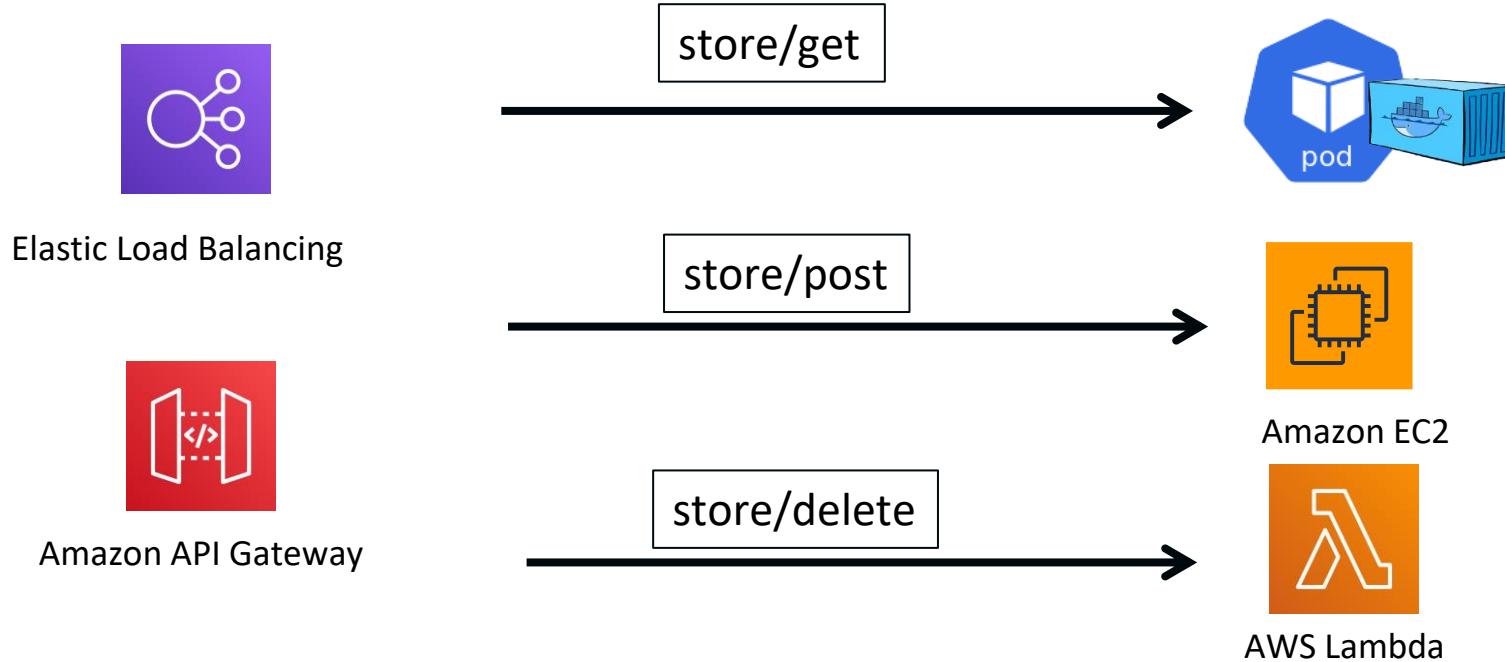


# Container Scaling

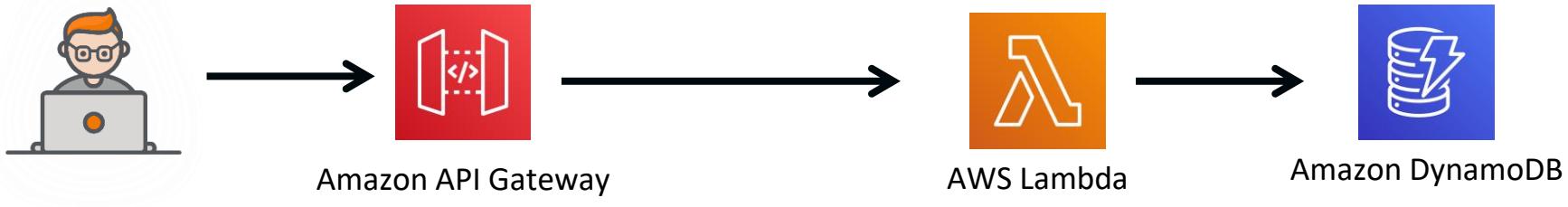


# Synchronous vs. Event Driven/Async Architectures

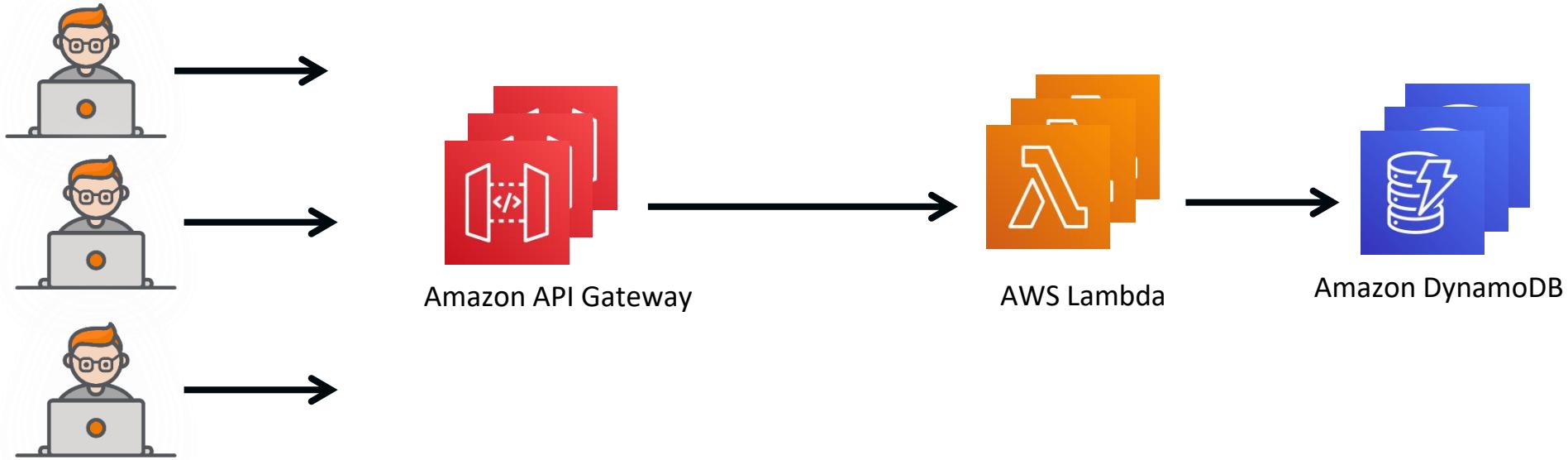
# Microservices



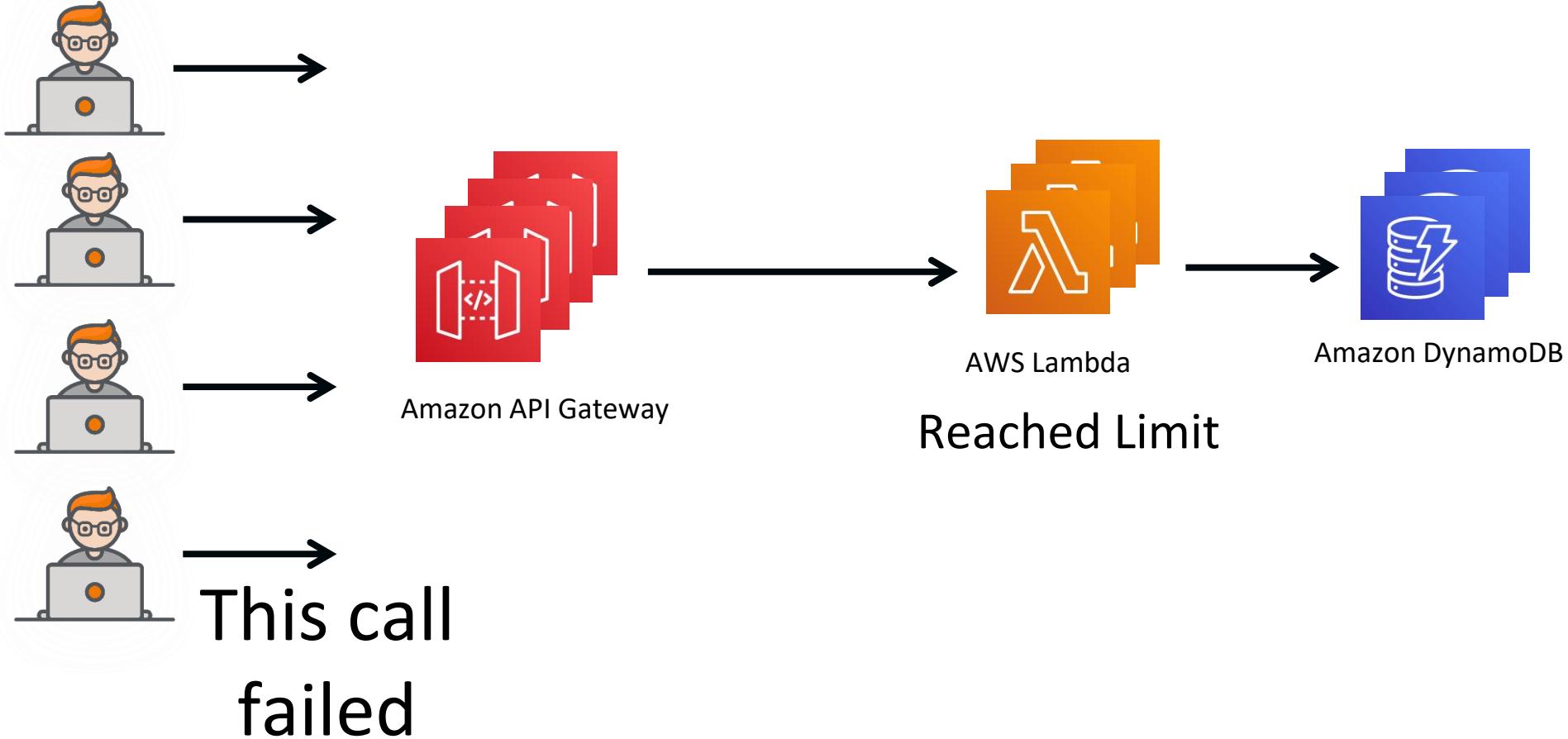
# Synchronous Architecture



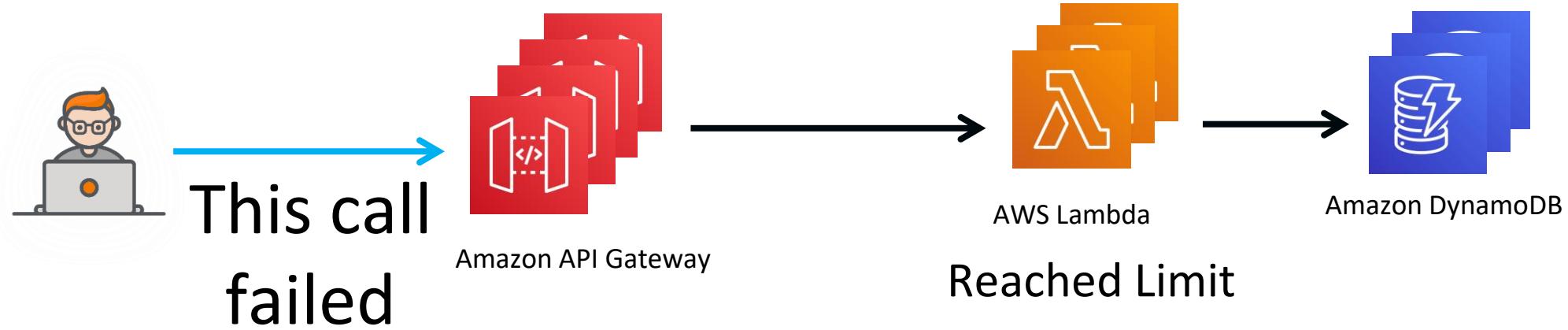
# Synchronous Architecture



# Synchronous Architecture



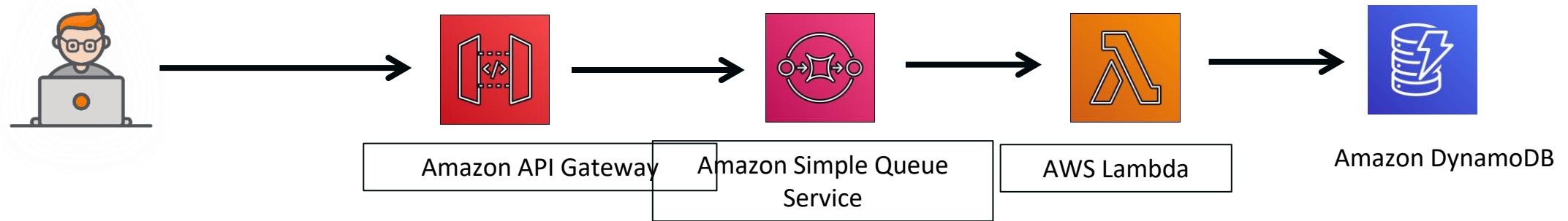
# Synchronous Architecture



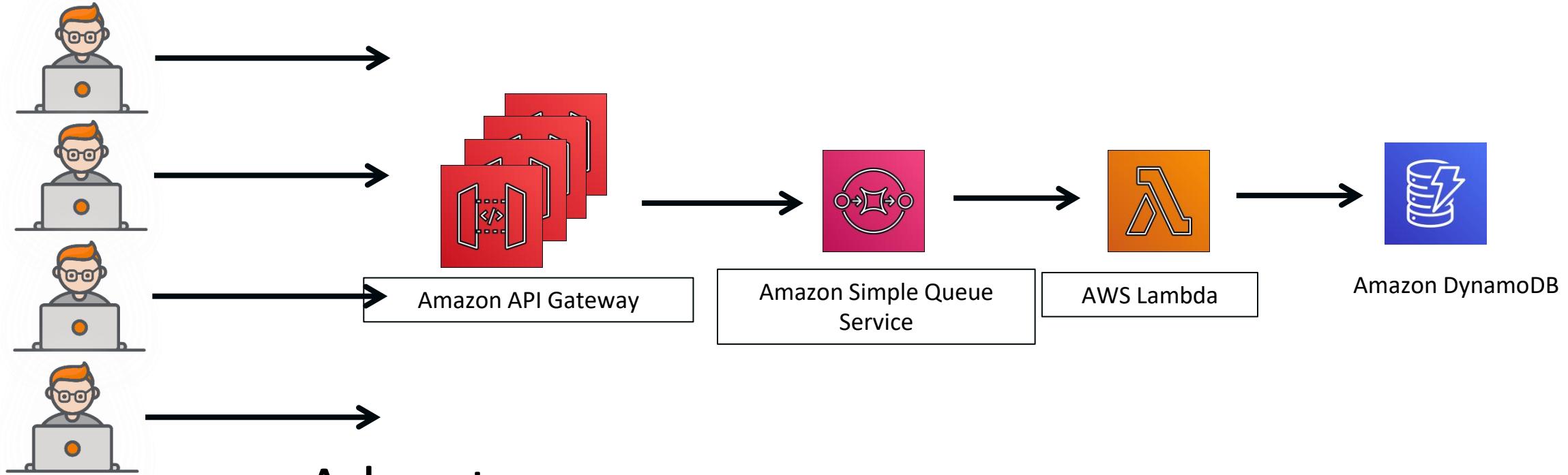
## Challenges

- All components of Synchronous architectures **MUST** scale together
- Consumer needs to resend transaction for re-processing
- Expensive

# Event-Driven/Async Architecture



# Event-Driven/Async Architecture



## Advantages

- Each component can scale independently
- Retry built in
- Cost effective than synchronous architecture

# Stronger Together!

- Use synchronous and event-driven architectures where applicable
- Example ordering system
  - Order inserts can be done event-driven
  - Order status retrieval synchronously

# PubSub Vs Queues

# Streaming vs Messaging

# SQL vs NoSQL

# What We Going To Learn...

- SQL Vs NoSQL
- AWS Database Options
- Amazon Aurora Vs DynamoDB
- Conclusion

# SQL Vs NoSQL Database

SQL Database (RDBMS)	NoSQL Database
Tables have predefined schema	Schemaless
Holds structured data	Holds structured and unstructured data
Good fit for joins and complex queries	Generally, not good fit for complex multi table queries
Emphasizes on ACID properties (Atomicity, Consistency, Isolation and Durability)	Follows the Brewers CAP theorem (Consistency, Availability and Partition tolerance )
Generally, scales vertically	Generally, scales horizontally. AWS DynamoDB scales automatically!
Example - Oracle, DB2, Amazon Aurora, AWS RDS	Example - AWS DynamoDB, MongoDB, Cassandra
<b>Important:</b> With the advent of technology, segregation of use cases for SQL and NoSQL are <u>NOT</u> as black and white as it used to be, multiple factors need to be considered	

# Schema Vs Schemaless

## Schema

Artist	Song Title	Album Title	Price	Genre	Critic Rating

## Schemaless

```
{  
  "Artist": "No One You Know",  
  "SongTitle": "My Dog Spot",  
  "AlbumTitle": "Hey Now",  
  "Price": 1.98,  
  "Genre": "Country",  
  "CriticRating": 8.4  
}
```

```
{  
  "Artist": "No One You Know",  
  "SongTitle": "Somewhere Down The Road",  
  "AlbumTitle": "Somewhat Famous",  
  "Genre": "Country",  
  "CriticRating": 8.4,  
  "Year": 1984  
}
```

```
{  
  "Artist": "The Acme Band",  
  "SongTitle": "Still in Love",  
  "AlbumTitle": "The Buck Starts Here",  
  "Price": 2.47,  
  "Genre": "Rock",  
  "PromotionInfo": {  
    "RadioStationsPlaying": [  
      "KHCR",  
      "KQBX",  
      "WTNR",  
      "WJJH"  
    ],  
    "TourDates": {  
      "Seattle": "20150625",  
      "Cleveland": "20150630"  
    },  
    "Rotation": "Heavy"  
  }  
}
```

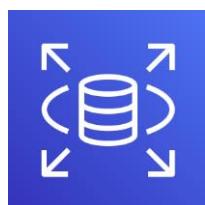
```
{  
  "Artist": "The Acme Band",  
  "SongTitle": "Look Out, World",  
  "AlbumTitle": "The Buck Starts Here",  
  "Price": 0.99,  
  "Genre": "Rock"  
}
```

# SQL Vs NoSQL in AWS

## SQL DATABASES



Amazon Aurora



Amazon RDS

## NOSQL DATABASES



Amazon DynamoDB



Amazon DocumentDB (with  
MongoDB compatibility)



Amazon Managed  
Apache Cassandra  
Service

Note - You can always run your favorite non-AWS database on EC2

# Amazon DynamoDB





## Amazon Aurora

MySQL and PostgreSQL compatible relational database built for the cloud. 5 times faster than standard MySQL, 3 times faster than standard PostgreSQL at 1/10th the cost

Multi-Master Supported for MySQL

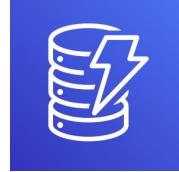
Cross region Active-Passive replication Supported for MySQL (Global Database)

Choosing Multi-AZ & Read Replicas provide High Availability

Vertical scaling. Serverless Aurora scales automatically, not as scalable as Dynamo.

Has integrated caches, can't be adjusted

Enable backups, snapshots for DR



## Amazon DynamoDB

Key-value and document database with single-digit millisecond performance AT ANY SCALE

Multi-Master

Cross region Active-Active replication Supported (Global Tables)

Inherently replicates across three AZs - HA and Durable

Inherently Scalable, can handle more than 10 trillion requests/day & peaks of more than 20 million requests/second

Provides adjustable in-memory caching via DAX

Inherently durable, Point In Time Backups can be enabled

# TAKING IT ALL IN - RIGHT TOOL FOR RIGHT JOB!

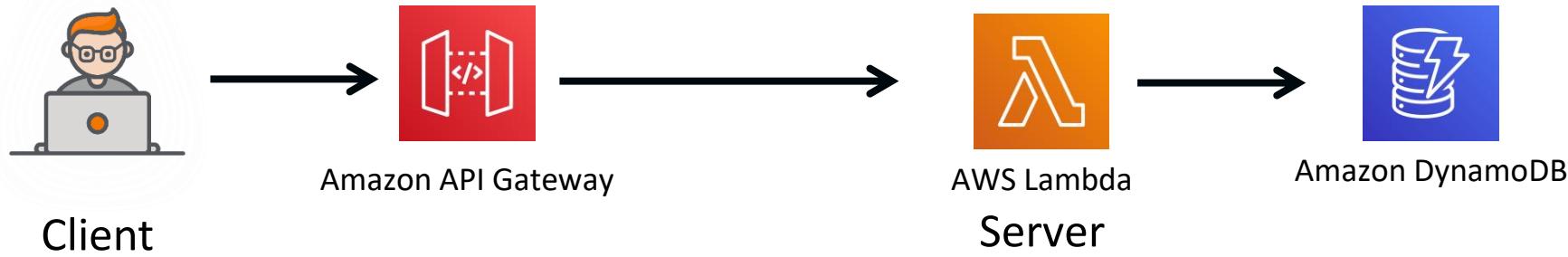


Do not use a cannon to kill a mosquito.

~ Confucius

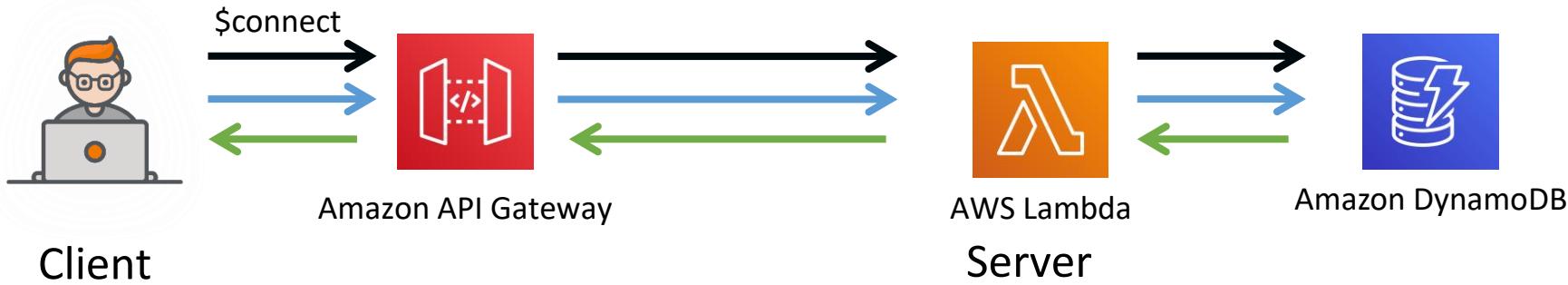
# Websockets

# Request-Response



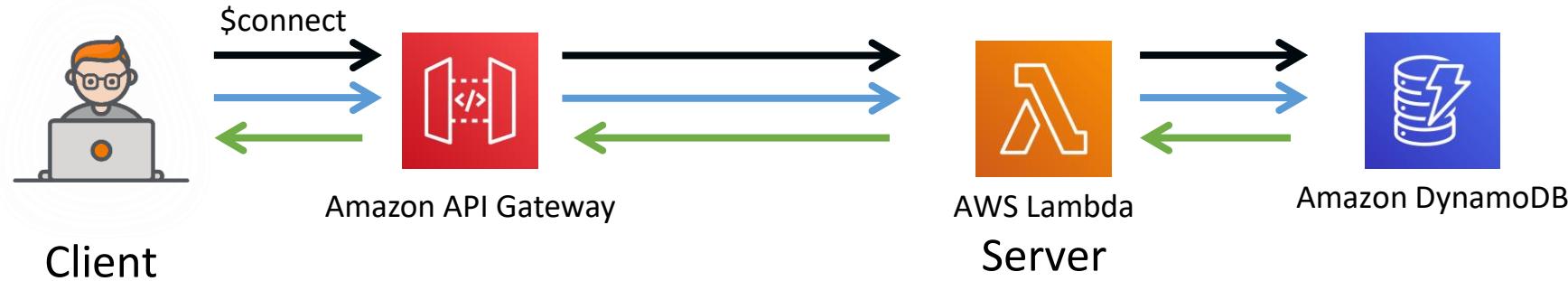
Only client can invoke server  
Server can NOT initiate connection to client

# Websocket



Connection stays open  
Server can send messages to client  
Can be achieved using Load Balancer and API Gateway

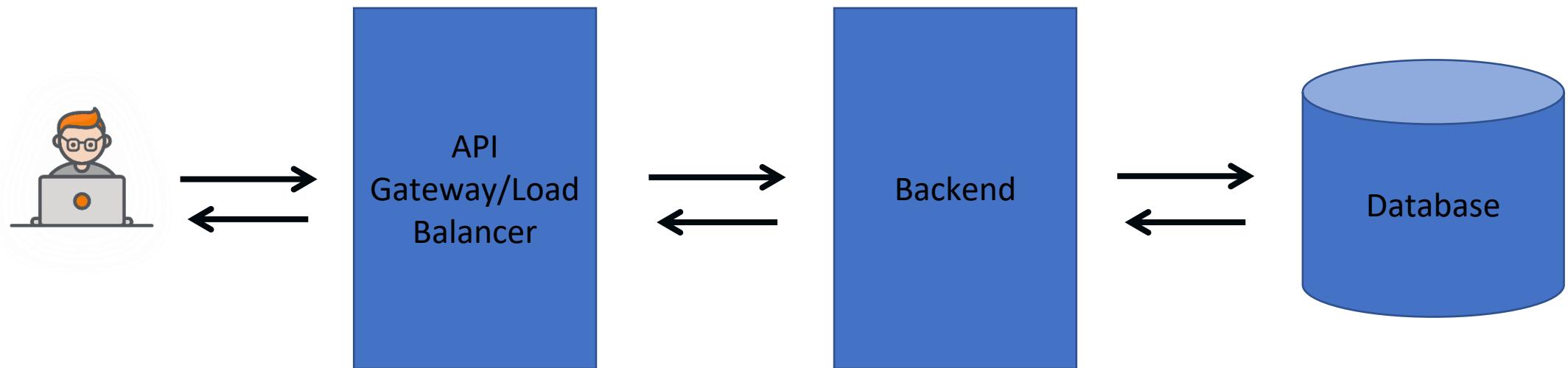
# Websocket Use Cases



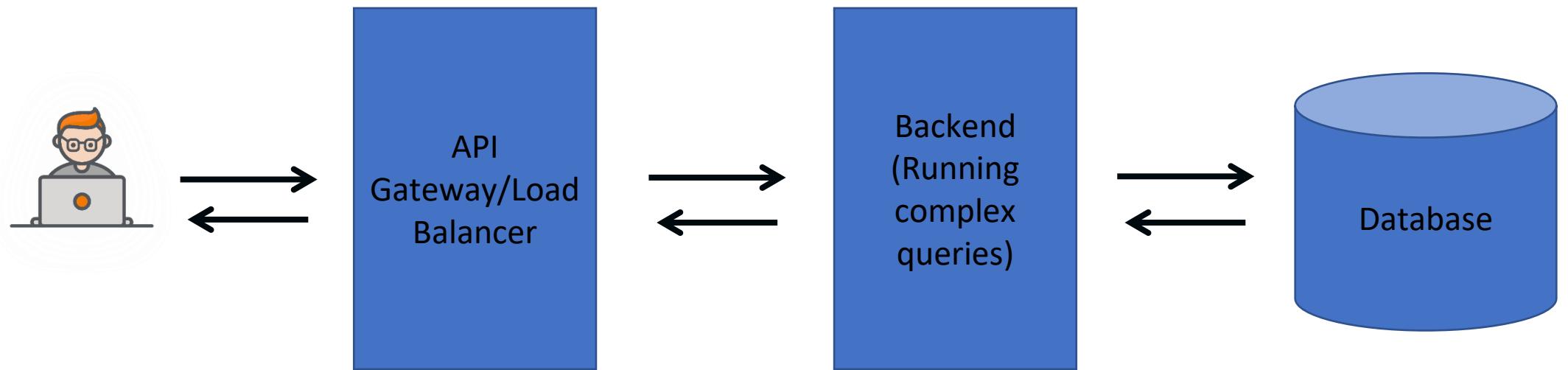
Chat applications – WhatsApp, Chatbots, Telegram

# Caching

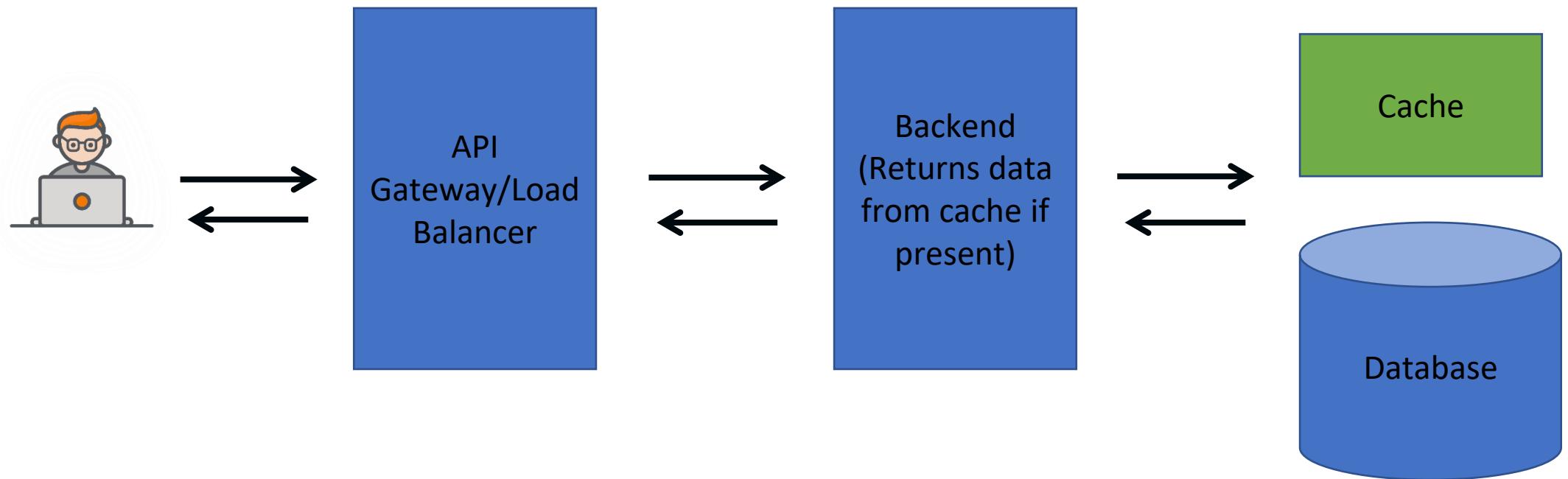
# Caching – What and Why



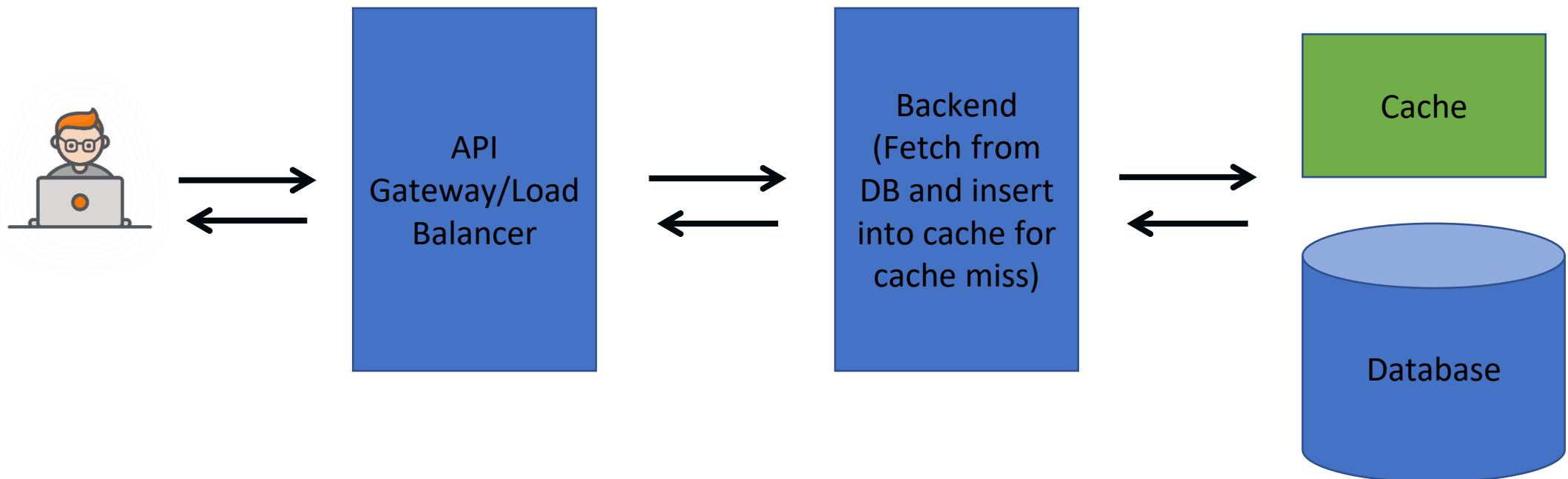
# Caching – What and Why



# Faster & Cost-efficient



# How Does Cache get Populated?

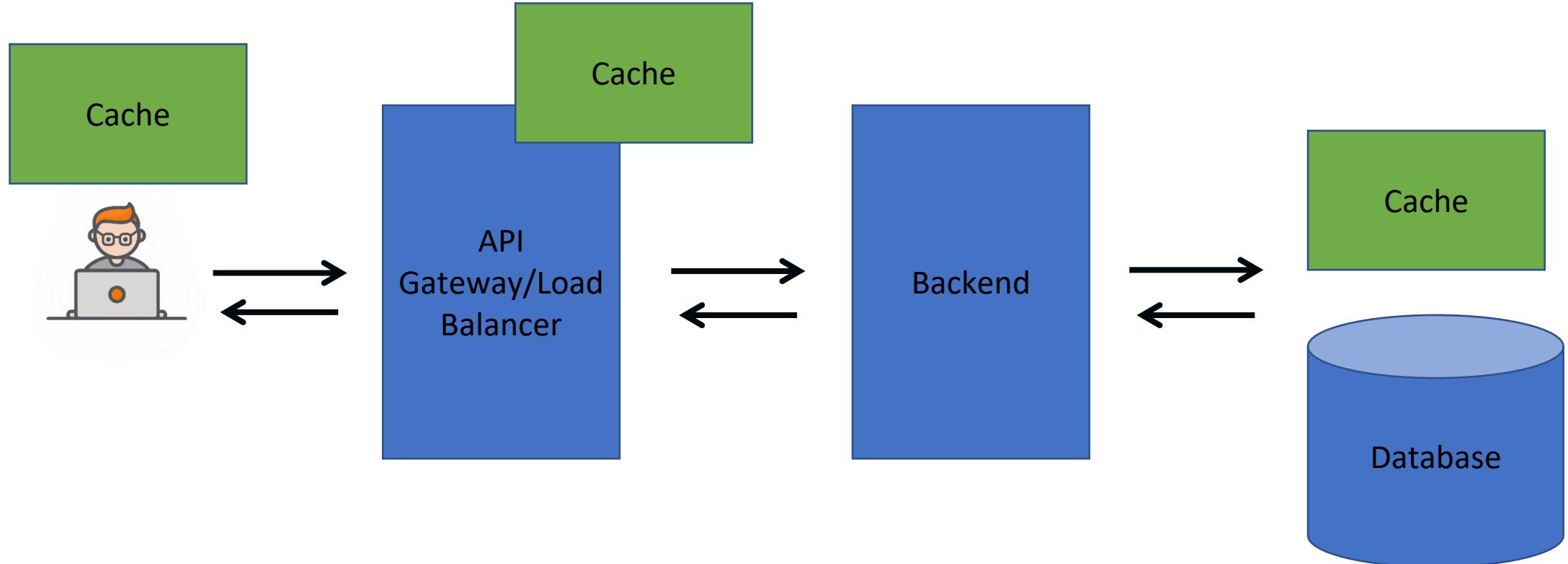


- Cache insert strategies later

# How Does Cache get Deleted?

- Cache entries deleted after a specified time
- This is Time To Live (TTL)
- Cache entries can be updated with backend code
  - Think of cache like another database or file system

# Cache is NOT restricted to Backend



# Which Caching Service to Use When?

- Use managed caching of the service
- If service doesn't provide caching then use cache database



Amazon ElastiCache

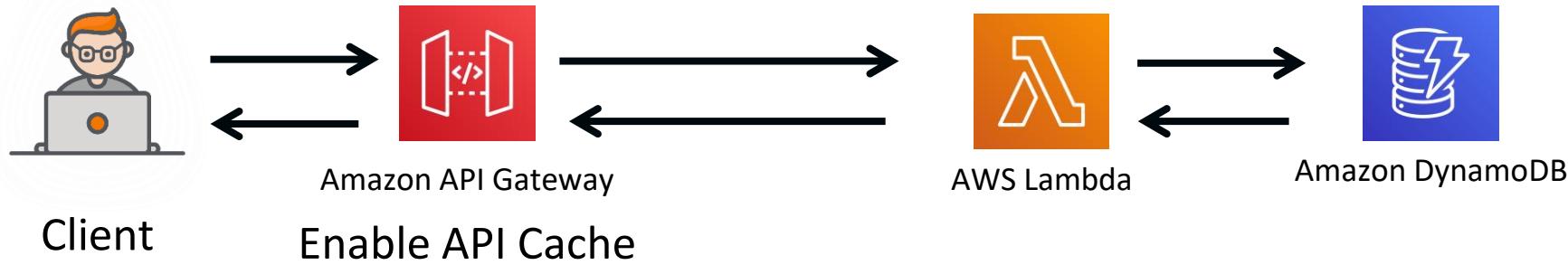


ElastiCache for  
Redis

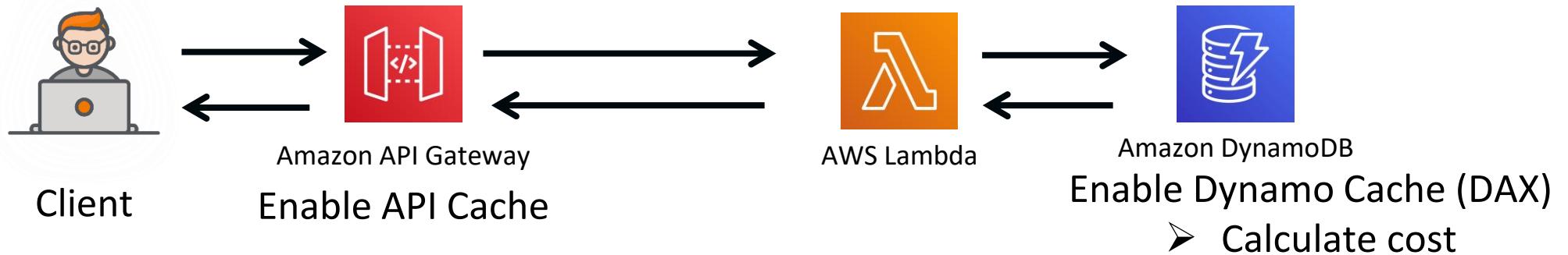


ElastiCache for  
Memcached

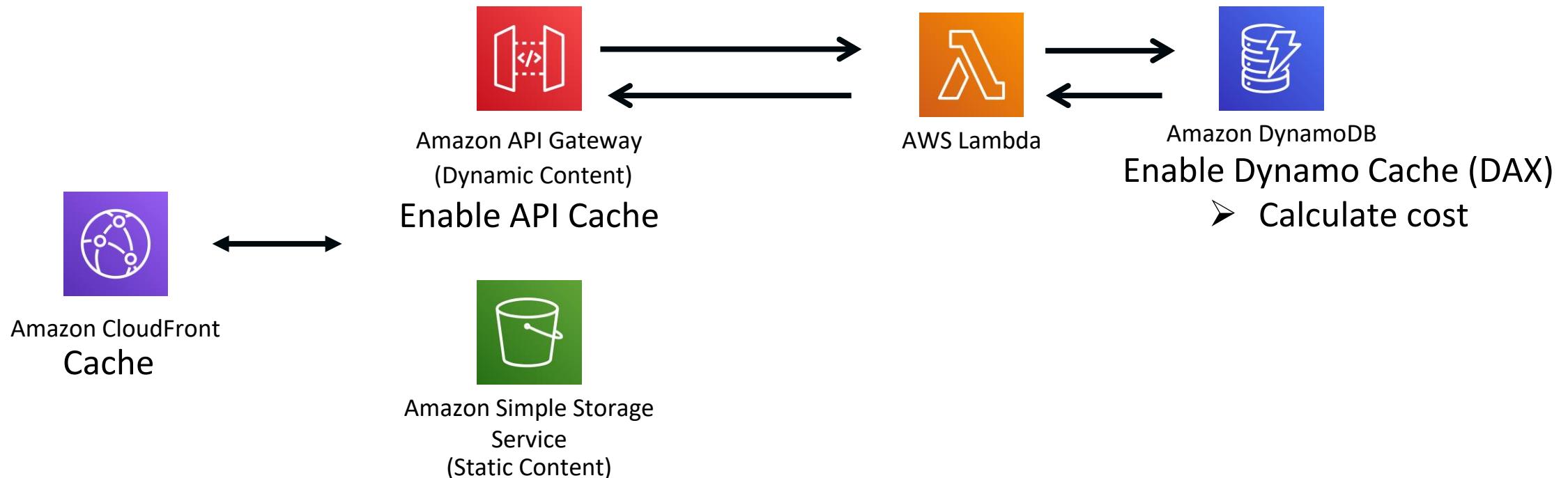
# Using Caching on AWS Services



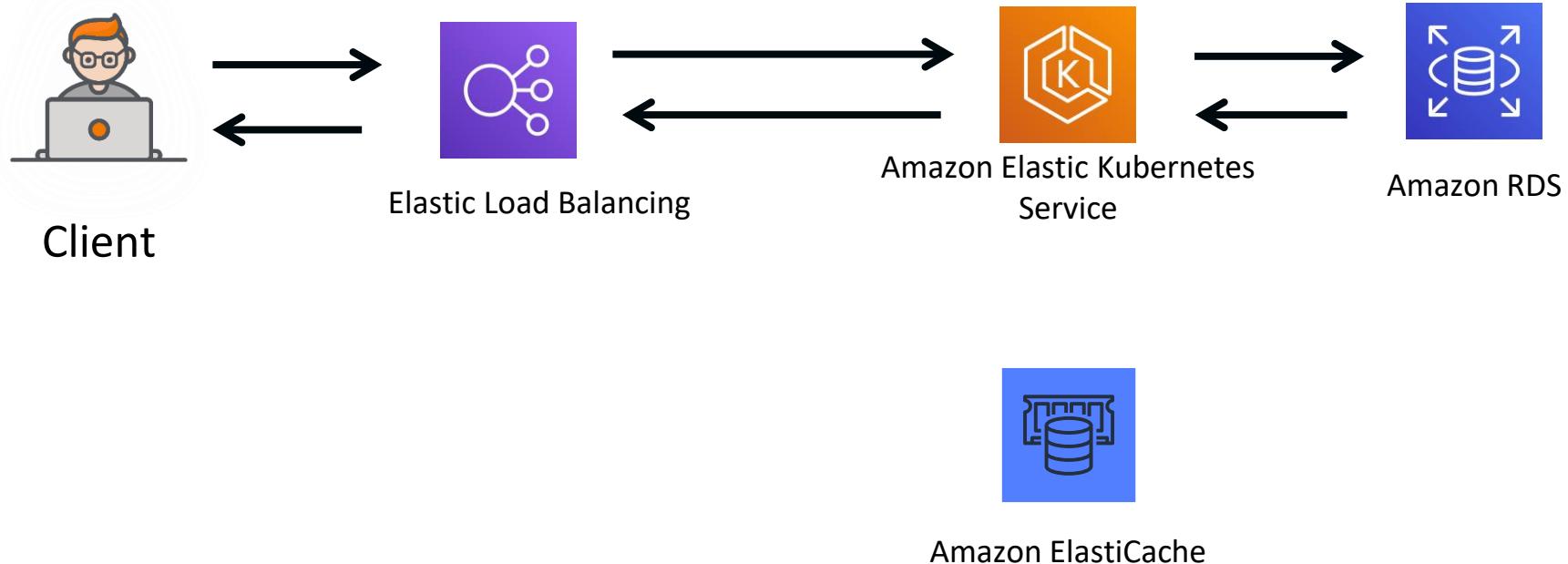
# Using Caching on AWS Services



# Using Caching on AWS Services



# Using Caching on AWS Services



# Redis Memcached & Caching Strategies

# Memcached vs. Redis

## Memcached

Simple data types

Large nodes with multiple cores or threads

Ability to scale out/in

Can cache object

## Redis

Complex data types

Sort or rank in-memory datasets

Replicate data

Automatic failover

Backup and restore

Publish and subscribe

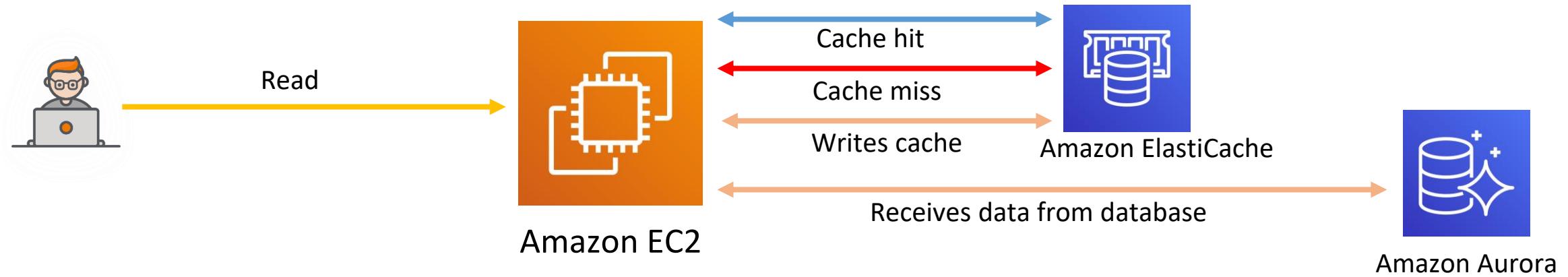
Support multiple databases

# ElastiCache Use Cases

- Cache frequently accessed data – user profile, preferences, item descriptions etc.
- Gaming leaderboards, real-time recommendations, messaging, and more



# Lazy Loading



# Write-Through



# High Availability

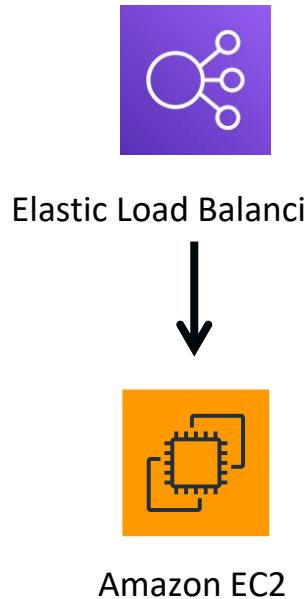
# High Availability

- System continues functioning even when some of its components fail
- System guarantees certain percentage of uptime

# Identifying Single Point of Failure

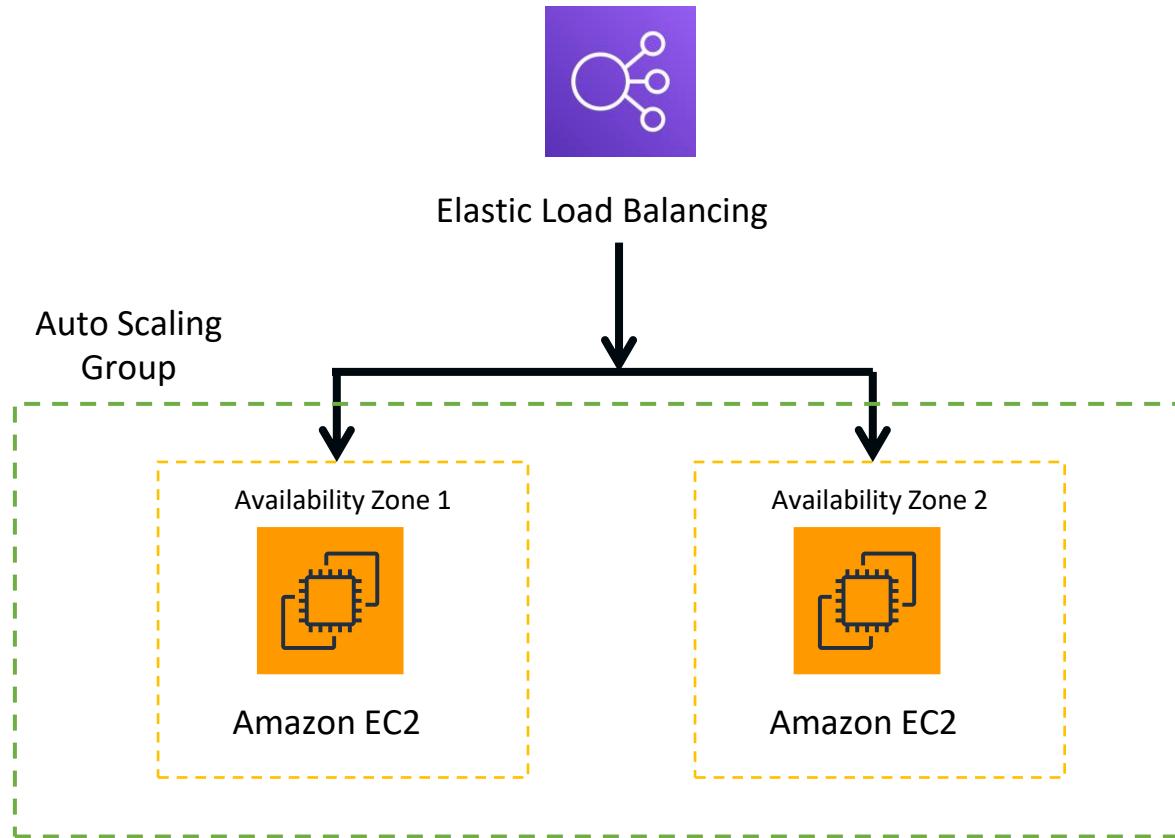
- Servers running your applications
- Database
- Load balancer
- Analyze each component and validate single point of failure

# Achieving High Availability on Cloud



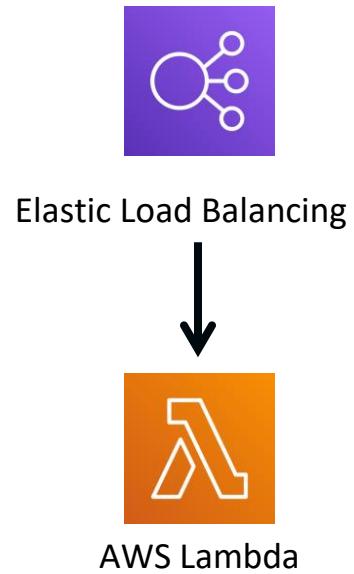
- Elastic Load Balancer is inherently highly available (managed by Cloud Provider)
- Auto Scaling Group makes the server scalable, not highly available
  - There is a delay to spin server up

# Achieving High Availability on Cloud



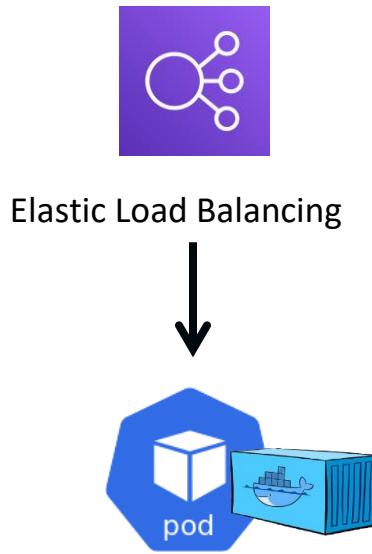
- Achieve high availability but costs extra money
- What is an option which is automatically highly available i.e. HA managed by Cloud Provider?

# Achieving High Availability on Cloud

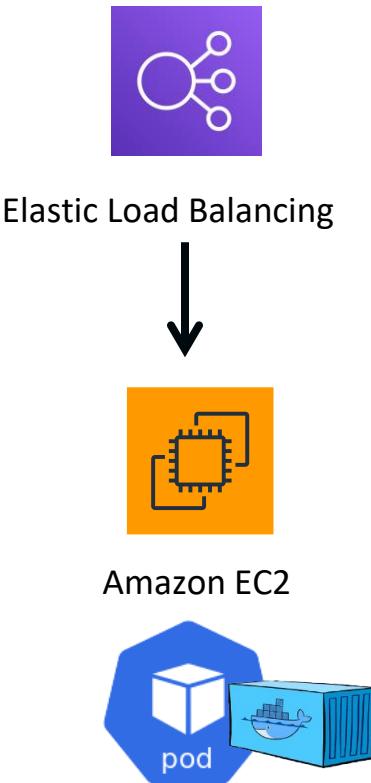


- How about Kubernetes?

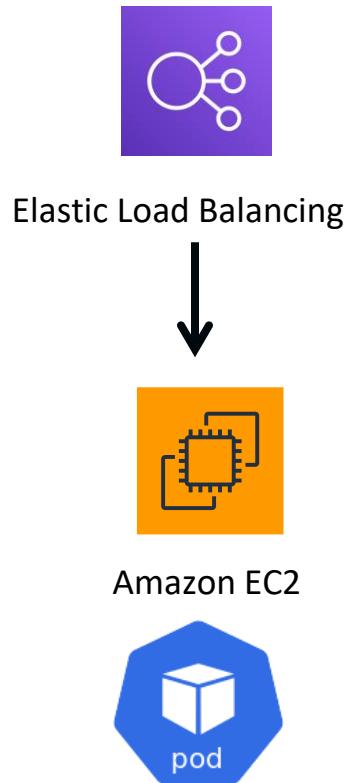
# Achieving High Availability on Cloud



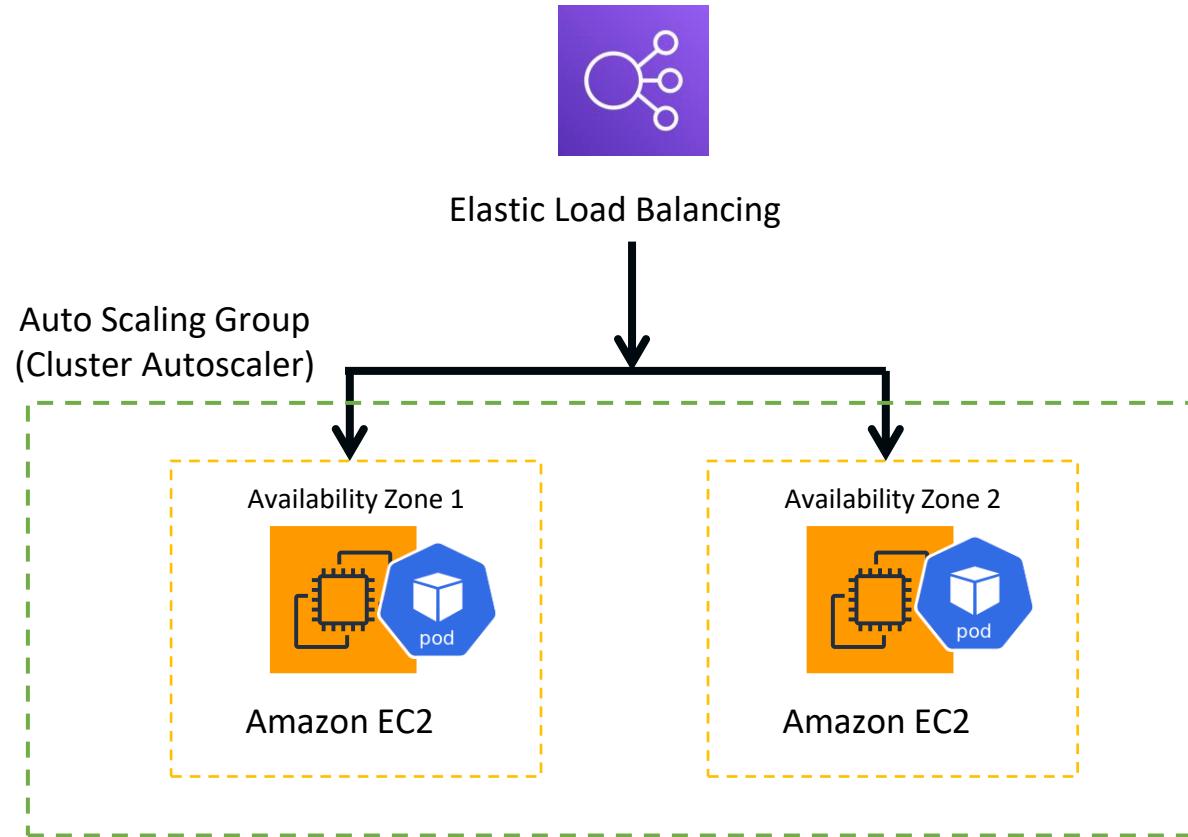
# Achieving High Availability on Cloud



# Achieving High Availability on Cloud



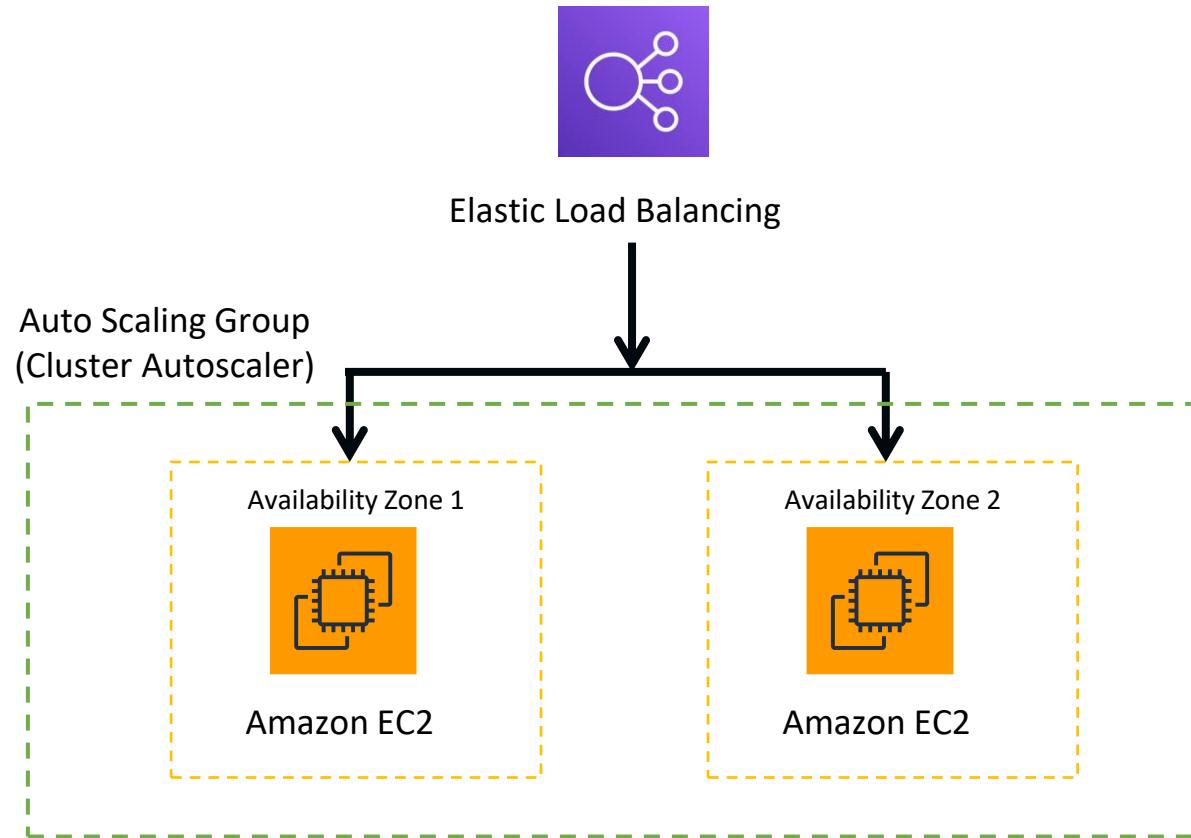
# Achieving High Availability on Cloud



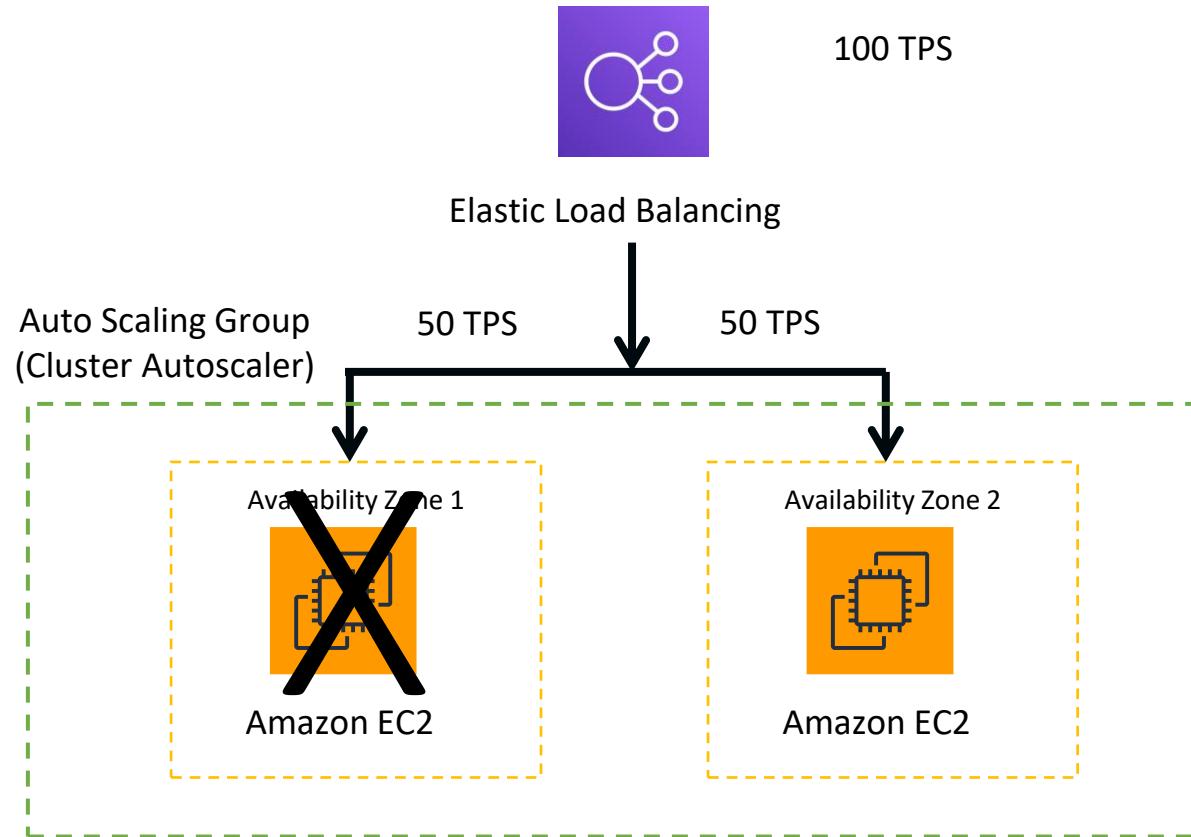
- Don't over index on cost when you design or answer interview question

# High Availability Vs Fault Tolerance

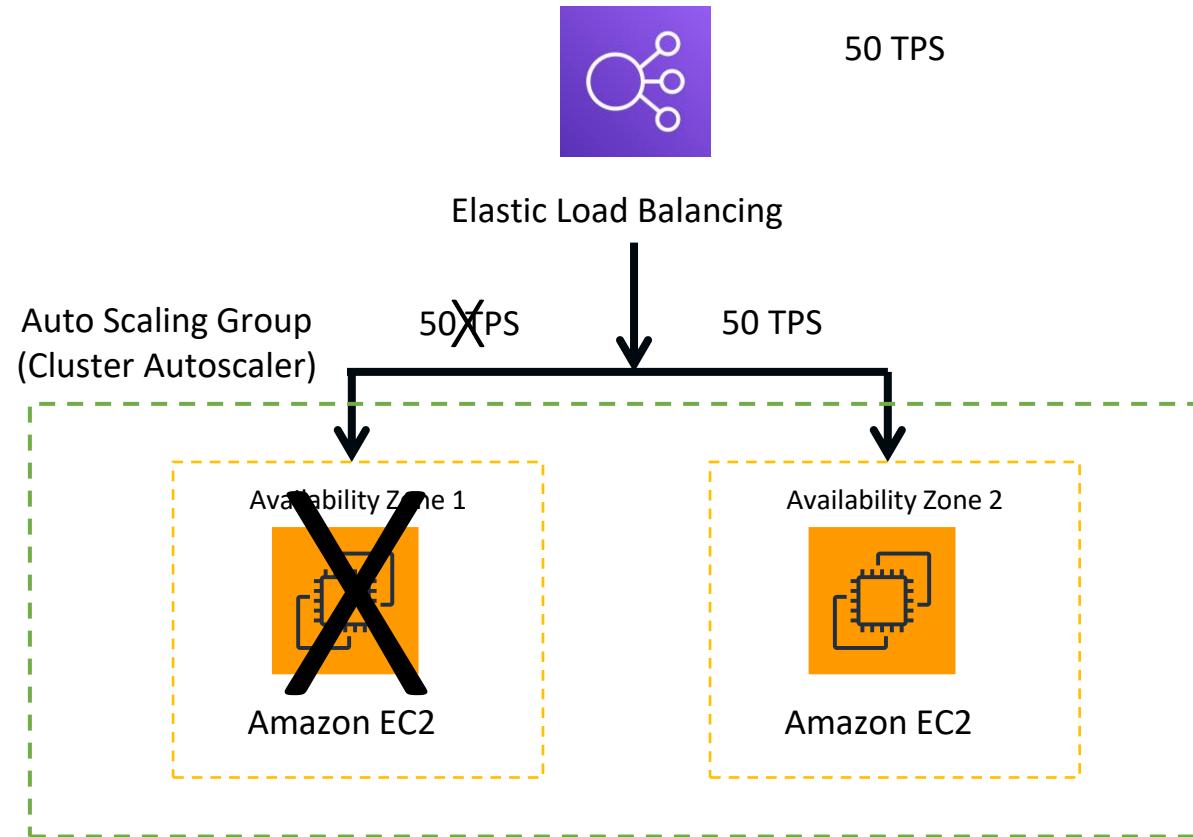
# High Availability



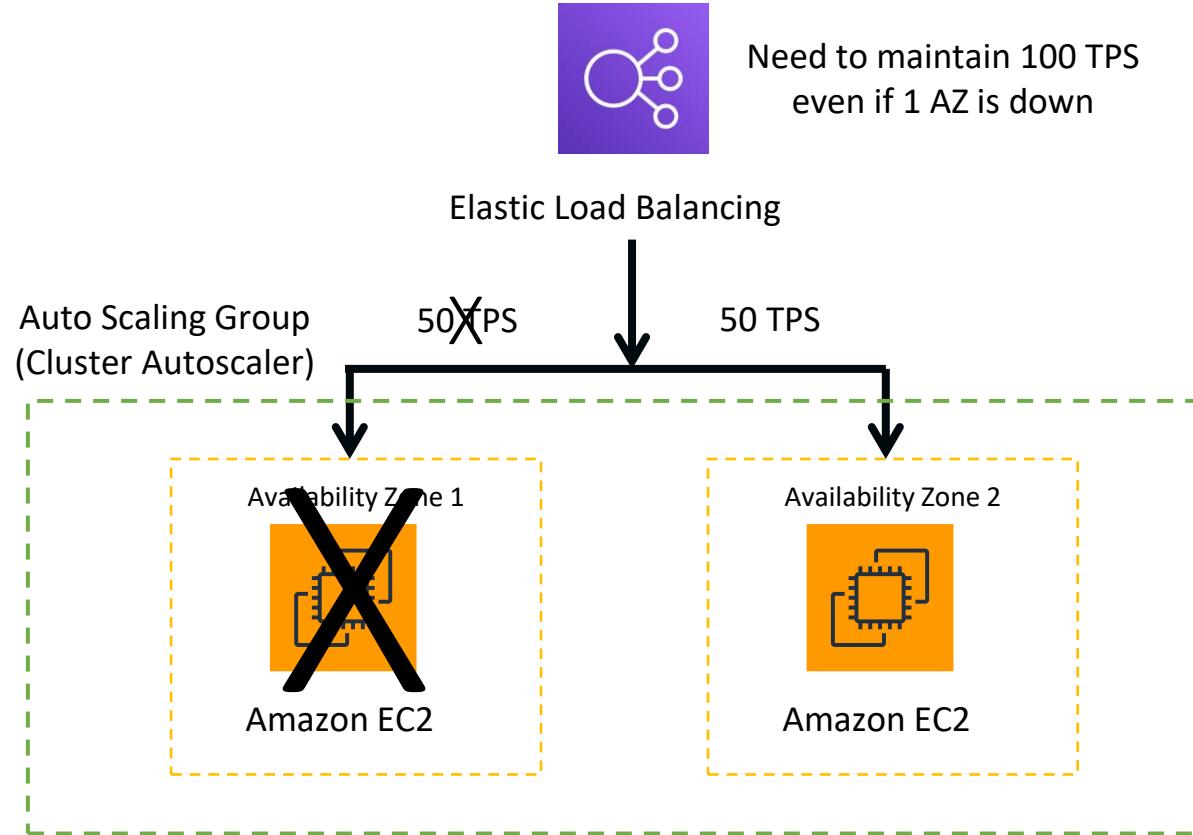
# High Availability



# High Availability



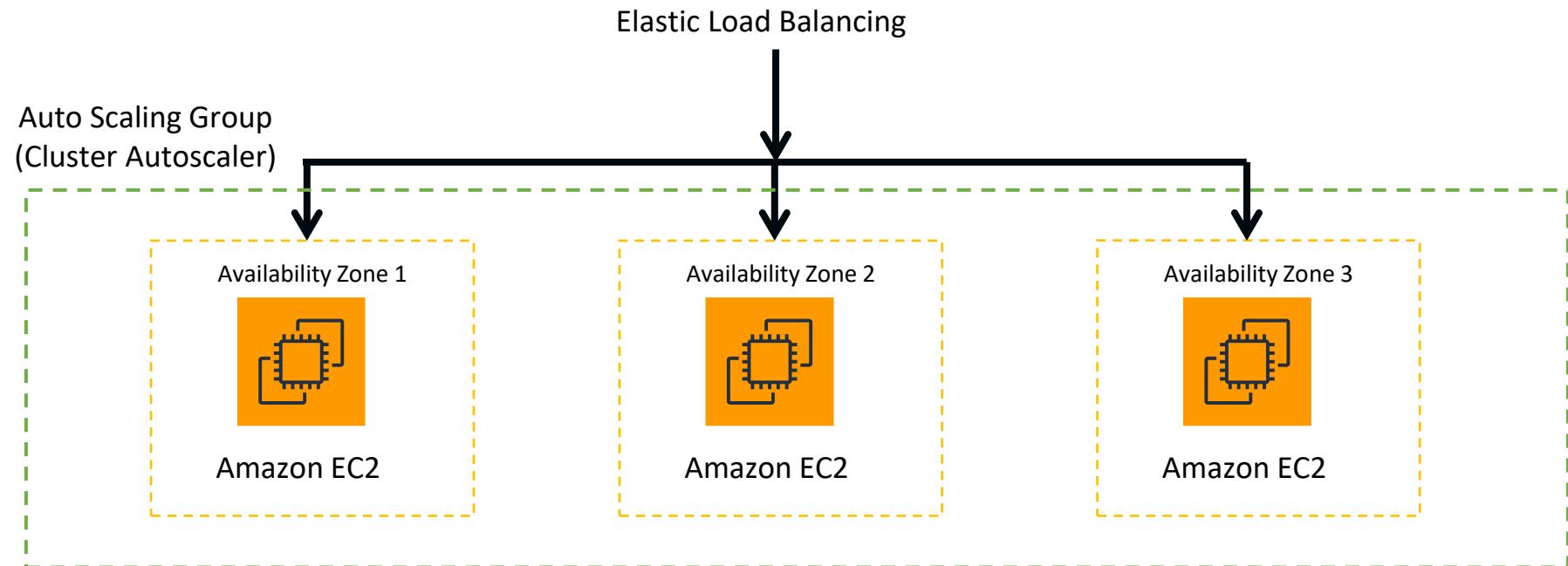
# Fault Tolerant



# Fault Tolerant



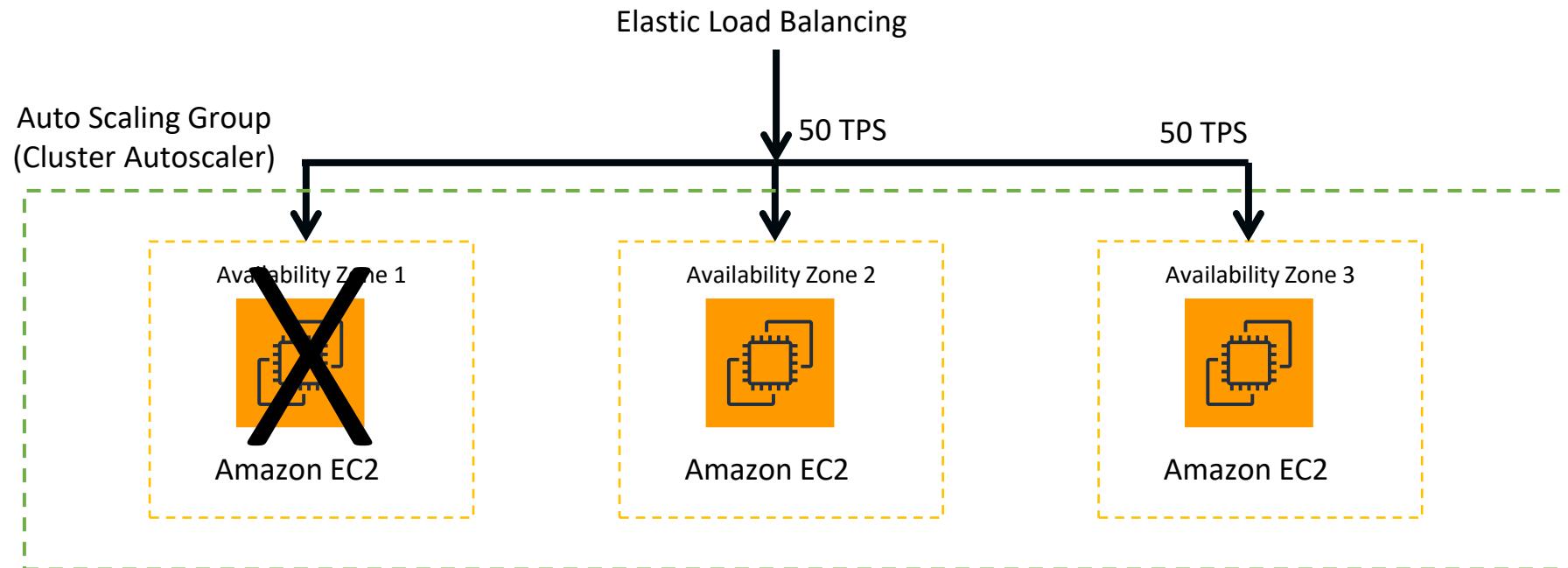
Need to maintain 100 TPS  
even if 1 AZ is down



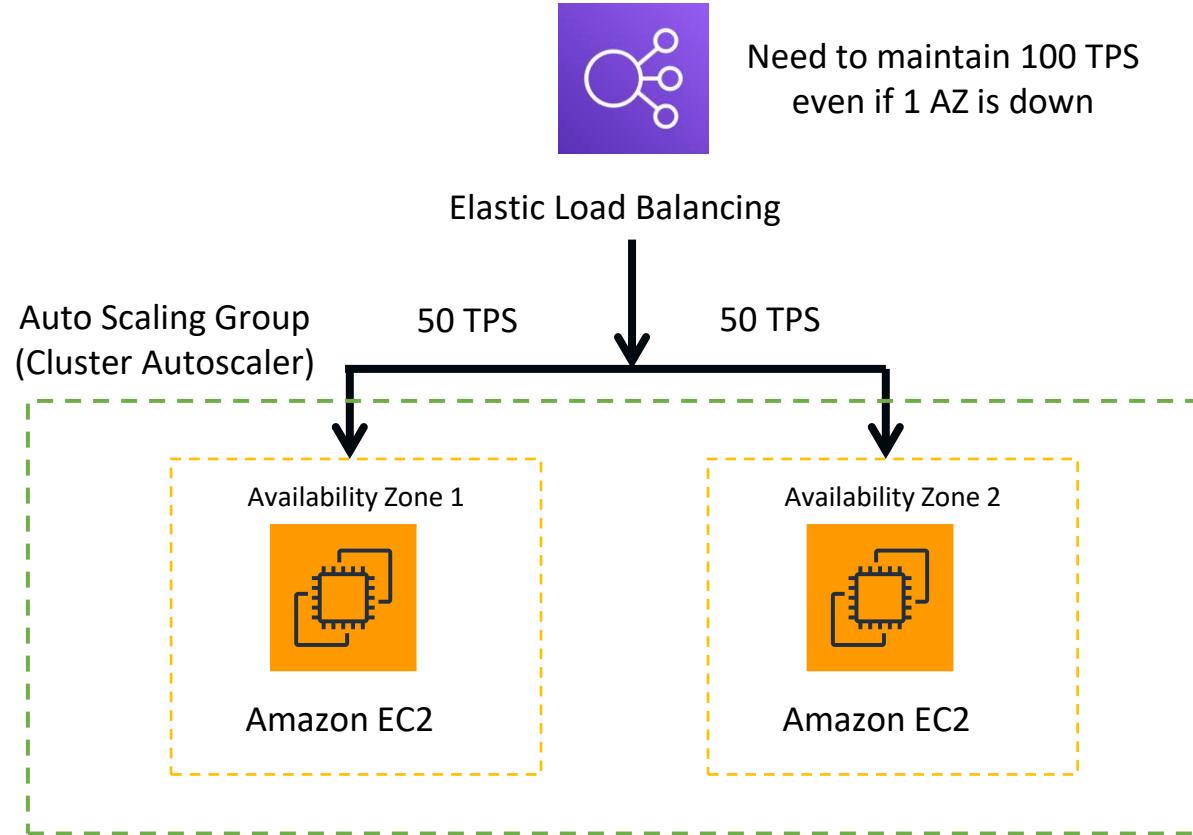
# Fault Tolerant



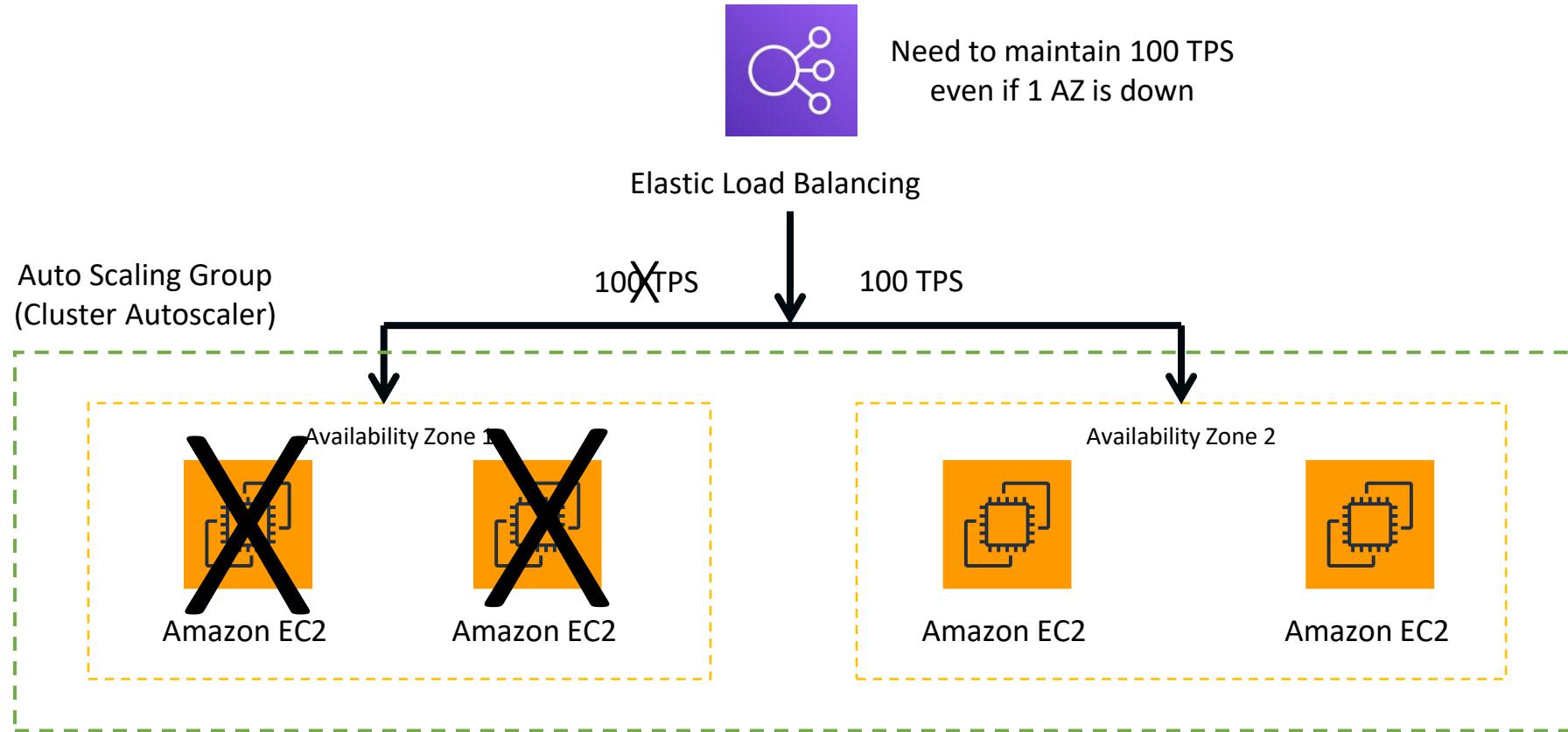
Need to maintain 100 TPS  
even if 1 AZ is down



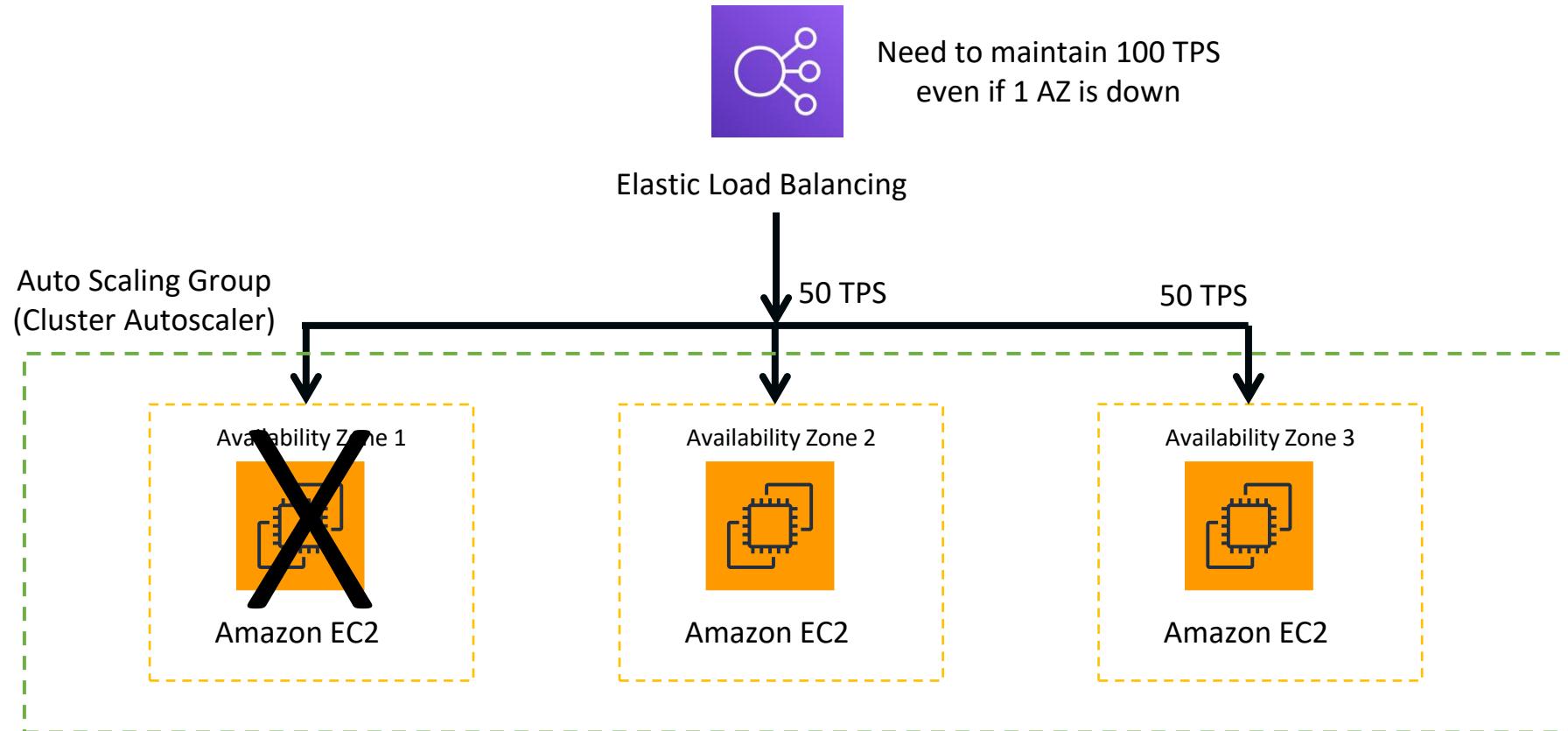
# Fault Tolerant



# Fault Tolerant



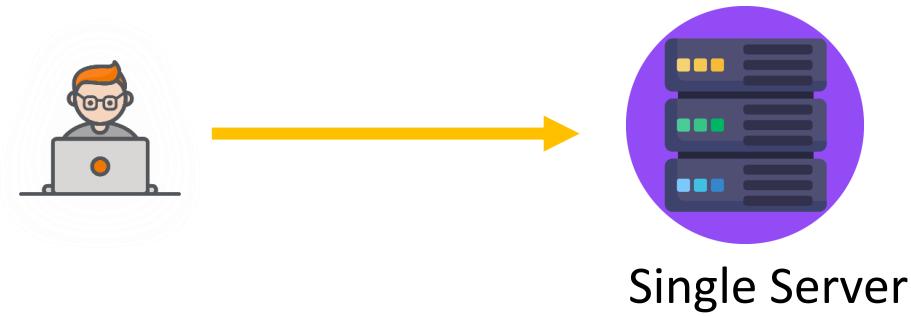
# Fault Tolerant



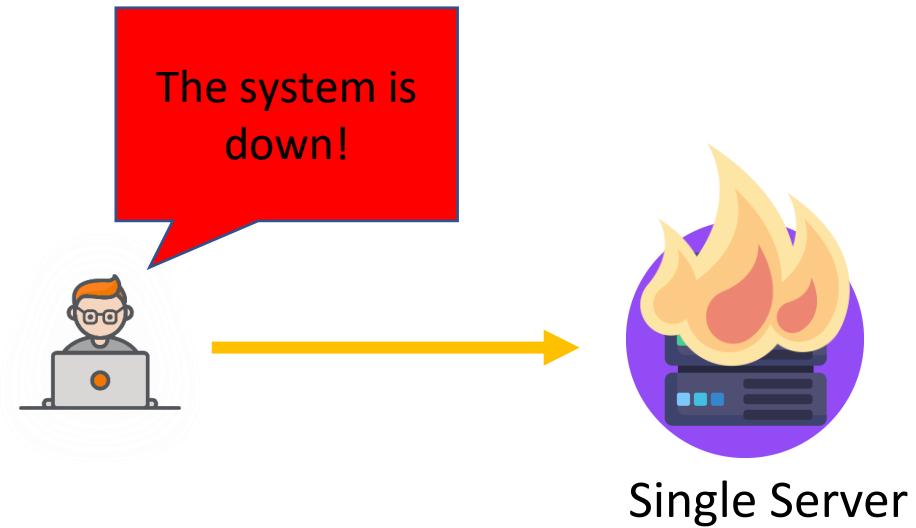
- Fault tolerant system is more expensive than highly available system

# Distributed Systems

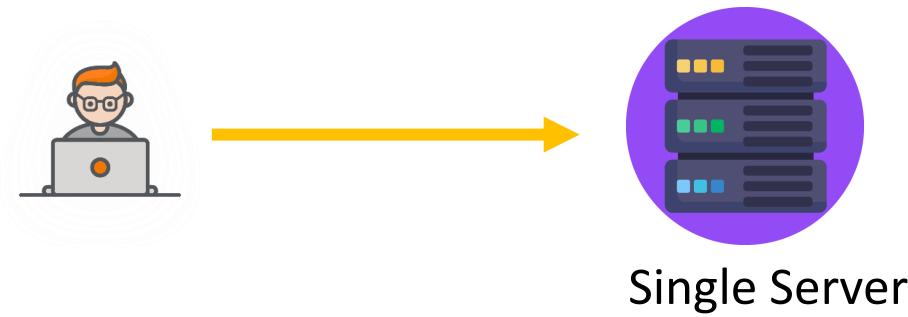
# Centralized Systems



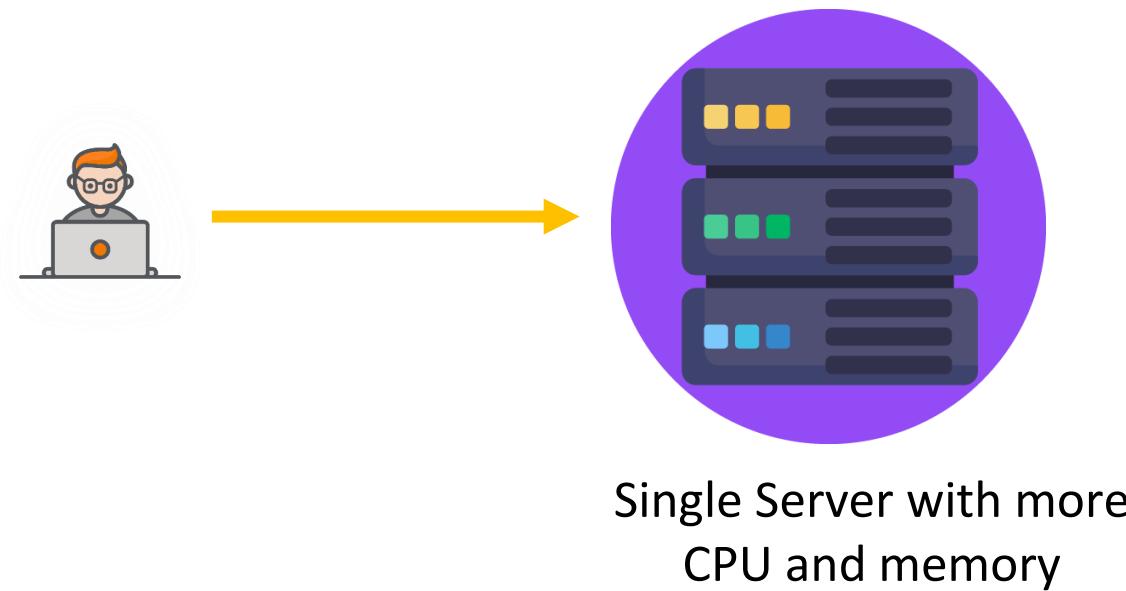
# Single Point of Failure



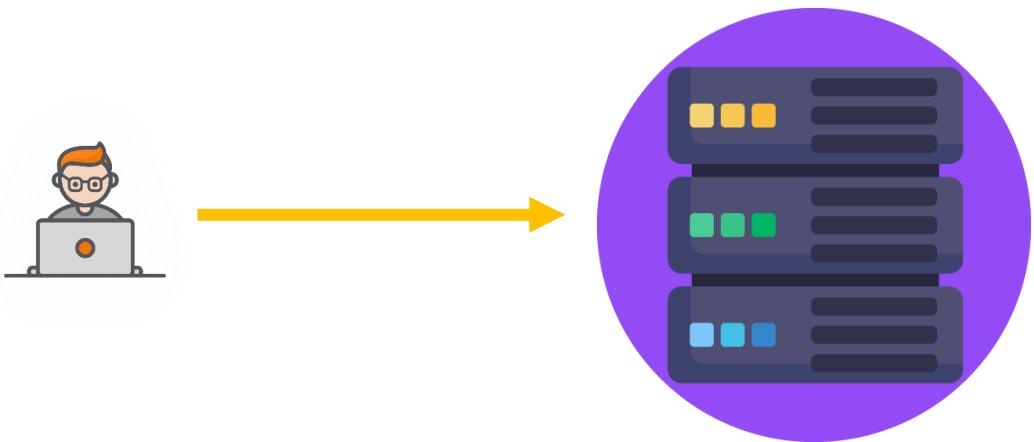
# Centralized Systems Scaling



# Centralized Systems Scaling



# Centralized Systems Examples



Single Server with more  
CPU and memory



Apps on local machine

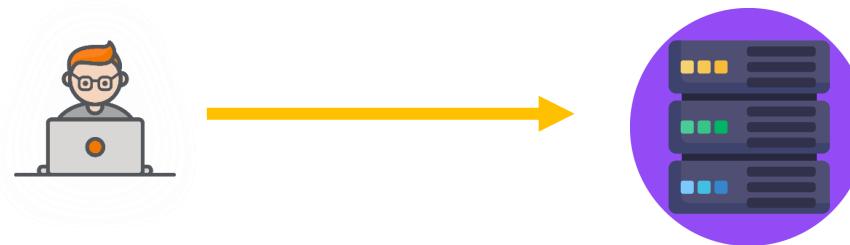


IBM DB2 on Mainframe

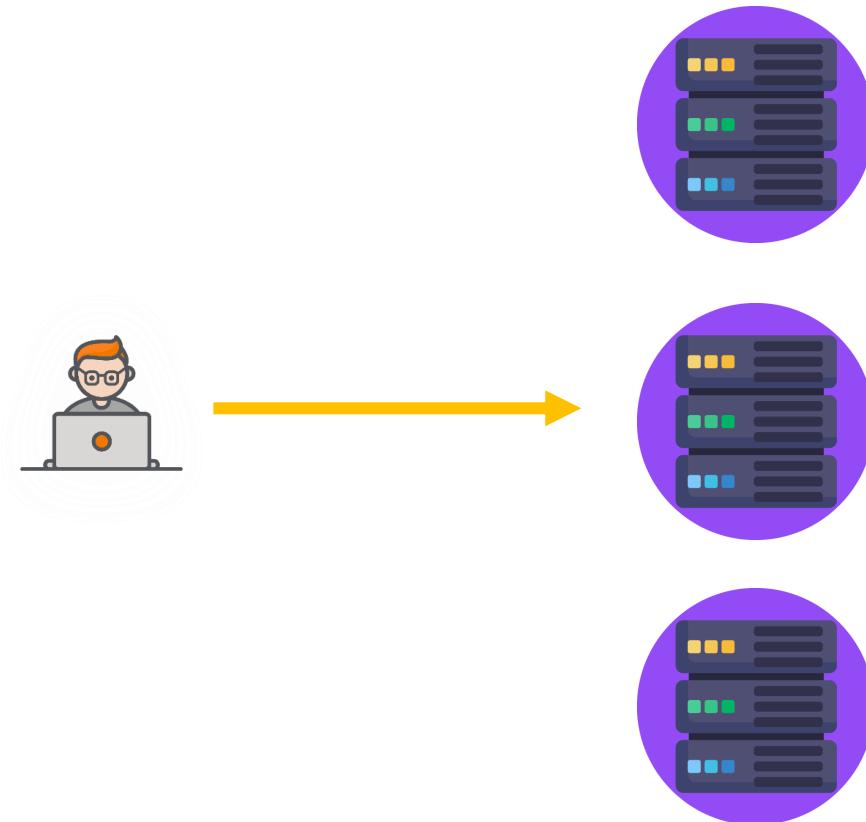


Any apps running on single  
datacenter server

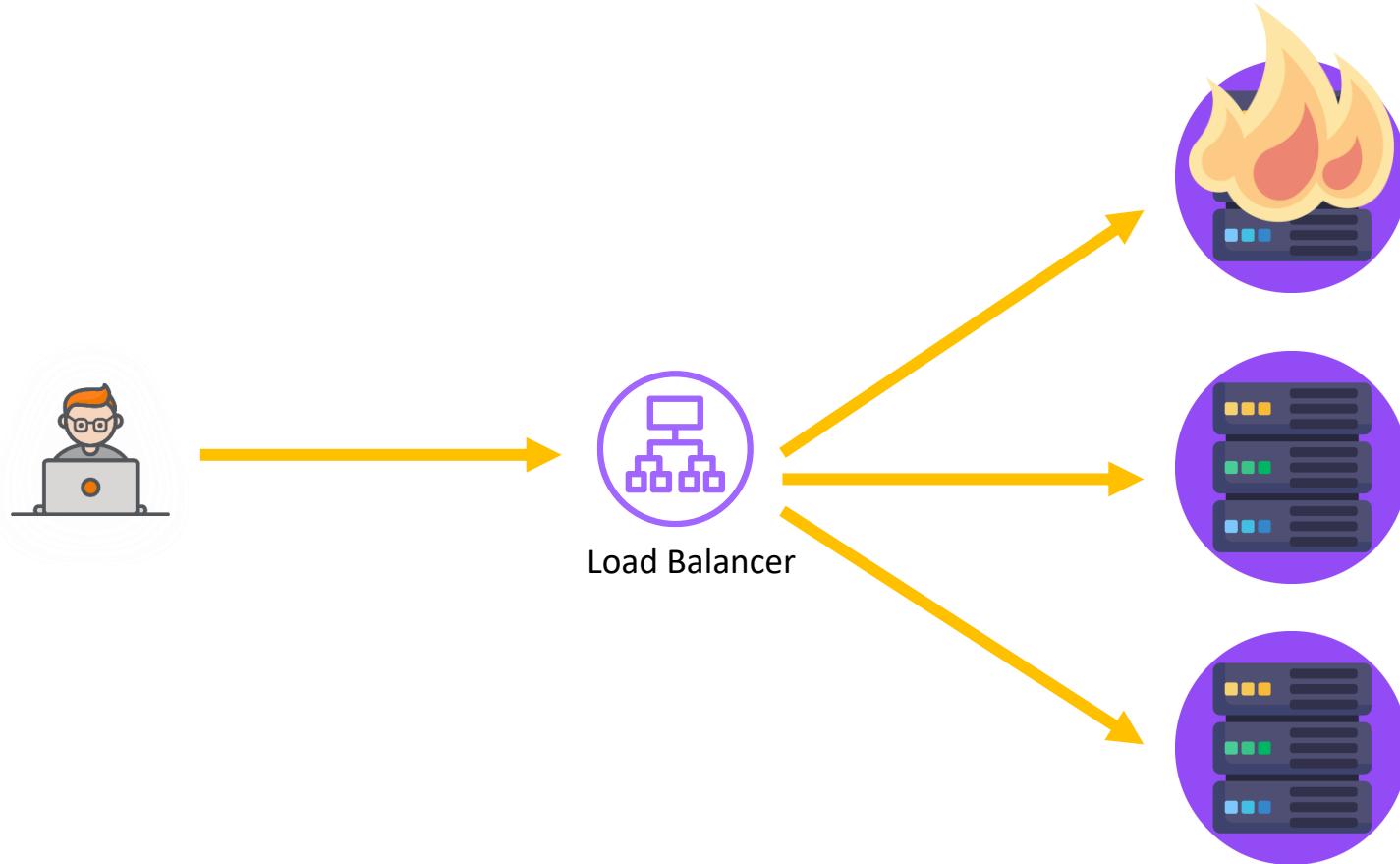
# Distributed Systems



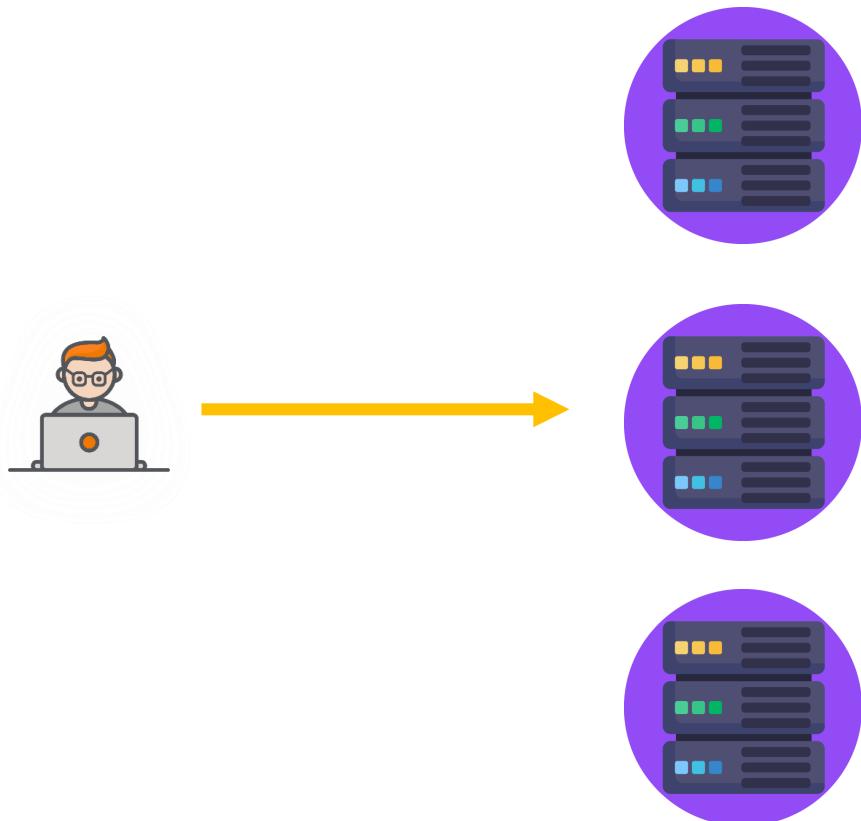
# Distributed Systems



# Distributed Systems



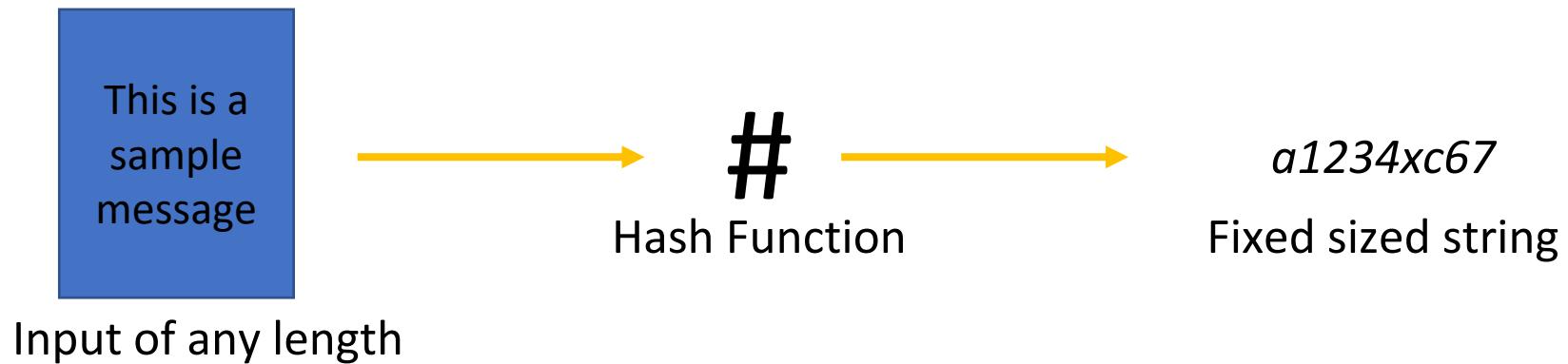
# Distributed Systems



- System “distributed” on many servers
- Scale out by adding more servers
  - Horizontal scaling
- No single point of failure
- Most modern systems are distributed

# Hashing

# Hashing



# Hashing

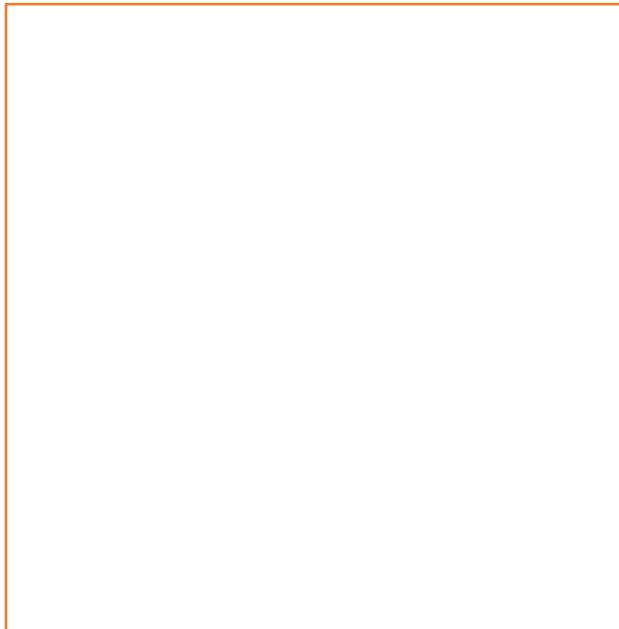
- Same input will always create same output
- Little change in input should create a vastly different output
- Hash function should be fast

More Importantly – How is hashing applied in system design??

# Table Partitions



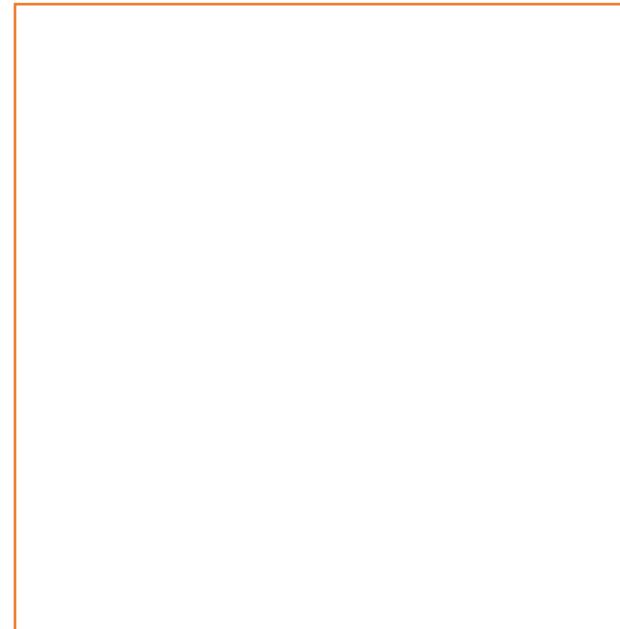
DynamoDB  
Table



Partition1

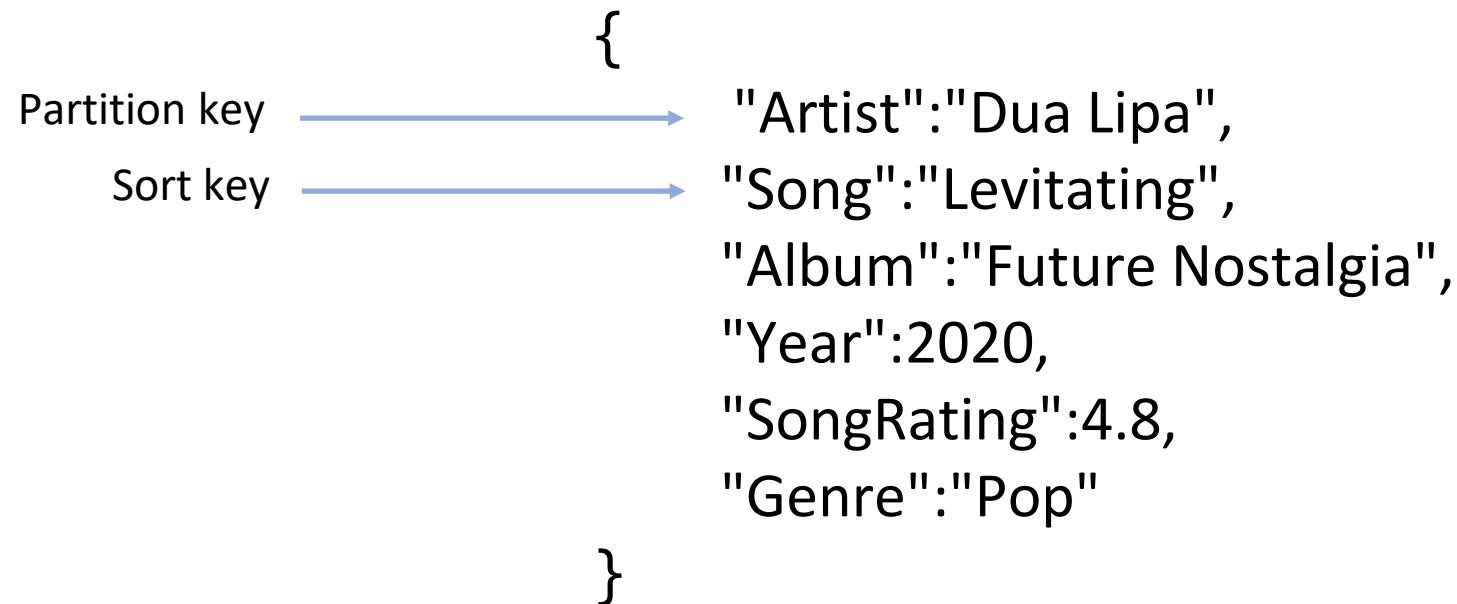


Partition2



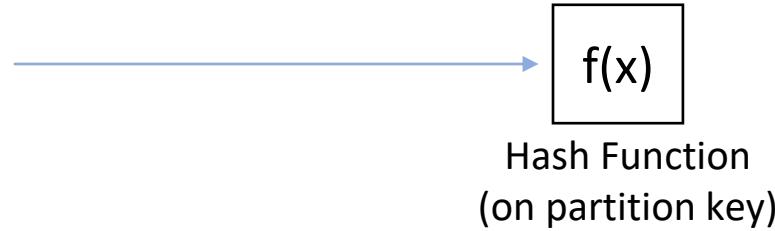
Partition3

# DynamoDB Primary Key



# DynamoDB Partitions

```
{  
    "Artist":"Dua Lipa",  
    "Song":"Levitating",  
    "Album":"Future Nostalgia",  
    "Year":2020,  
    "SongRating":4.8,  
    "Genre":"Pop"  
}
```

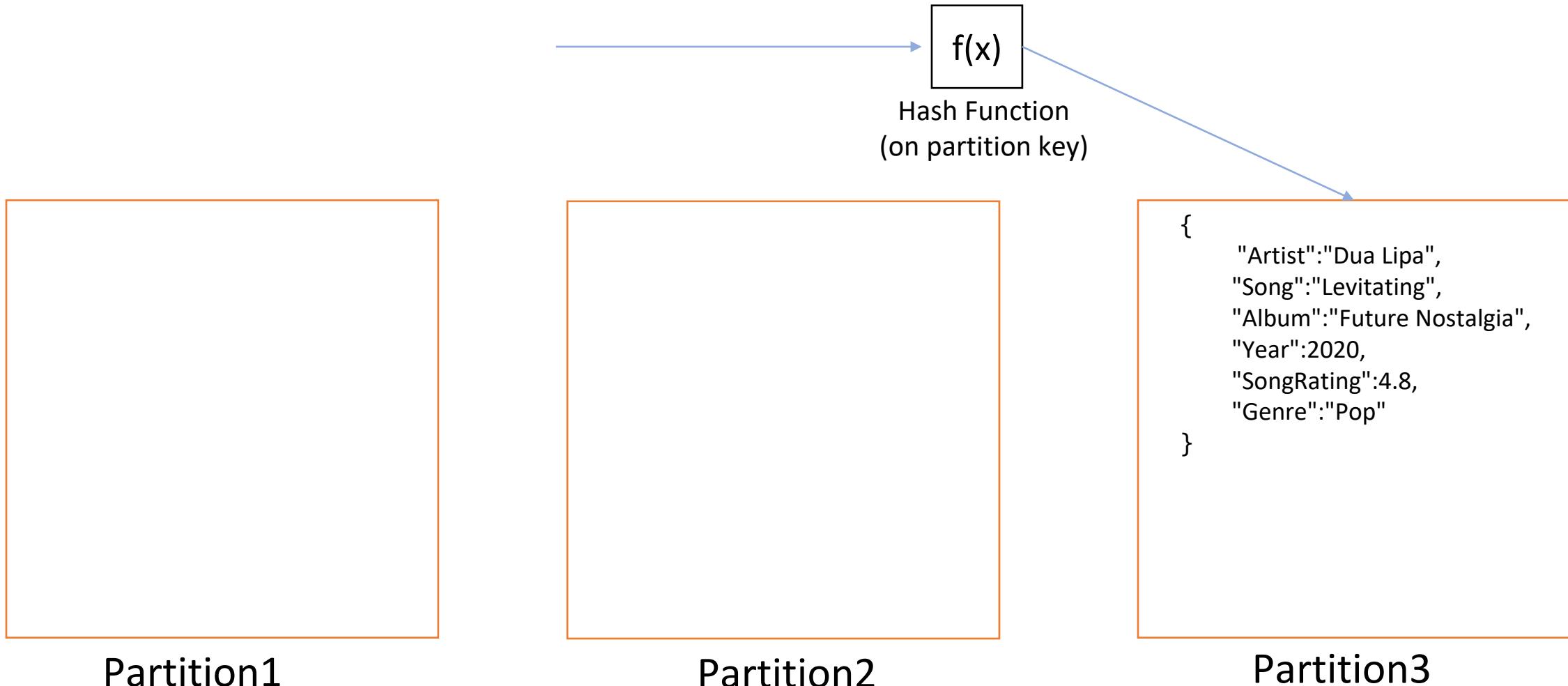


Partition1

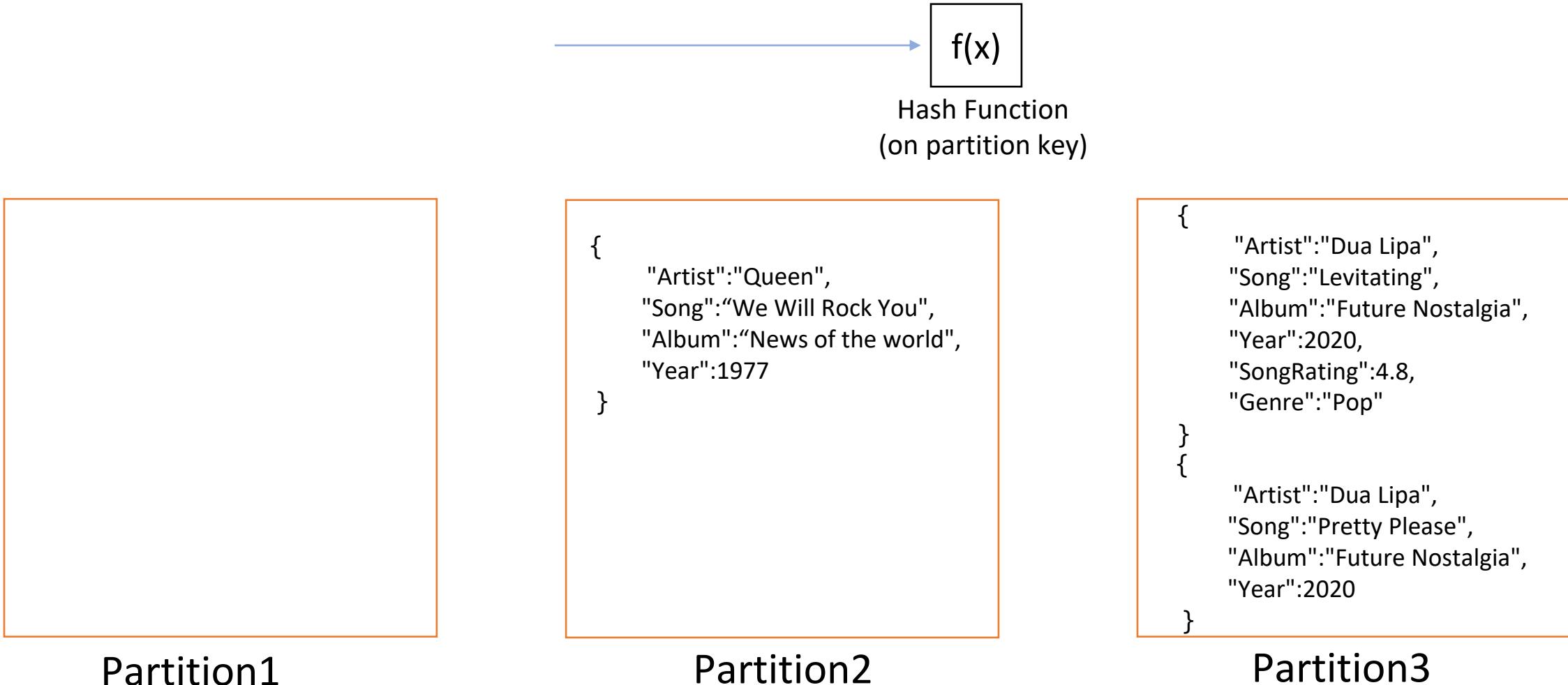
Partition2

Partition3

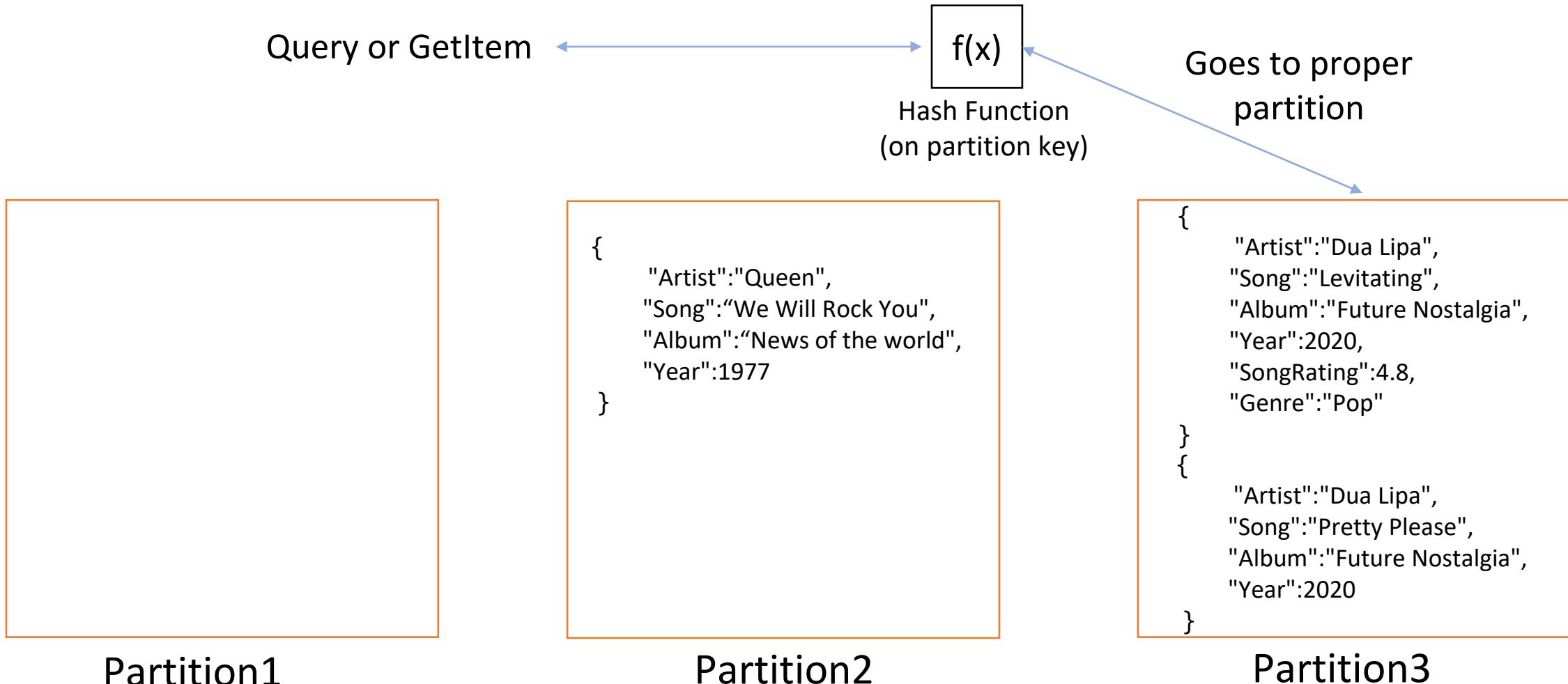
# DynamoDB Partitions



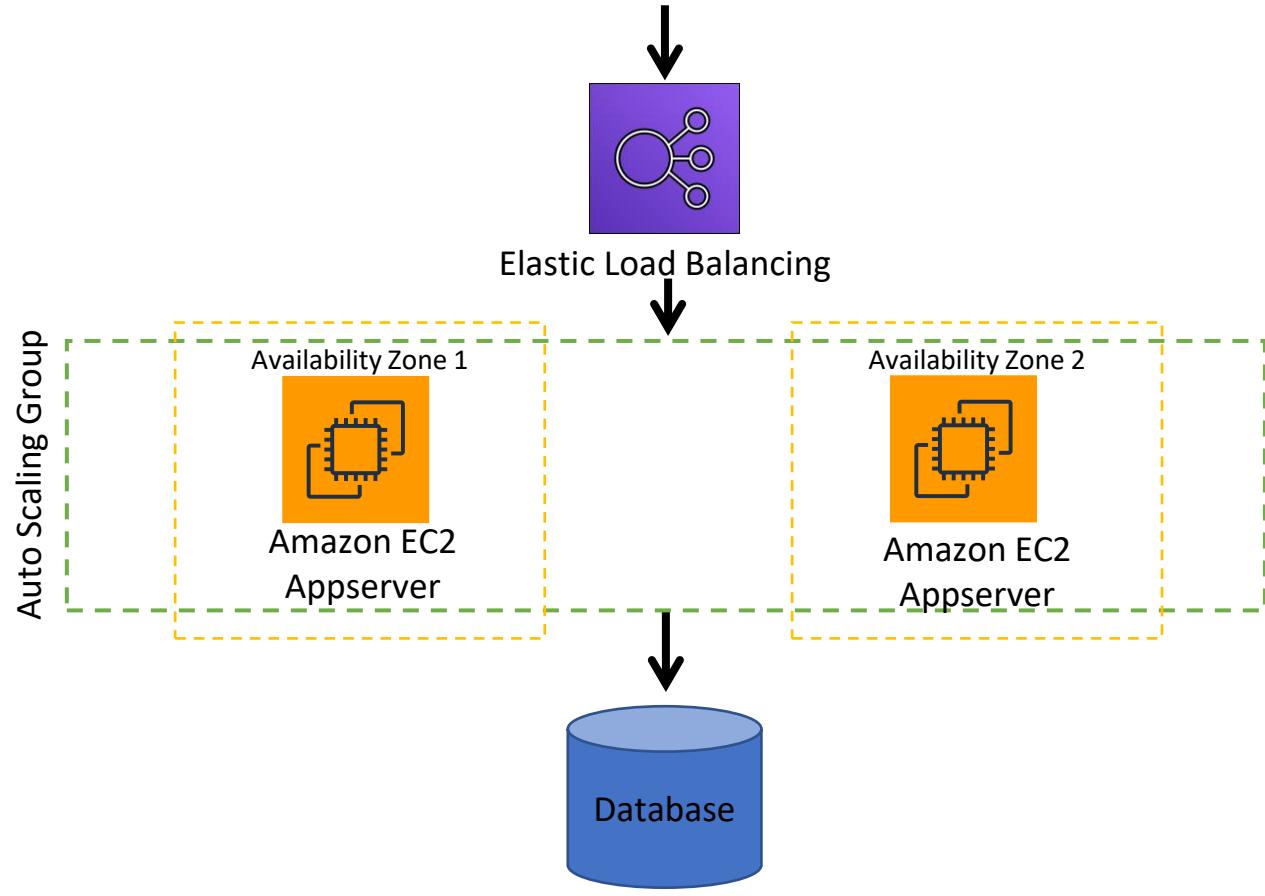
# DynamoDB Partitions

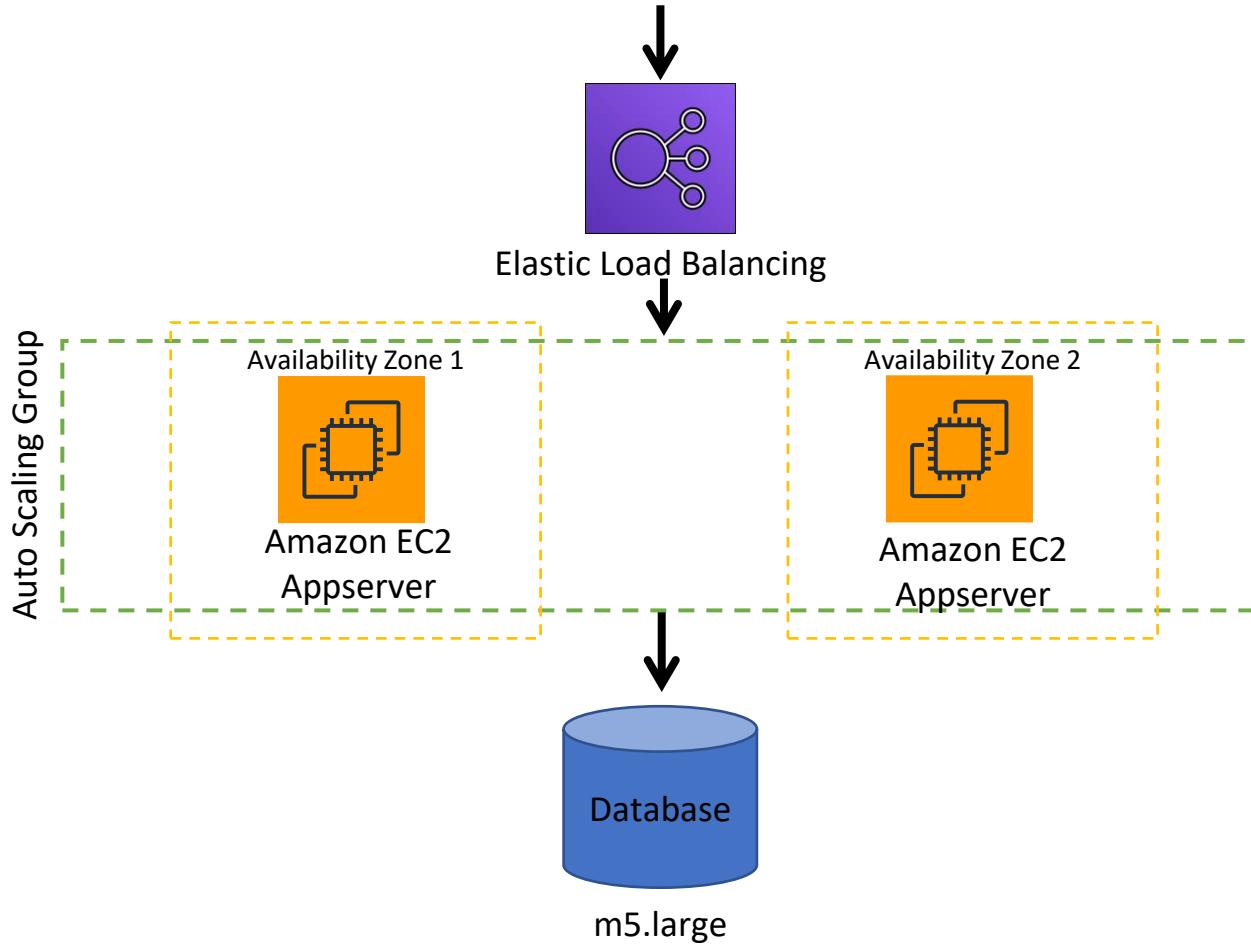


# DynamoDB Partitions

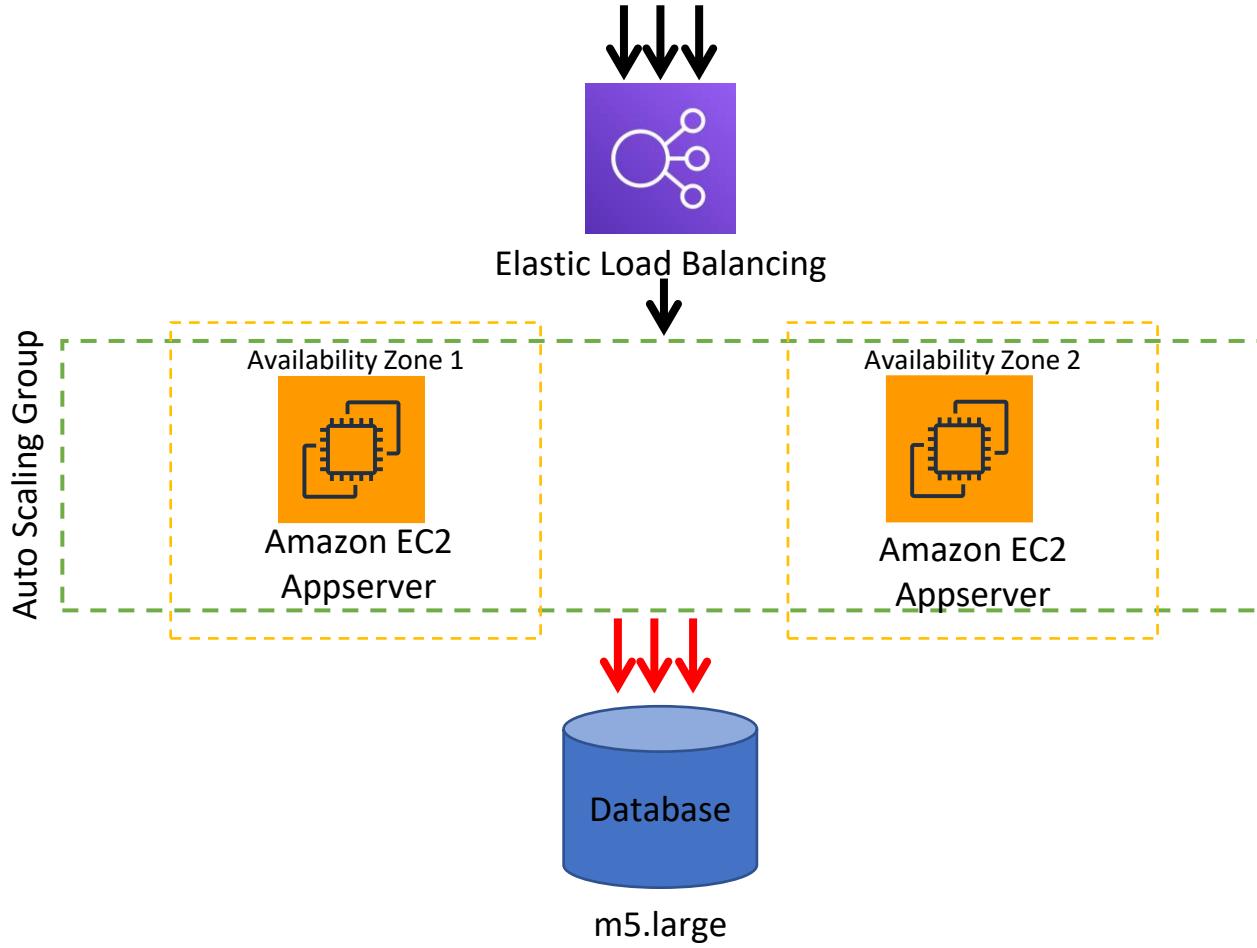


# Database Sharding (Horizontal Partitioning)

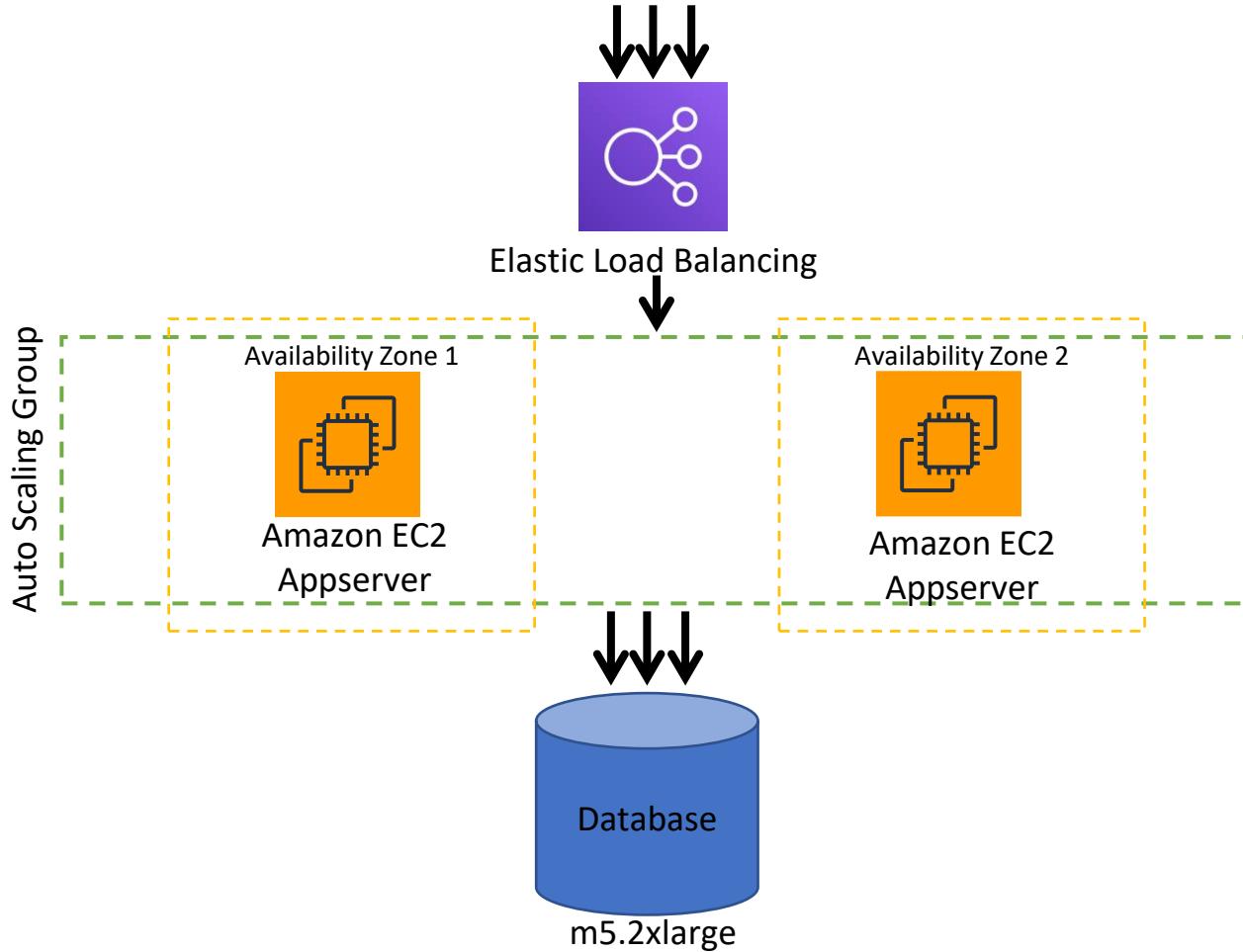




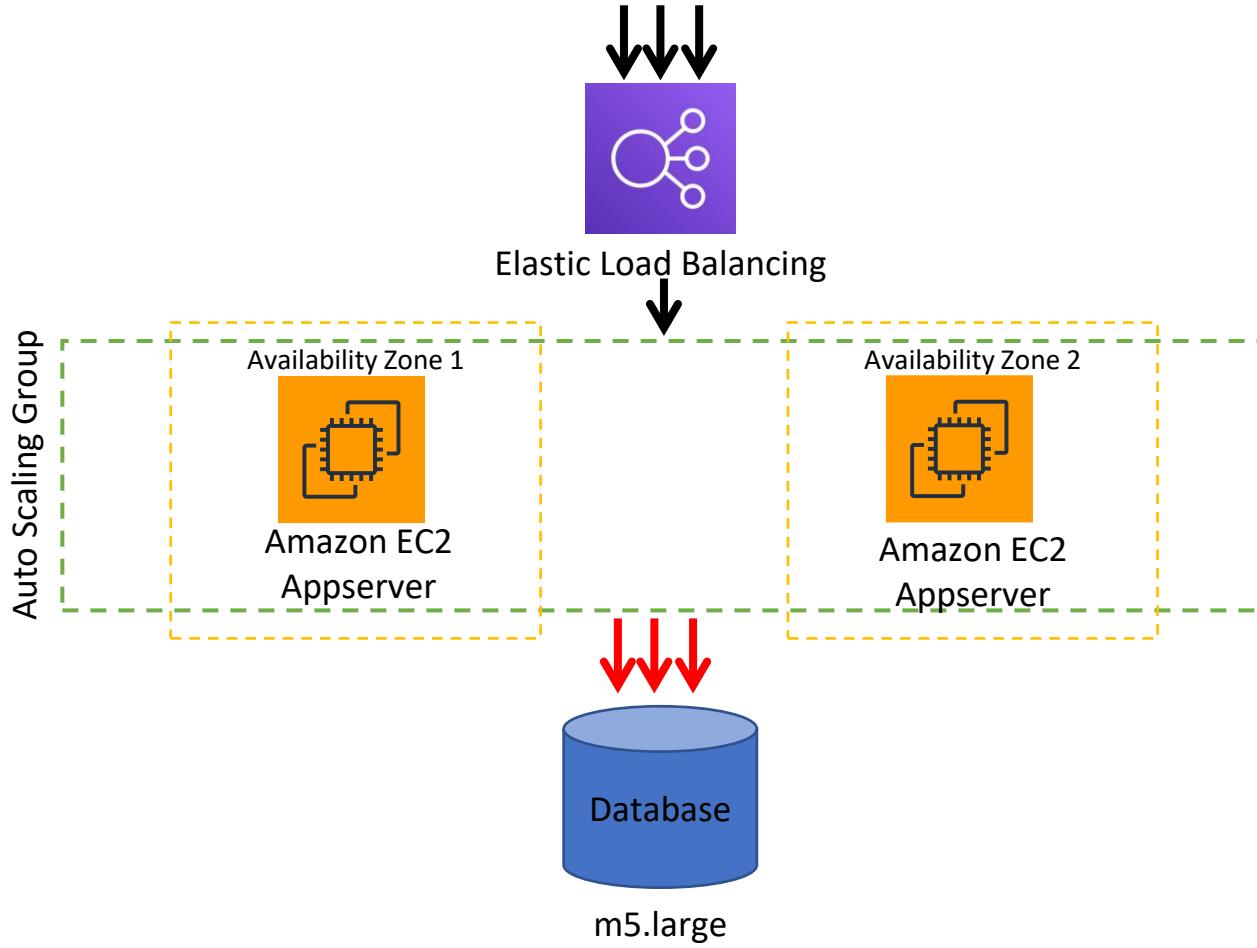
ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20
3	Chocolate	10
4	TV	400
5	Couch	100



ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20
3	Chocolate	10
4	TV	400
5	Couch	100



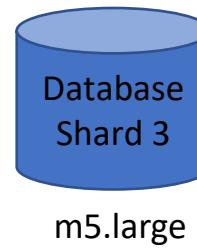
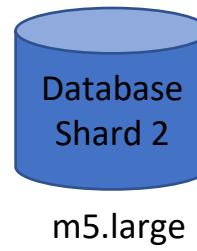
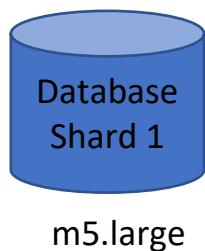
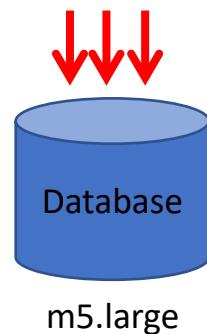
ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20
3	Chocolate	10
4	TV	400
5	Couch	100



ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20
3	Chocolate	10
4	TV	400
5	Couch	100

# Database Sharding

ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20
3	Chocolate	10
4	TV	400
5	Couch	100

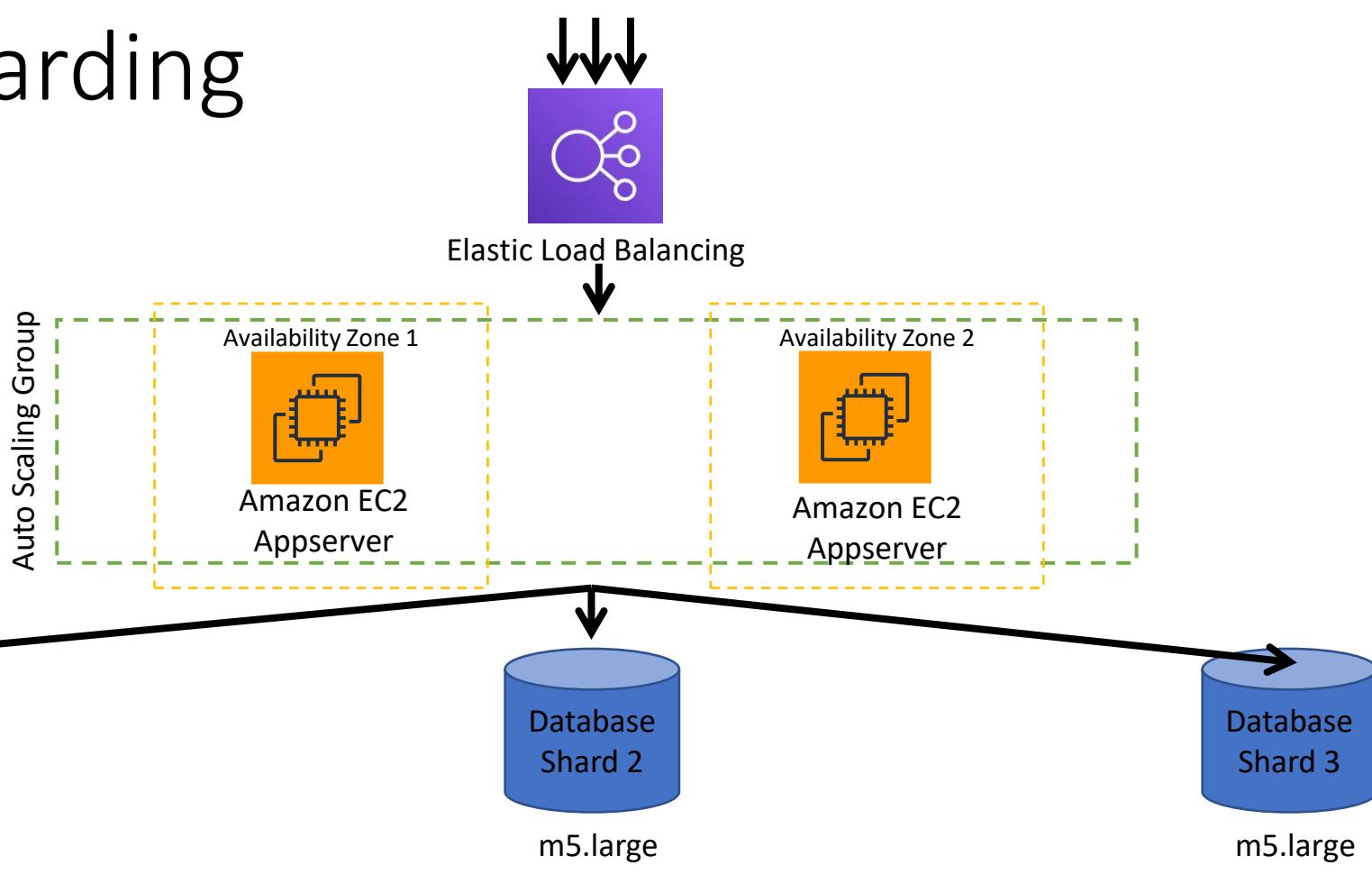


ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20

ID	NAME	PRICE
3	Chocolate	10
4	TV	400

ID	NAME	PRICE
5	Couch	100

# Database Sharding

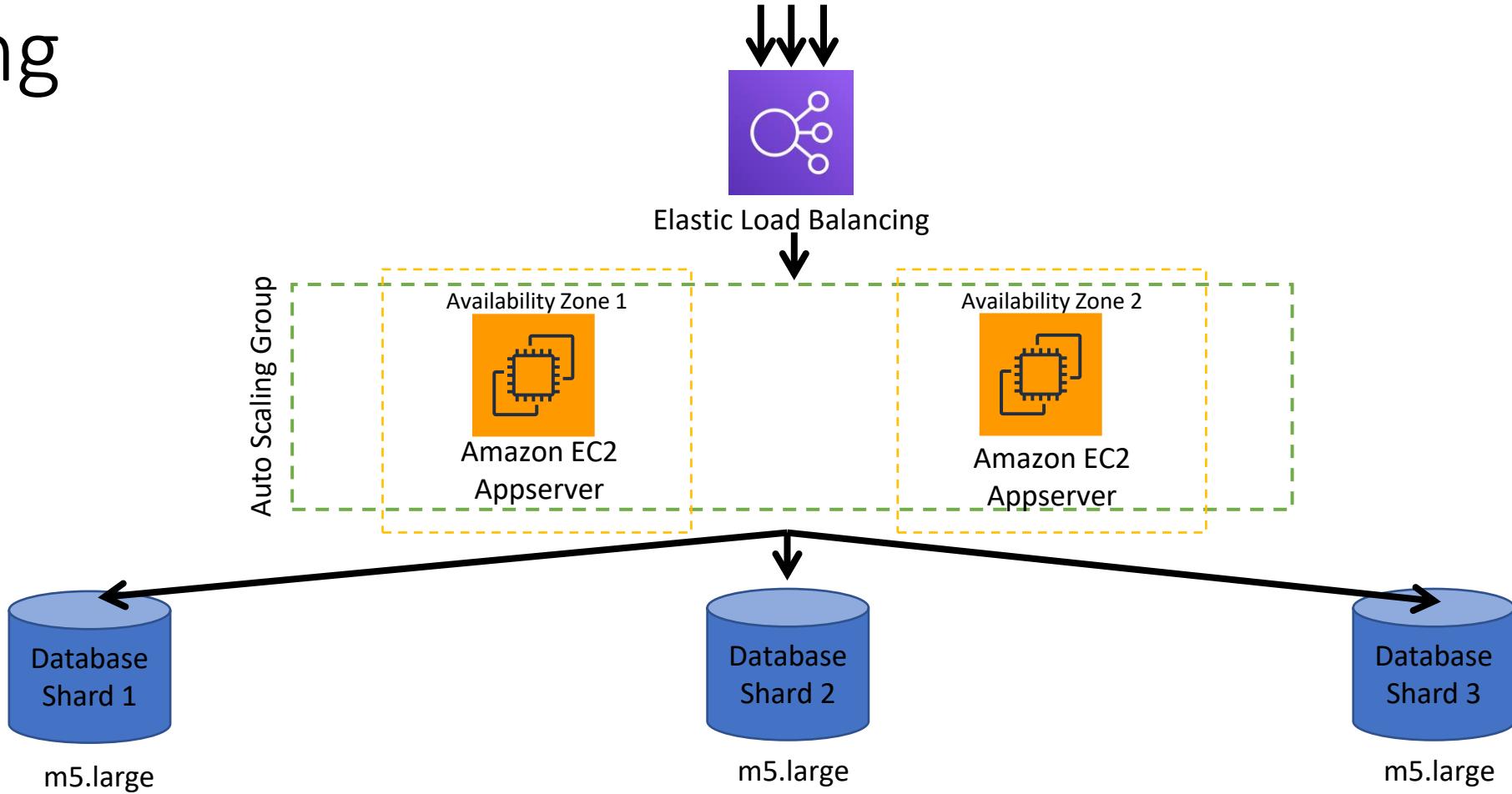


ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20

ID	NAME	PRICE
3	Chocolate	10
4	TV	400

ID	NAME	PRICE
5	Couch	100

# Hashing



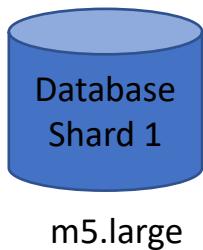
ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20

ID	NAME	PRICE
3	Chocolate	10
4	TV	400

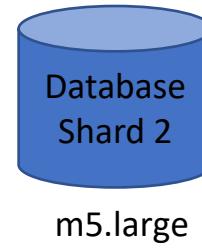
ID	NAME	PRICE
5	Couch	100

# Advantages

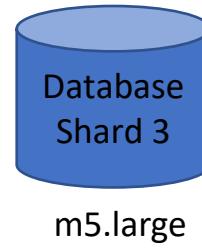
- Scaling horizontally supports distributed computing
- Faster query response times
- Limited blast radius during outage



m5.large



m5.large



m5.large

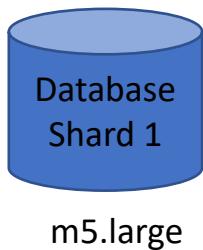
ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20

ID	NAME	PRICE
3	Chocolate	10
4	TV	400

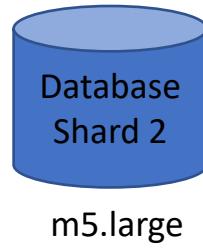
ID	NAME	PRICE
5	Couch	100

# Disadvantages

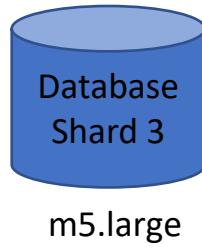
- Unbalanced shards
- Resharding is painful
- Implementing sharding logic is an overhead



m5.large



m5.large



m5.large

ID	NAME	PRICE
1	Alarm clock	25
2	Chair	20

ID	NAME	PRICE
3	Chocolate	10
4	TV	400

ID	NAME	PRICE
5	Couch	100

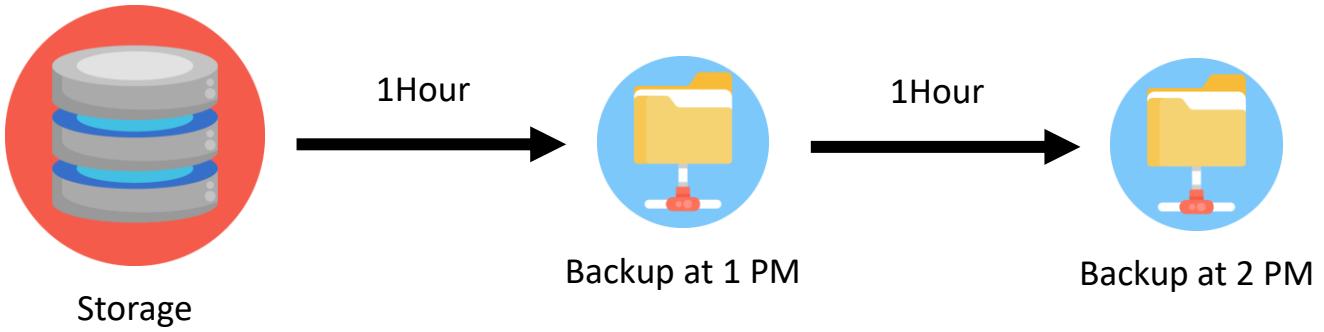
# Disaster Recovery

## RPO & RTO

# How will you achieve DR?

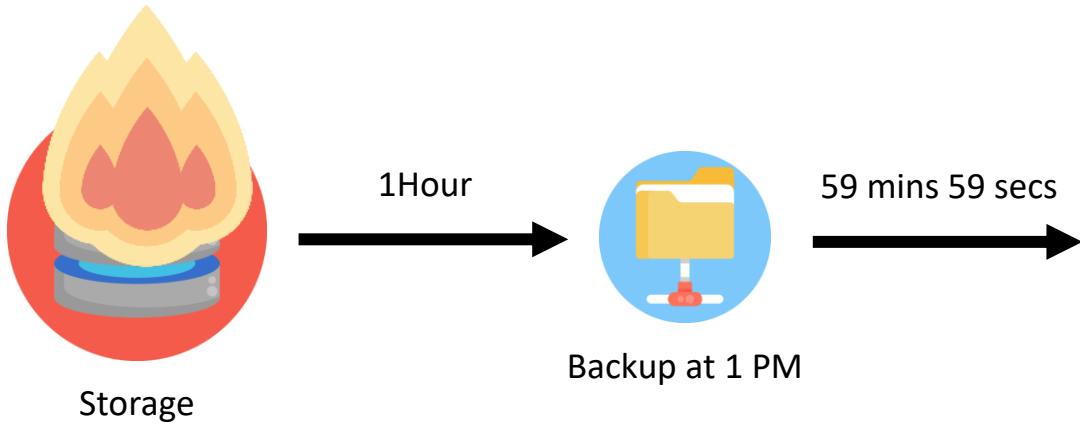
- There are multiple approaches to DR
  - Active-Active is NOT the only solution
- RPO/RTO plays a critical role

# RPO



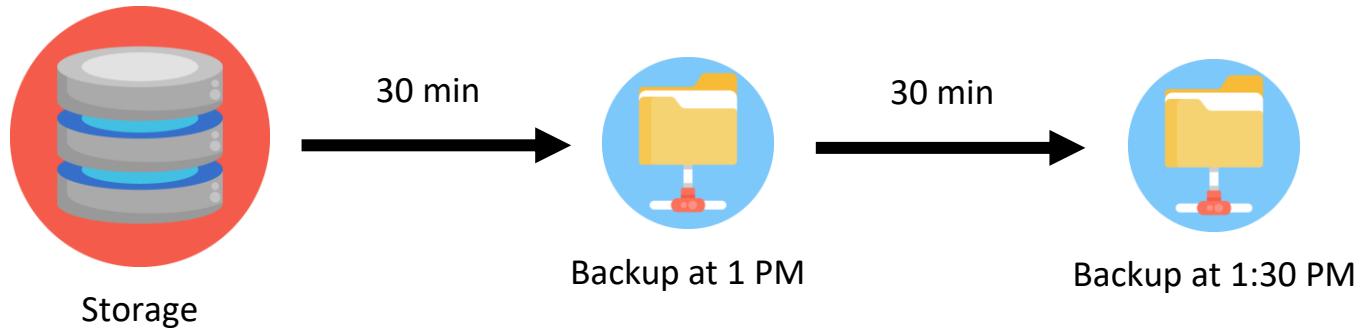
# RPO

1:59:59 PM

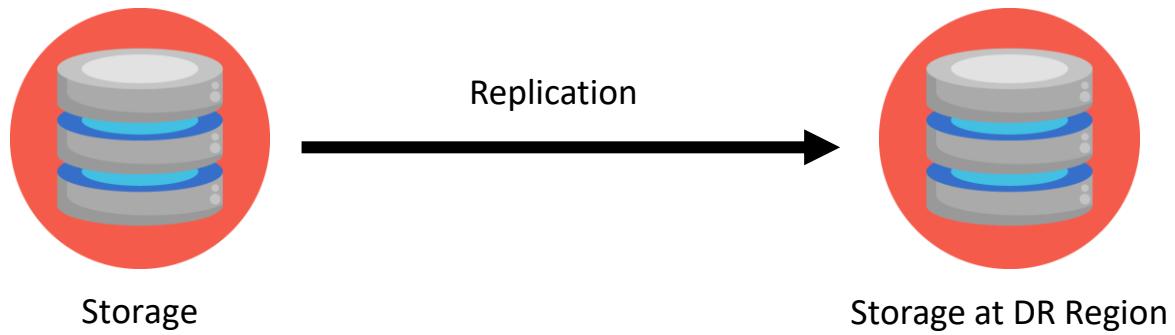


Recovery Point Objective = Amount of DATA that is allowed to be lost during a disaster measured in time  
= 1 Hour for this case

# How do you reduce RPO?



# How about real time RPO?



# RTO

1 PM



Application

Recovery Time Objective = Amount of TIME application can be down during a disaster

# RTO

2 PM

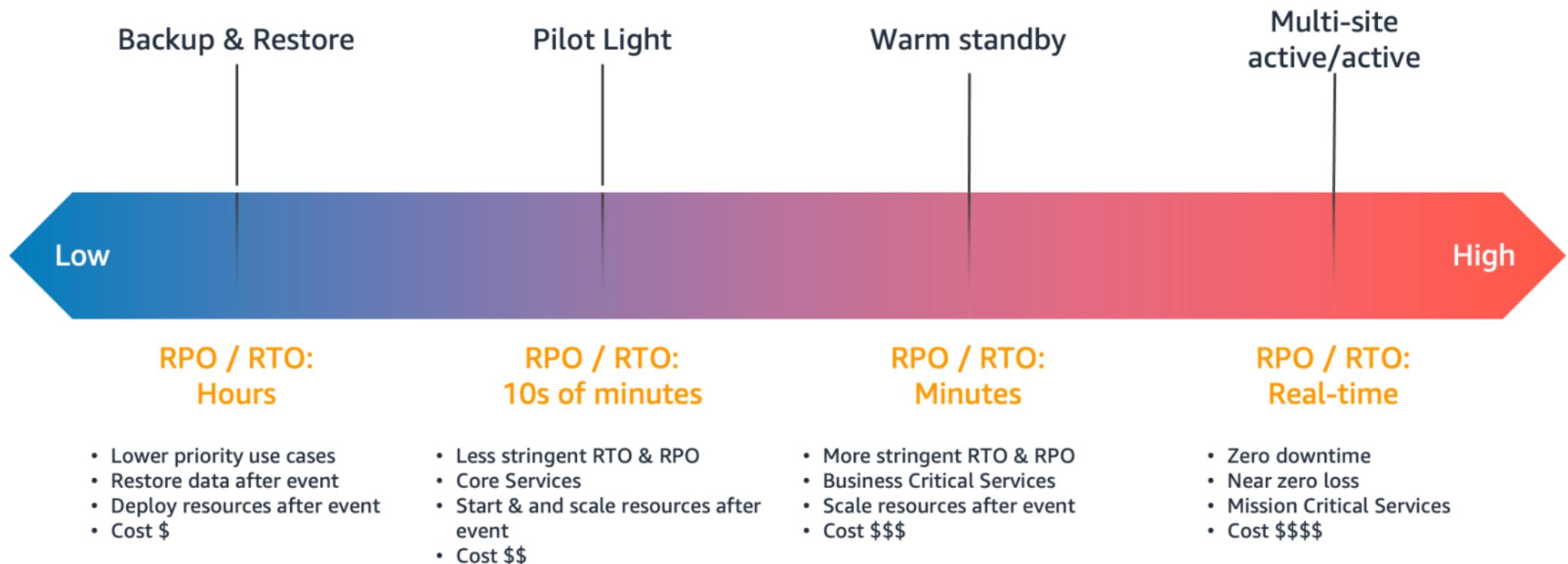


Application

Recovery Time Objective = Amount of TIME application can be down during a disaster  
= 1 Hour for this case

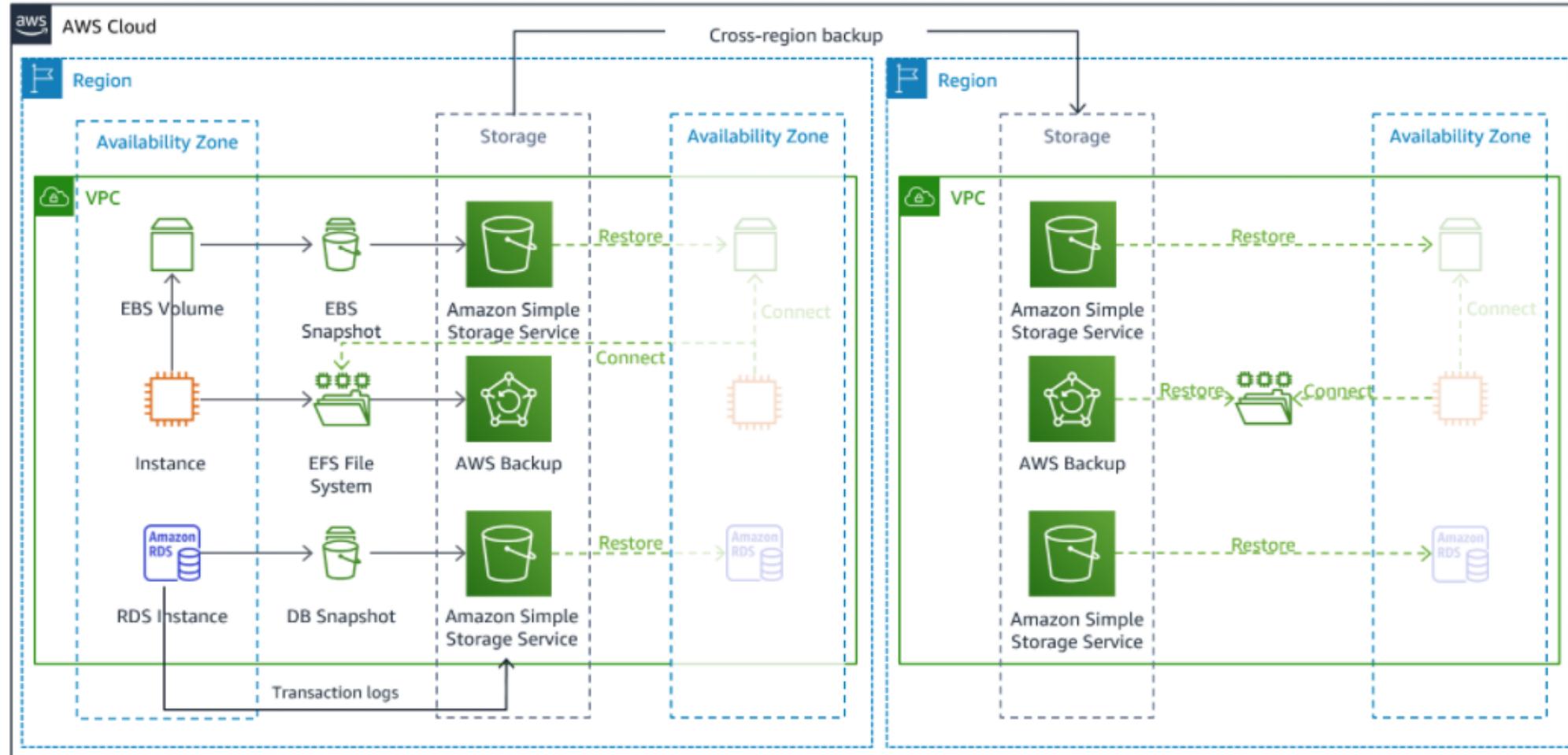
# Disaster Recovery Options and Strategies

# DR Strategies



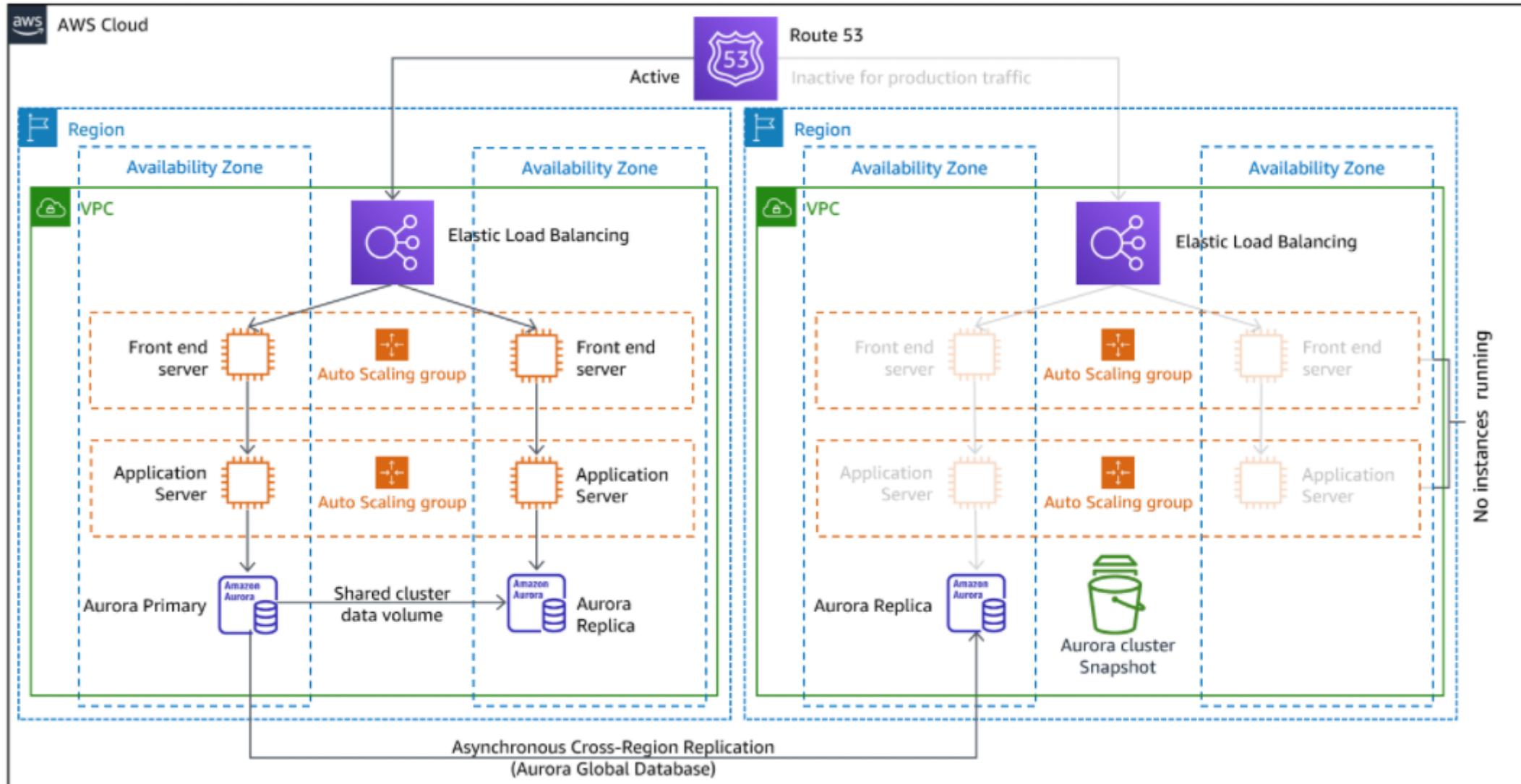
<https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>

# Backup Restore



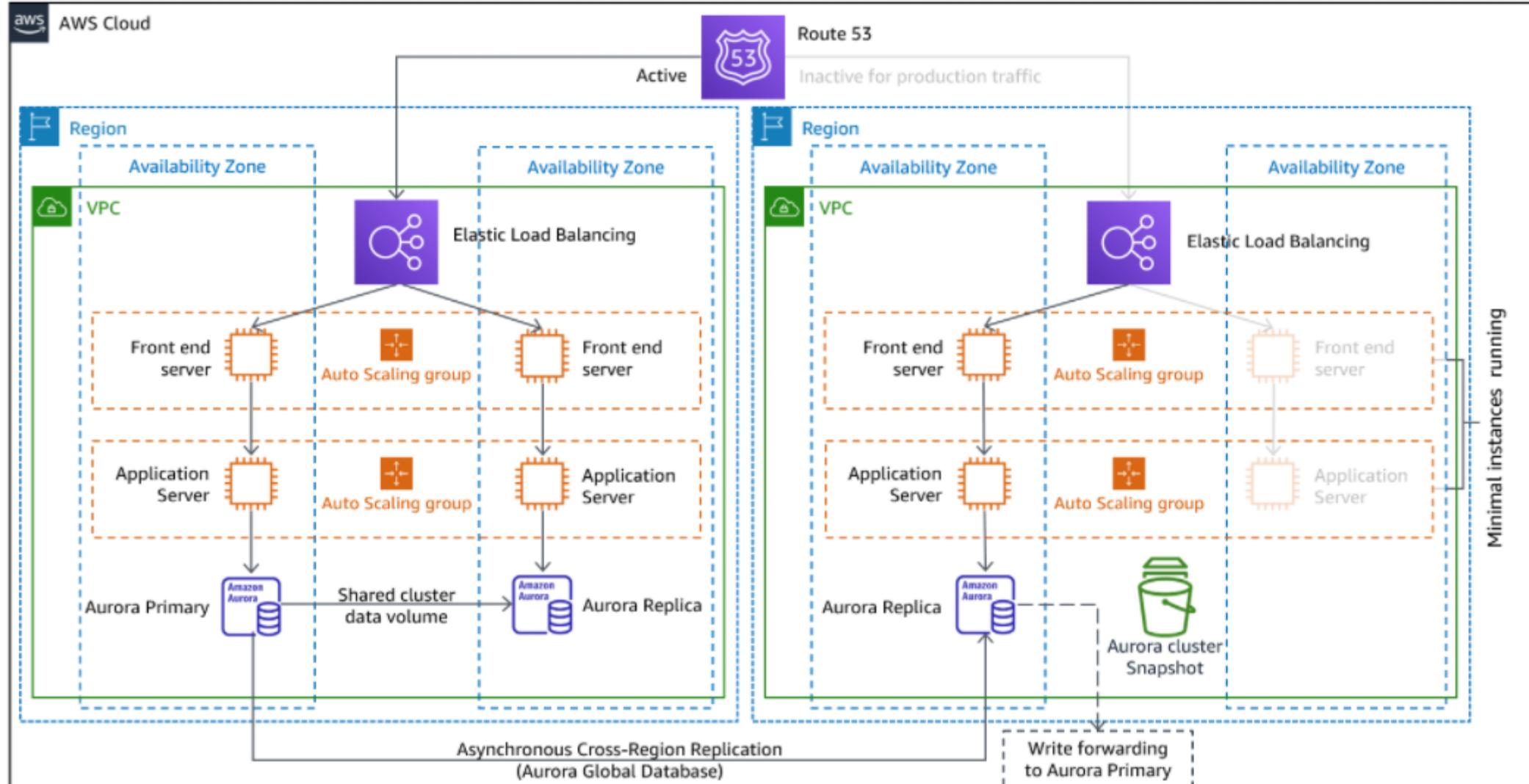
<https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>

# Pilot Light



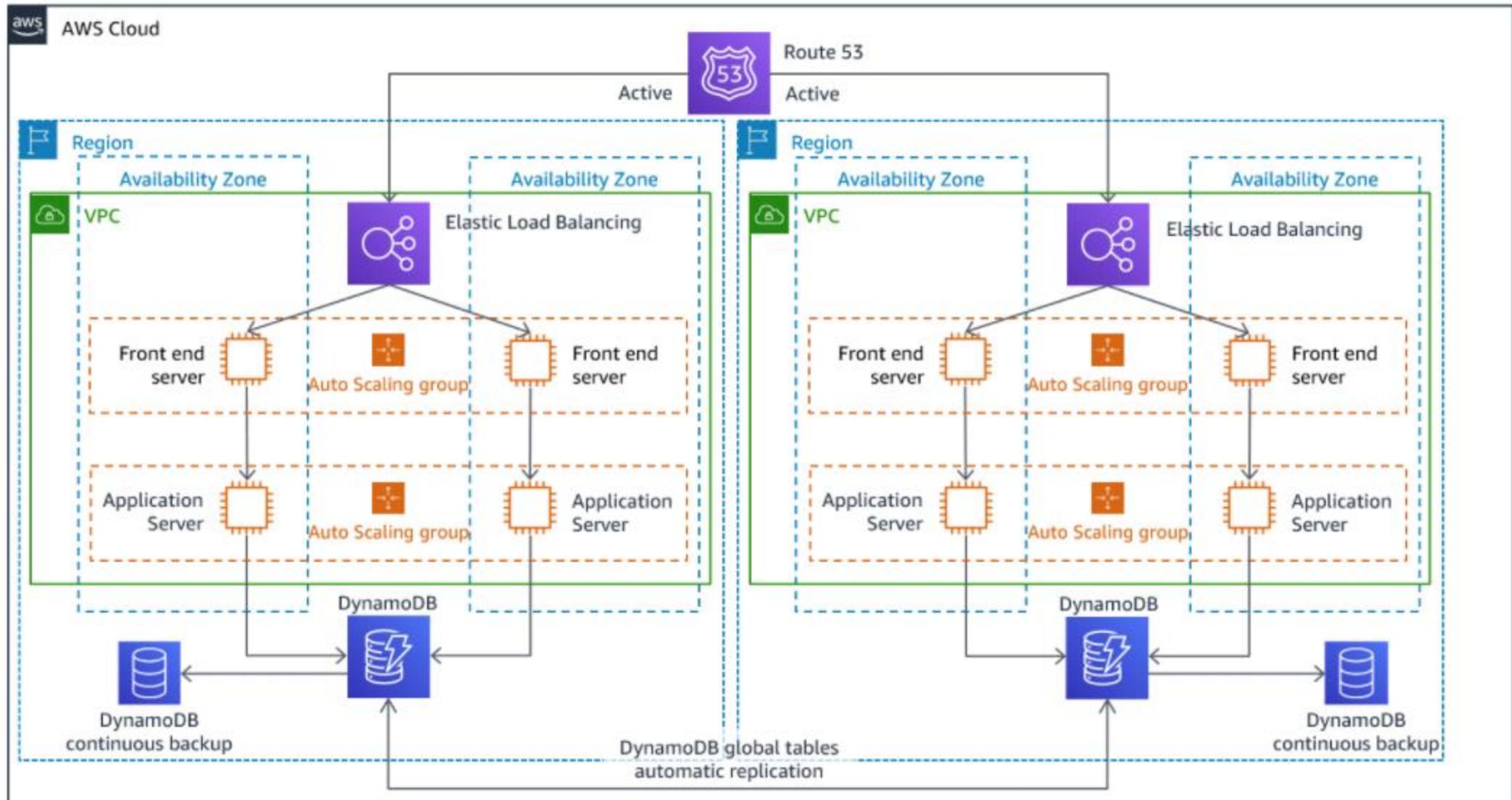
<https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>

# Warm Standby



<https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>

# Multi-site Active/Active



<https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html>

# The Twelve-Factor App

## **I. Codebase**

One codebase tracked in revision control, many deploys

## **II. Dependencies**

Explicitly declare and isolate dependencies

## **III. Config**

Store config in the environment

## **IV. Backing services**

Treat backing services as attached resources

## **V. Build, release, run**

Strictly separate build and run stages

## **VI. Processes**

Execute the app as one or more stateless processes

## **VII. Port binding**

Export services via port binding

## **VIII. Concurrency**

Scale out via the process model

## **IX. Disposability**

Maximize robustness with fast startup and graceful shutdown

## **X. Dev/prod parity**

Keep development, staging, and production as similar as possible

## **XI. Logs**

Treat logs as event streams

## **XII. Admin processes**

Run admin/management tasks as one-off processes

- In 2012, developers at Heroku invented twelve-factor app to help build web apps
- Methodology used to make your app scalable, portable, resilient, faster recovery from disaster, minimize time and cost
- If you know cloud devops best practices, you are following most of these

**I. Codebase**

One codebase tracked in revision control, many deploys

**II. Dependencies**

Explicitly declare and isolate dependencies

**III. Config**

Store config in the environment

**IV. Backing services**

Treat backing services as attached resources

**V. Build, release, run**

Strictly separate build and run stages

**VI. Processes**

Execute the app as one or more stateless processes

**VII. Port binding**

Export services via port binding

**VIII. Concurrency**

Scale out via the process model

**IX. Disposability**

Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**

Keep development, staging, and production as similar as possible

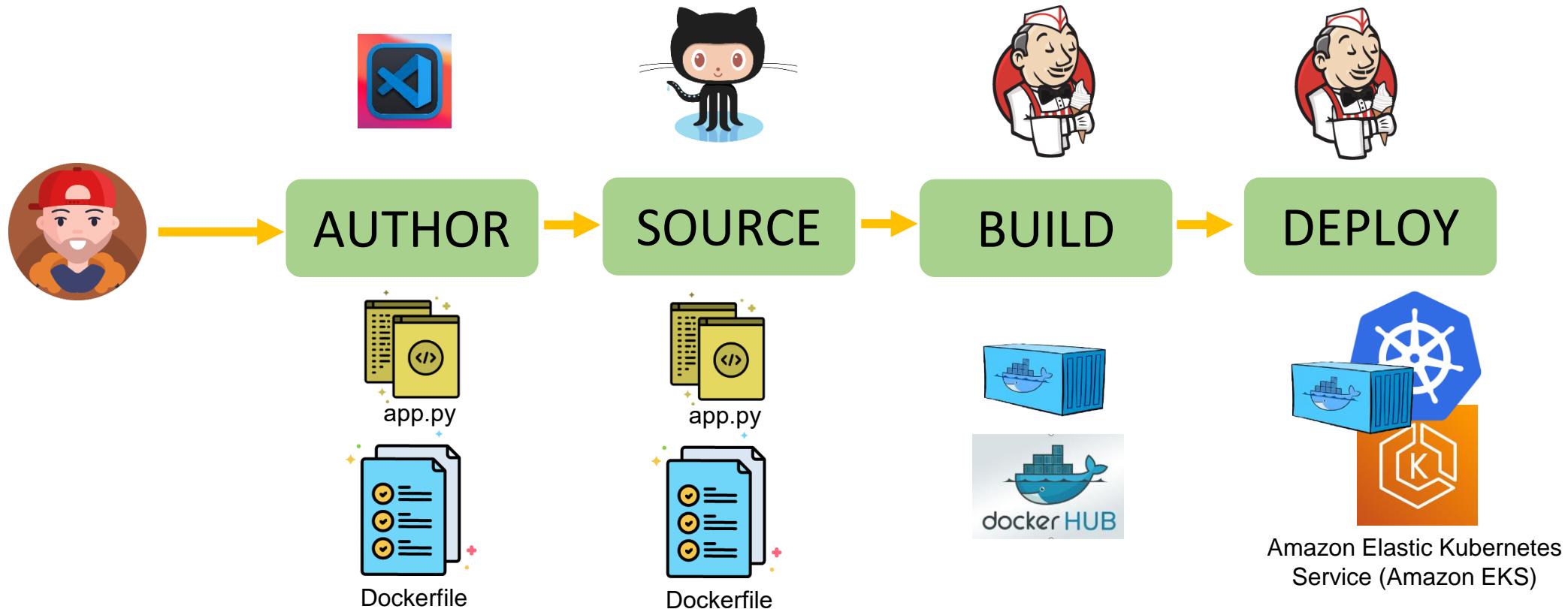
**XI. Logs**

Treat logs as event streams

**XII. Admin processes**

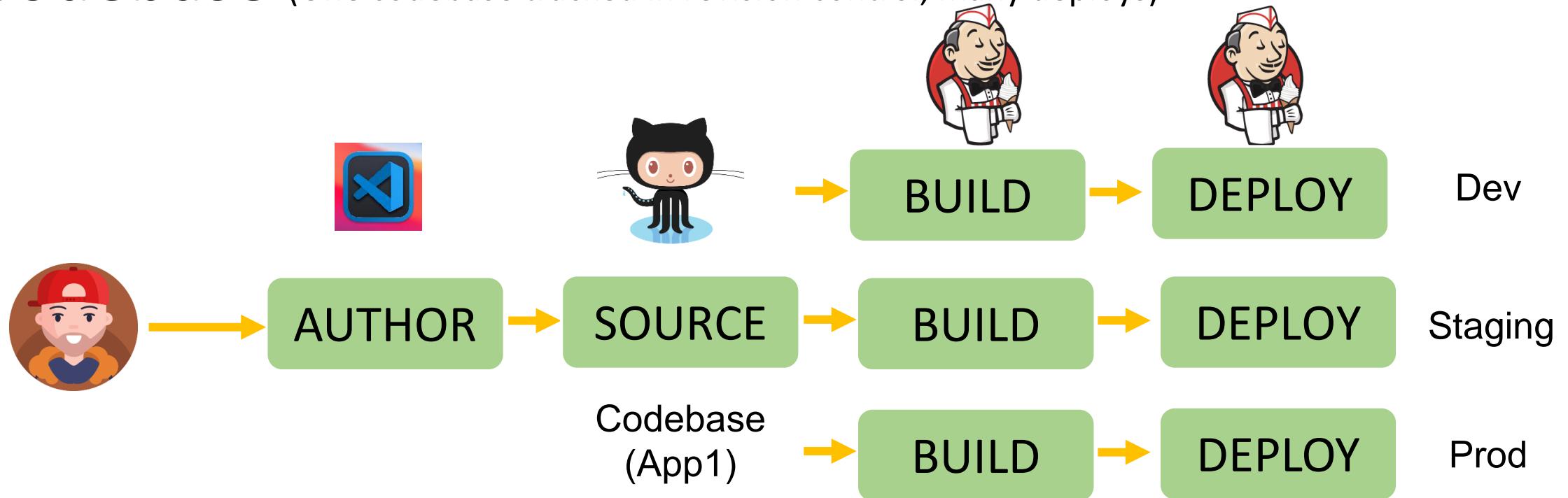
Run admin/management tasks as one-off processes

# I. Codebase (One codebase tracked in revision control, many deploys)



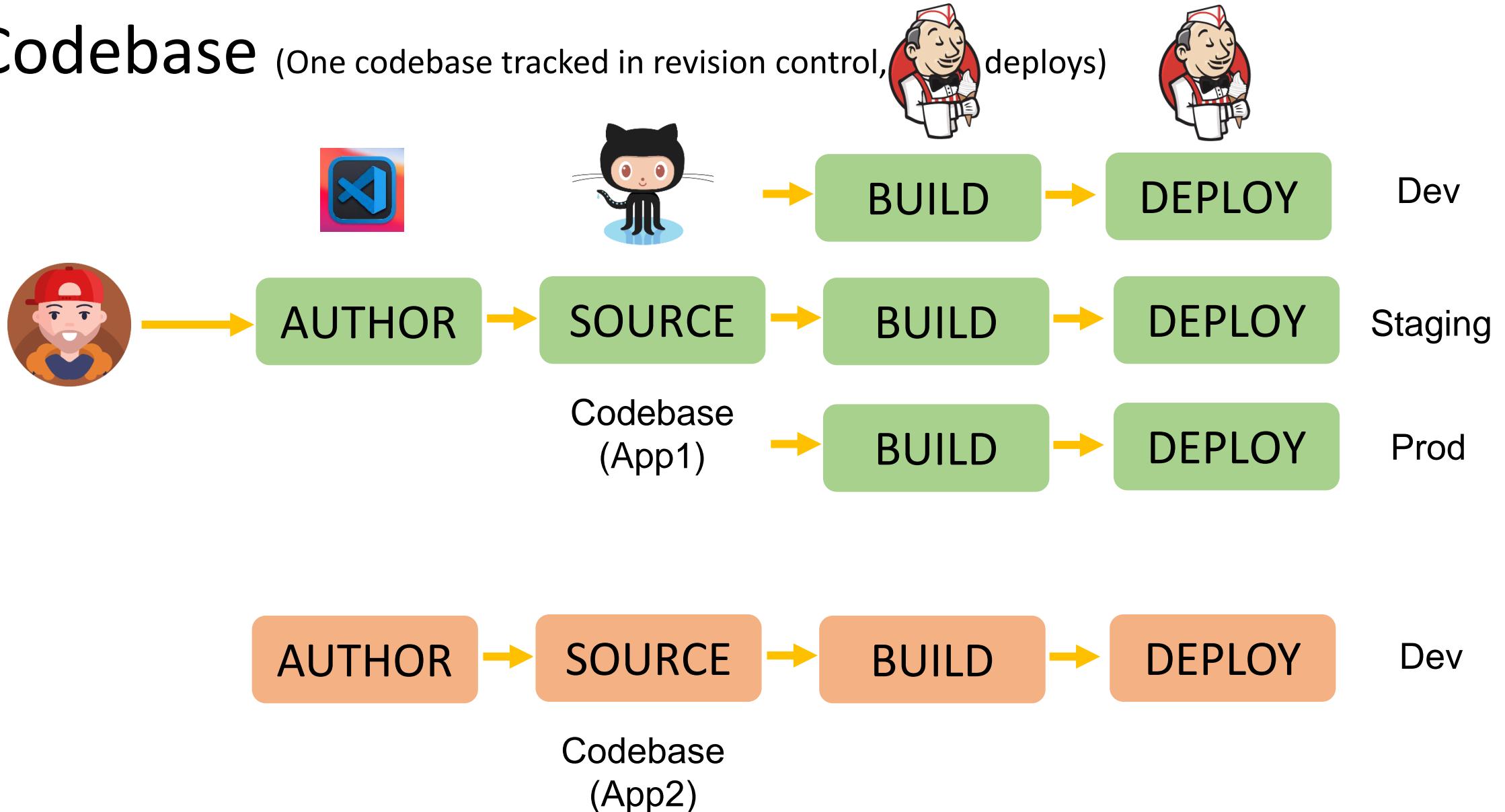
# I. Codebase

(One codebase tracked in revision control, many deploys)



# I. Codebase

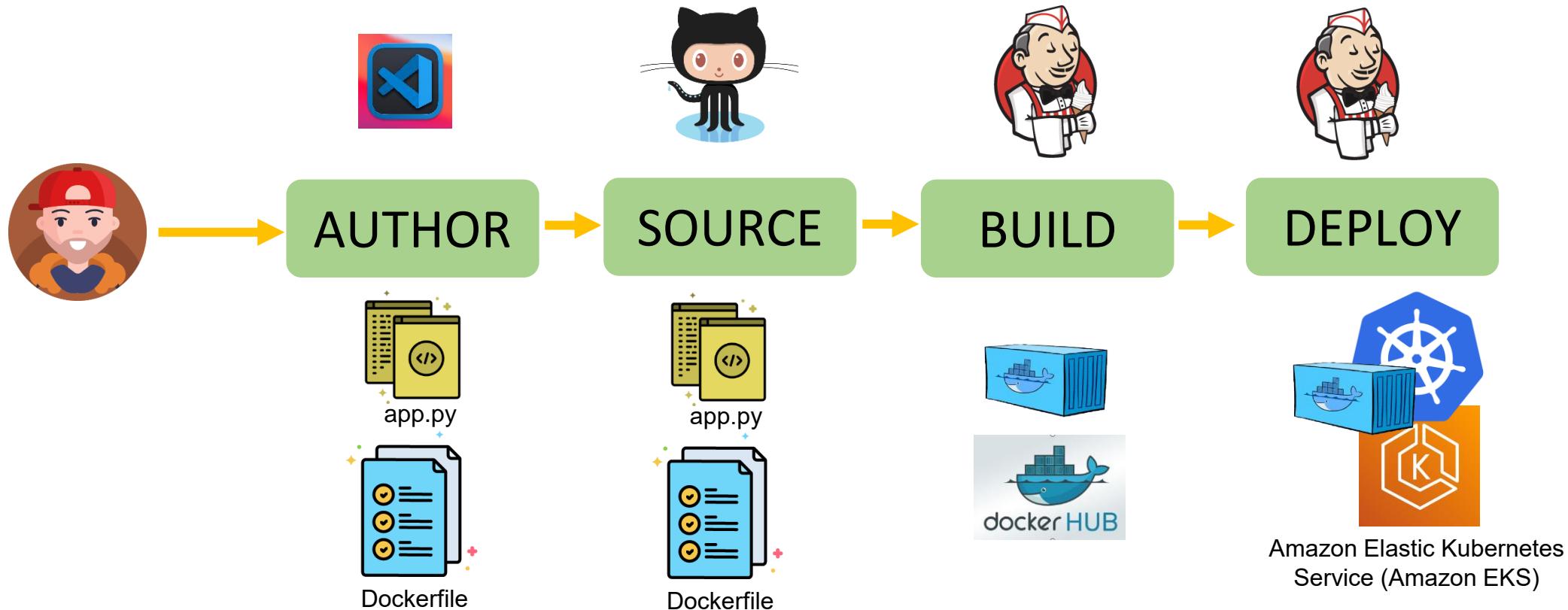
(One codebase tracked in revision control, deploys)



# Objective - Code can be deployed to any environment without any changes

- Remove any environment specific dependencies from the code
- Achieve maximum portability

# II. Dependencies (Explicitly declare and isolate dependencies)



# II. Dependencies (Explicitly declare and isolate dependencies)



AUTHOR



```
server.py > ...
1   from flask import Flask
2   app = Flask(__name__)
3
4   @app.route("/")
5   def hello():
6       return "Hello World!"
7
8   if __name__ == "__main__":
9       app.run(host='0.0.0.0')
```

```
requirements.txt
1   Flask==1.1.1
2
```

Dockerfile

```
# set base image (host OS)
1   FROM python:3.8
2
3
4   # set the working directory in the container
5   WORKDIR /code
6
7
8   # copy the dependencies file to the working directory
9   COPY requirements.txt .
10
11  # install dependencies
12  RUN pip install -r ./requirements.txt
13
14  # copy the content to the working directory
15  COPY server.py .
16
17  # command to run on container start
18  CMD [ "python", "./server.py" ]
```

# II. Dependencies

(Explicitly declare and isolate dependencies)

```
# server.py > ...
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route("/")
5  def hello():
6      return "Hello World!"
7
8  if __name__ == "__main__":
9      app.run(host='0.0.0.0')
```

Dev

```
≡ requirements.txt
1  Flask==1.1.1
2  |
```

Python Virtual  
environment  
for Dev

Staging

```
≡ requirements.txt
1  Flask==2.1.1
2  |
```

Python Virtual  
environment  
for Staging

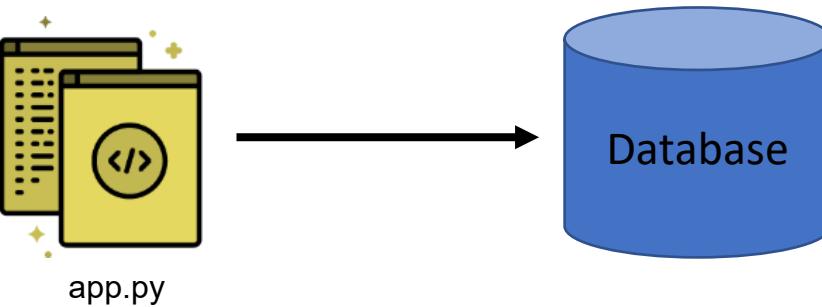
# II. Dependencies (Explicitly declare and isolate dependencies)

The screenshot shows a code editor interface with a dark theme. On the left, there's a file tree with a file named 'server.py' containing the following code:

```
1  from flask import Flask
2  app = Flask(__name__)
3
4  @app.route("/")
5  def hello():
6      return "Hello World!"
7
8  if __name__ == "__main__":
9      app.run(host='0.0.0.0')
```

A tooltip is displayed over the word 'flask' in line 1, asking 'Select an environment type'. It lists two options: 'Venv' (Creates a '.venv' virtual environment in the current workspace) and 'Conda' (Creates a '.conda' Conda environment in the current workspace). In the bottom right corner of the editor window, there is a blue button labeled 'Create Environment...'. The top right corner of the slide has icons for presentation controls.

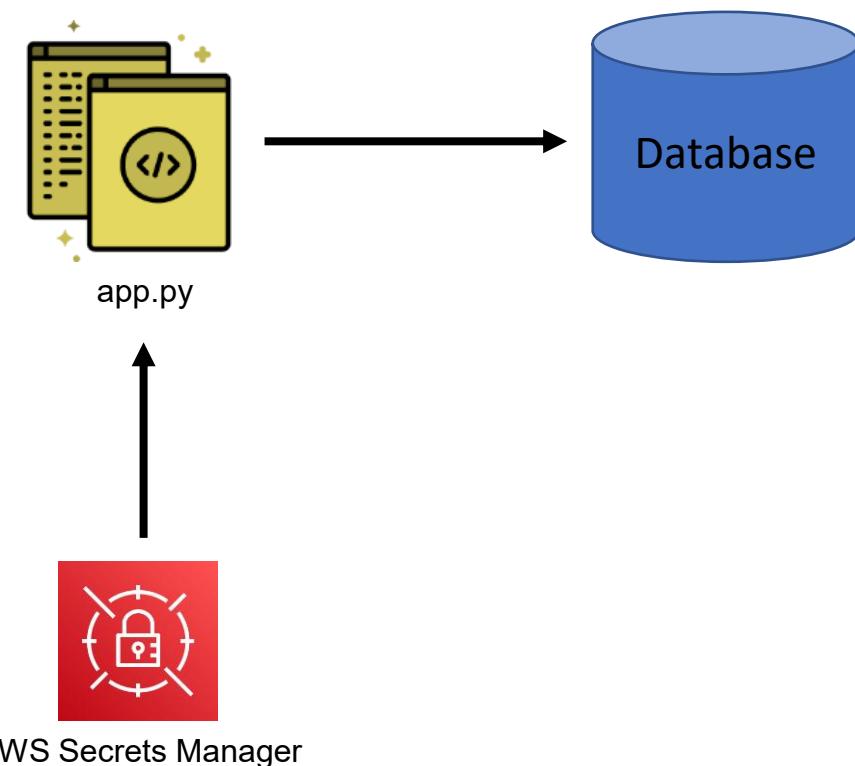
### III. Config (Store config in the environment)



```
1 import pymysql
2
3 # Replace the following variables with your RDS instance credentials
4 host = "your_rds_host_endpoint"
5 port = 3306
6 dbname = "your_database_name"
7 user = "your_username"
8 password = "your_password"
9
10 # Connect to the RDS instance
11 try:
12     connection = pymysql.connect(
13         host=host,
14         port=port,
15         db=dbname,
16         user=user,
17         password=password
18     )
19     print("Connected to the RDS instance")
20
21 except pymysql.Error as error:
22     print(f"Error connecting to the RDS instance: {error}")
23
24 # Don't forget to close the connection when you're done
25 connection.close()
```

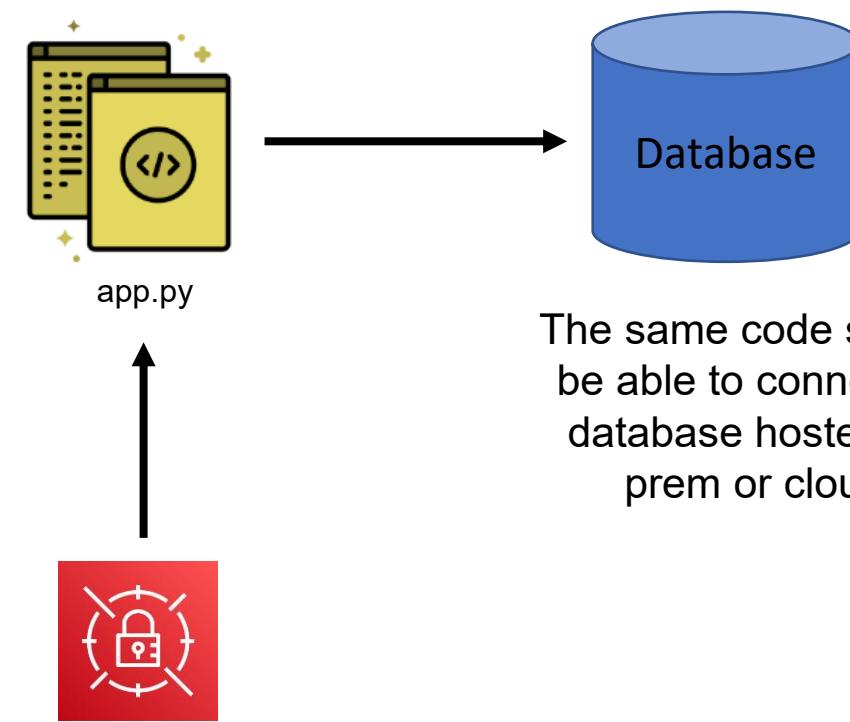
# III. Config (Store config in the environment)

```
9  session = boto3.session.Session()
10 client = session.client(
11     service_name="secretsmanager",
12     region_name=region_name
13 )
14
15 response = client.get_secret_value(
16     SecretId=secret_name
17 )
18
19 secret_string = response["SecretString"]
20 secrets = json.loads(secret_string)
21
22 # Extract database credentials from the secrets
23 host = secrets["host"]
24 port = secrets["port"]
25 dbname = secrets["dbname"]
26 user = secrets["username"]
27 password = secrets["password"]
28
29 # Connect to the RDS instance
30 try:
31     connection = pymysql.connect(
32         host=host,
33         port=port,
34         db=dbname,
35         user=user,
36         password=password
37     )
```



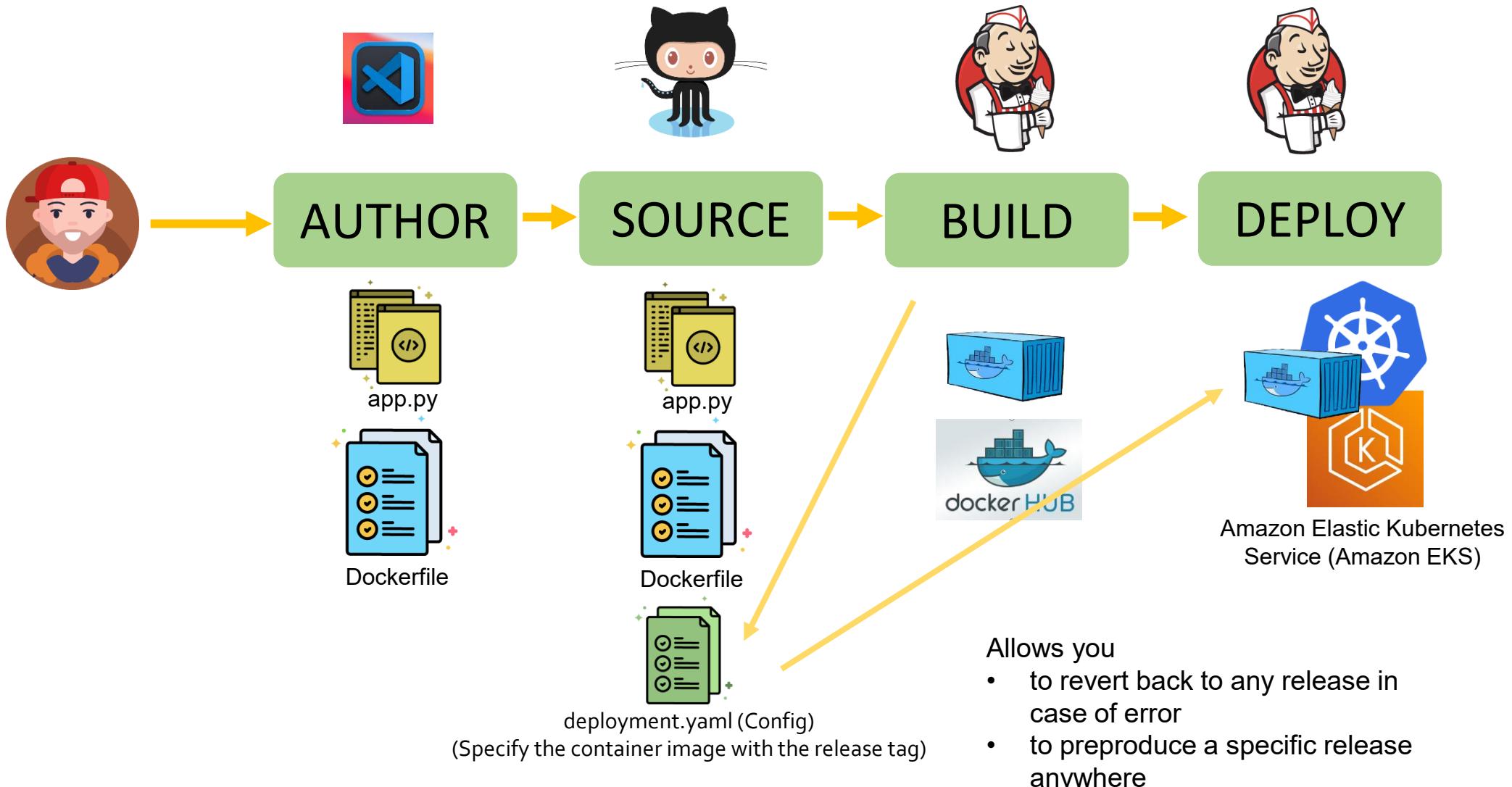
# IV. Backing services (Treat backing services as attached resource)

```
9 session = boto3.session.Session()
10 client = session.client(
11     service_name="secretsmanager",
12     region_name=region_name
13 )
14
15 response = client.get_secret_value(
16     SecretId=secret_name
17 )
18
19 secret_string = response["SecretString"]
20 secrets = json.loads(secret_string)
21
22 # Extract database credentials from the secrets
23 host = secrets["host"]
24 port = secrets["port"]
25 dbname = secrets["dbname"]
26 user = secrets["username"]
27 password = secrets["password"]
28
29 # Connect to the RDS instance
30 try:
31     connection = pymysql.connect(
32         host=host,
33         port=port,
34         db=dbname,
35         user=user,
36         password=password
37     )
```



The same code should  
be able to connect to  
database hosted on  
prem or cloud

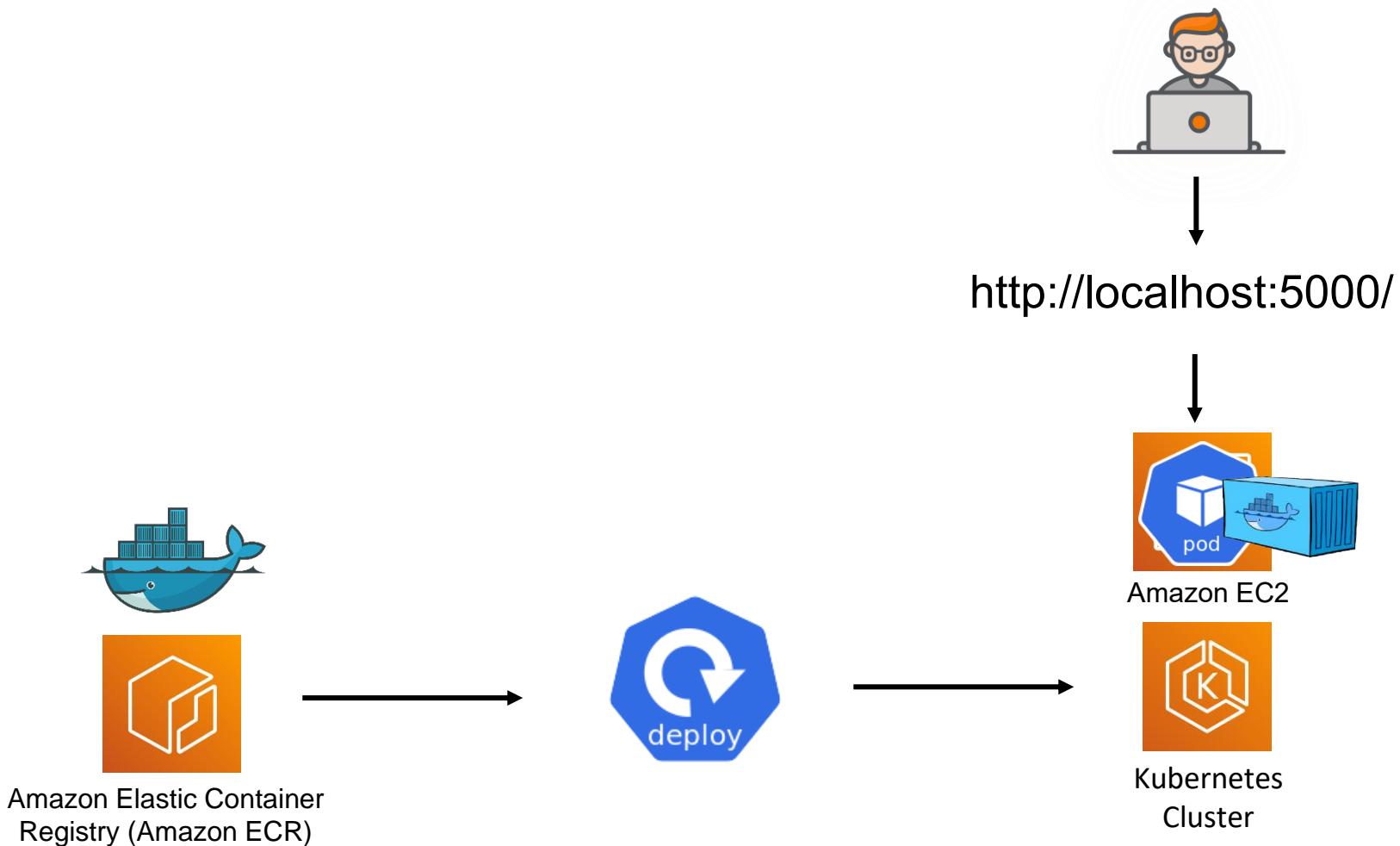
# V. Build, release, run (Strictly separate build and run stages)



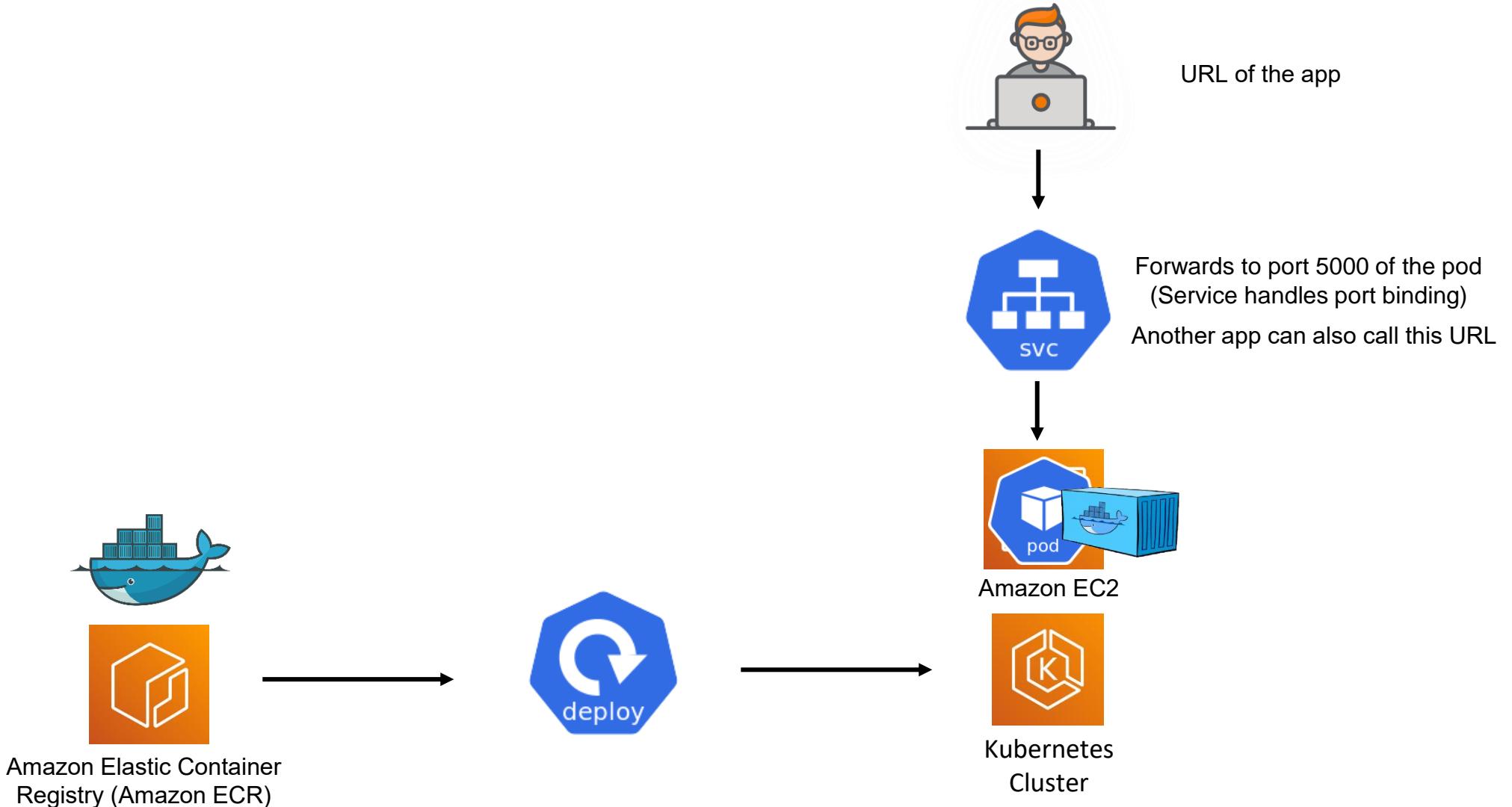
# VI. Processes (Execute the app as one or more stateless processes)

- Process is minimum deployable unit of your app
  - One microservice backend code is one process
  - Multiple microservices working together for a functionality will be multiple processes
- Processes should be stateless and share-nothing
  - Any data that needs to persist should be stored in a stateful backing service like database
- Sticky sessions are in violation of twelve factor app

# VII. Port binding (Execute services via port binding)



# VII. Port binding (Execute services via port binding)



# VIII. Concurrency (Scale out via process model)

- Till now we ensured all the processes (microservice/minimum deployable unit of the app) are portable
  - Moved dependencies/config/database-location out of code
  - Stateless
  - *We'll move logging dependencies out of the code as well (Number XI)*
- Scale the processes using horizontal scaling

# IX. Disposability

(Maximize robustness with fast startup and graceful shutdown)

- Processes are disposable, meaning they can be started or stopped at a moment's notice
- Graceful shutdown
  - Stop accepting new requests
  - Allow current requests to finish
  - Exit
- In case graceful shutdown not possible
  - Return jobs to the queue

SIGTERM  
SIGKILL

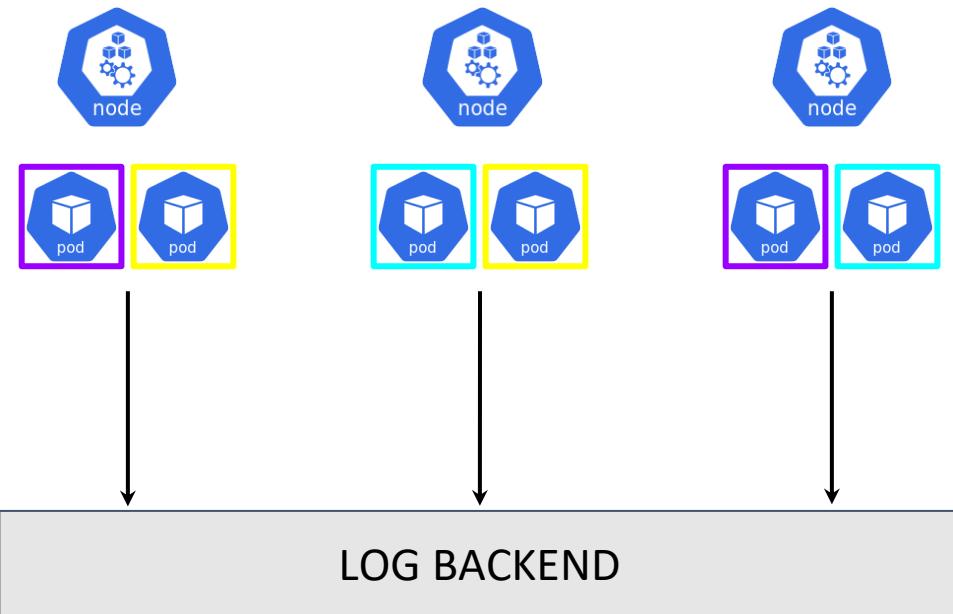


Amazon EC2

# X. Dev/prod parity (Keep development, staging, and production as similar as possible)

- Pre DevOps/Containers era – substantial gap between dev and prod
  - Time gap – longer time to go to prod
  - Personnel gap – separation between dev and ops
  - Tools gap – different tools used in dev and prod
- Utilize DevOps practices
  - Continuous deployment
  - Same tools in dev and prod
    - Containerization solves this
    - Use same backing service in dev and prod (Cost efficient cloud services solve this)

# XI. Logs (Treat logs as event streams)



Code sending logs directly to Splunk

```
hec_url = "https://your_splunk_instance:8088/services/collector"
hec_token = "your_hec_token"

# Define your log event
event = {
    "time": "your_event_timestamp",
    "sourcetype": "your_sourcetype",
    "event": {
        "message": "This is a log message",
        "severity": "info",
        "custom_field": "custom_value"
    }
}

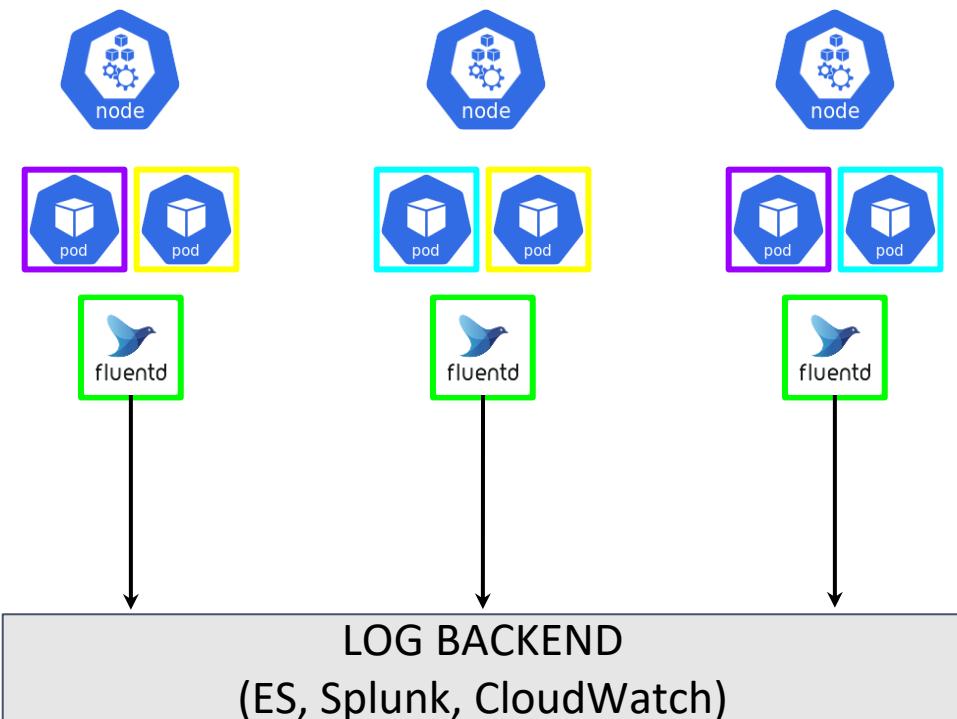
# Set headers for the POST request
headers = {
    "Content-Type": "application/json",
    "Authorization": f"Splunk {hec_token}"
}

# Send the log event to Splunk
try:
    response = requests.post(hec_url, headers=headers, data=json.dumps(event), verify=False)
```

## Challenges

- Code need to be changed between environments (dev, stage, prod)
- Code need to be changed for changing logging backend
- Tightly coupled log integration
  - Both compute and logging platform need to scale at same rate

# XI. Logs (Treat logs as event streams)



= Logging agent running as daemon, reading logs from stdout and sending to logging backend



Writing logs to stdout

```
import json

# Define your log event
event = {
    "time": "your_event_timestamp",
    "sourcetype": "your_sourcetype",
    "event": {
        "message": "This is a log message",
        "severity": "info",
        "custom_field": "custom_value"
    }
}

# Convert the event to a JSON string
event_json = json.dumps(event)

# Write the log event to stdout
print(event_json)
```

## XII. Admin processes (Run admin/management tasks as one-off processes)

- At times, one off admin processes needed to be run
  - Update database values
  - Run one time scripts
- Admin processes should:
  - Run in the identical environment of the app
  - Use same codebase
  - Ship with the application
    - Create separate run, build, release with admin code
  - Should produce logs

- Used to make your app scalable, resilient, faster recovery from disaster, faster time to market
- If you know cloud best practices, you are following most of these
- Do I need to remember all of these for interview?
  - Are all these still relevant?

**I. Codebase**

One codebase tracked in revision control, many deploys

**II. Dependencies**

Explicitly declare and isolate dependencies

**III. Config**

Store config in the environment

**IV. Backing services**

Treat backing services as attached resources

**V. Build, release, run**

Strictly separate build and run stages

**VI. Processes**

Execute the app as one or more stateless processes

**VII. Port binding**

Export services via port binding

**VIII. Concurrency**

Scale out via the process model

**IX. Disposability**

Maximize robustness with fast startup and graceful shutdown

**X. Dev/prod parity**

Keep development, staging, and production as similar as possible

**XI. Logs**

Treat logs as event streams

**XII. Admin processes**

Run admin/management tasks as one-off processes

# 12 Factor App Interview Q/A

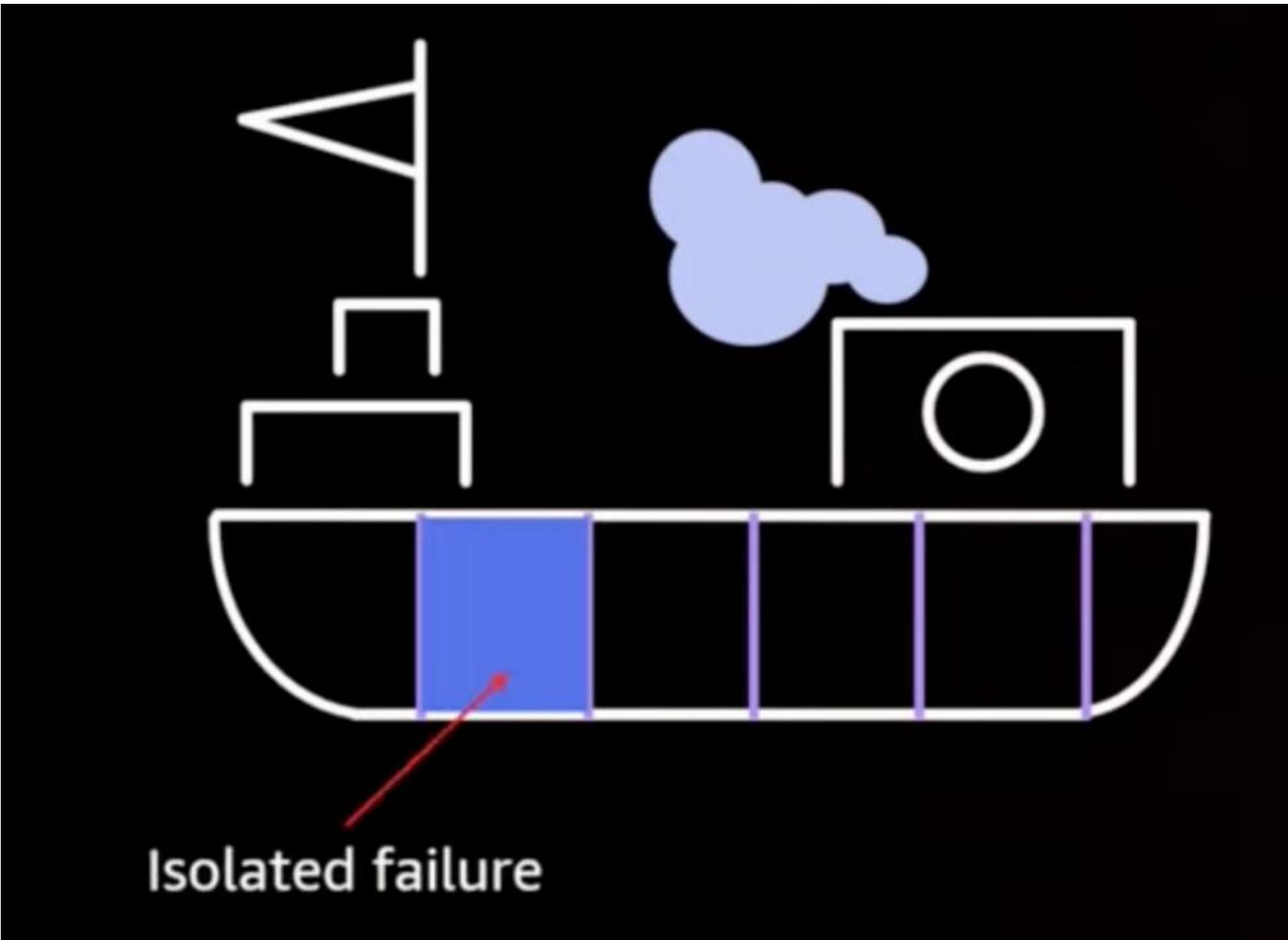
- You do NOT need to remember
  - the numbers-factors mapping
  - the cryptic taglines
  - ALL the factors
- Why is twelve factor app important?
- Explain how twelve factor app makes your code portable?
- Explain three twelve factor app principles?
- Give me an example where you couldn't follow twelve factor app principle, and why?

## VIII. Concurrency

Scale out via the process model

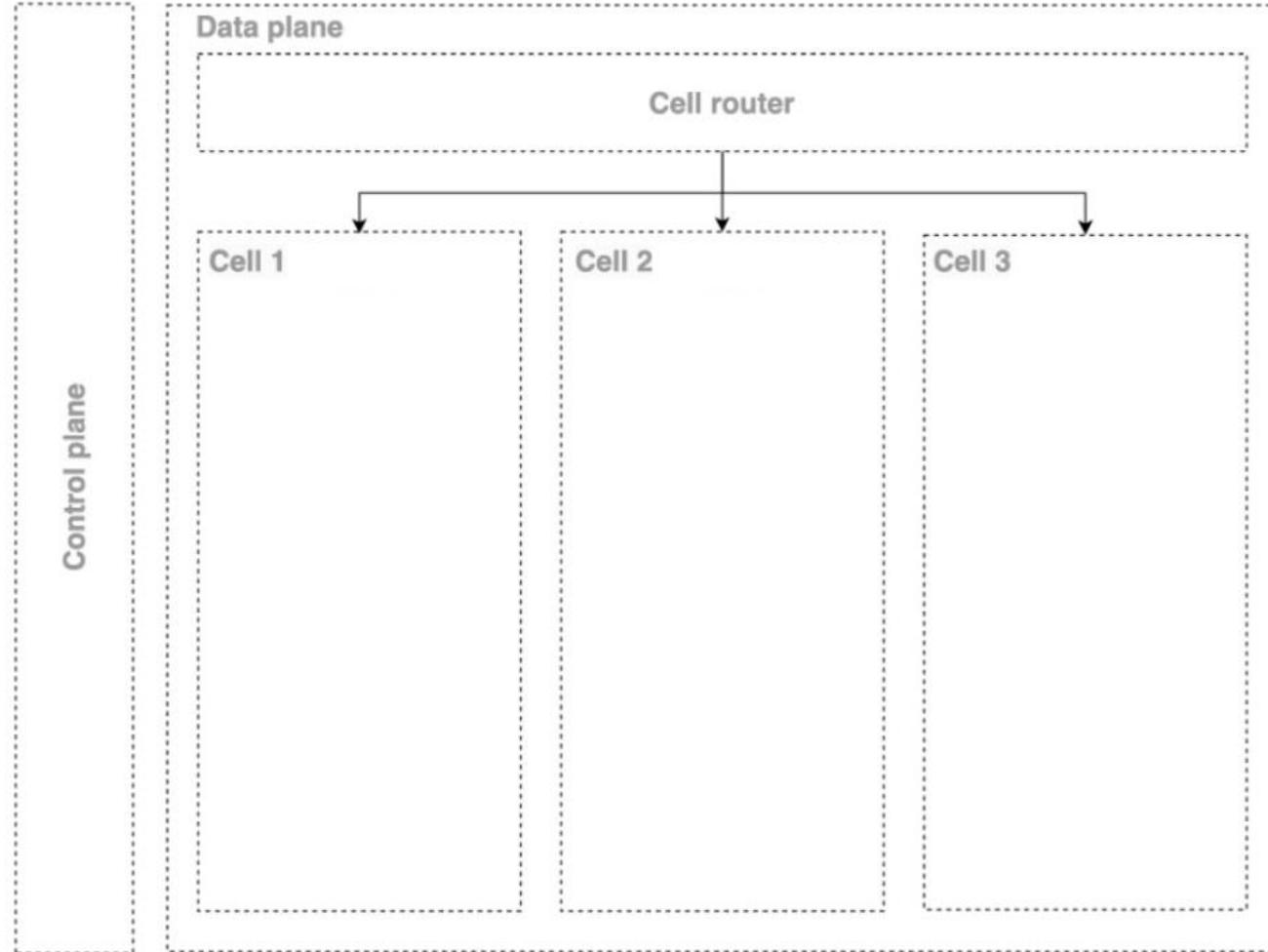
# Cell Based Architecture

# Cells



# Cell Based Architecture

AWS Cloud



- **Cell Router** - *thinnest possible layer*, with the responsibility of routing requests to the right cell, and only that
- **Cell** — A complete workload, with everything needed to operate independently
- **Control plane** — Responsible for administration tasks, such as provisioning cells, de-provisioning cells, and migrating cell customers.
- Main Goal - failure containment and reduced blast radius

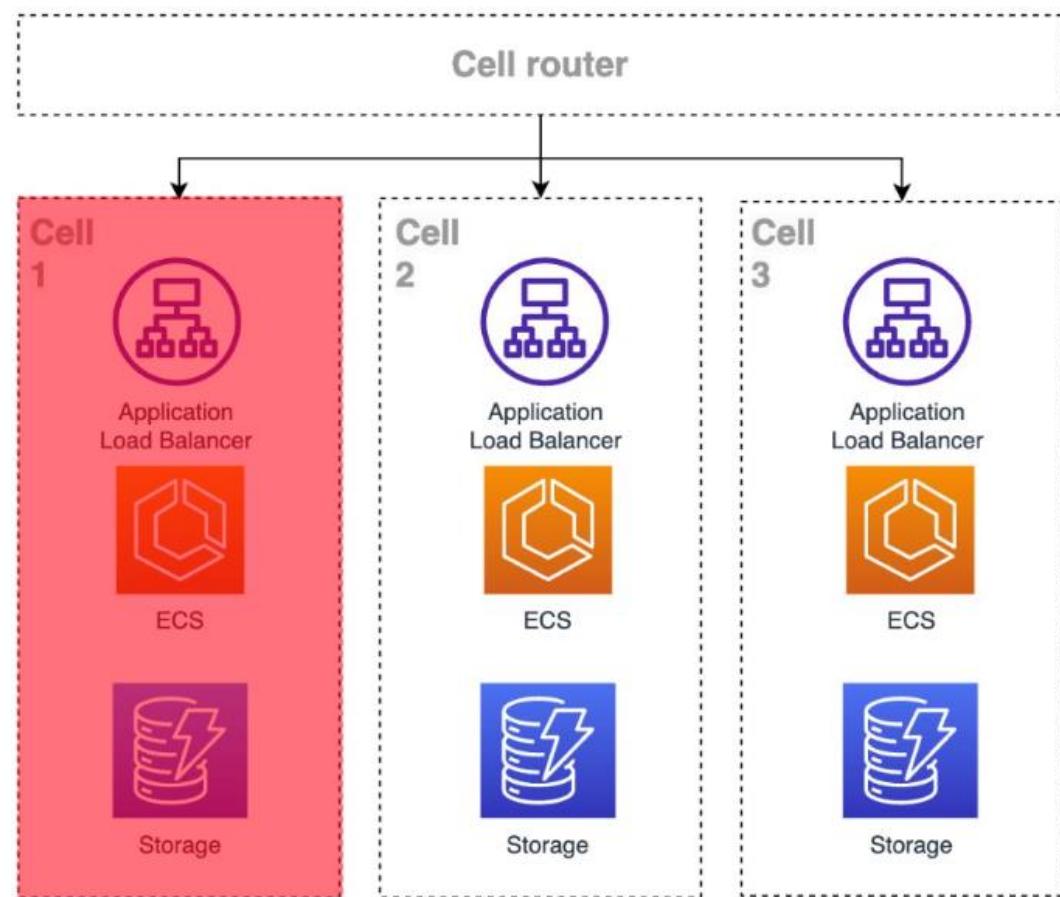
# Cell Based Architecture

## No Cell Architecture



100% of the customers unavailable

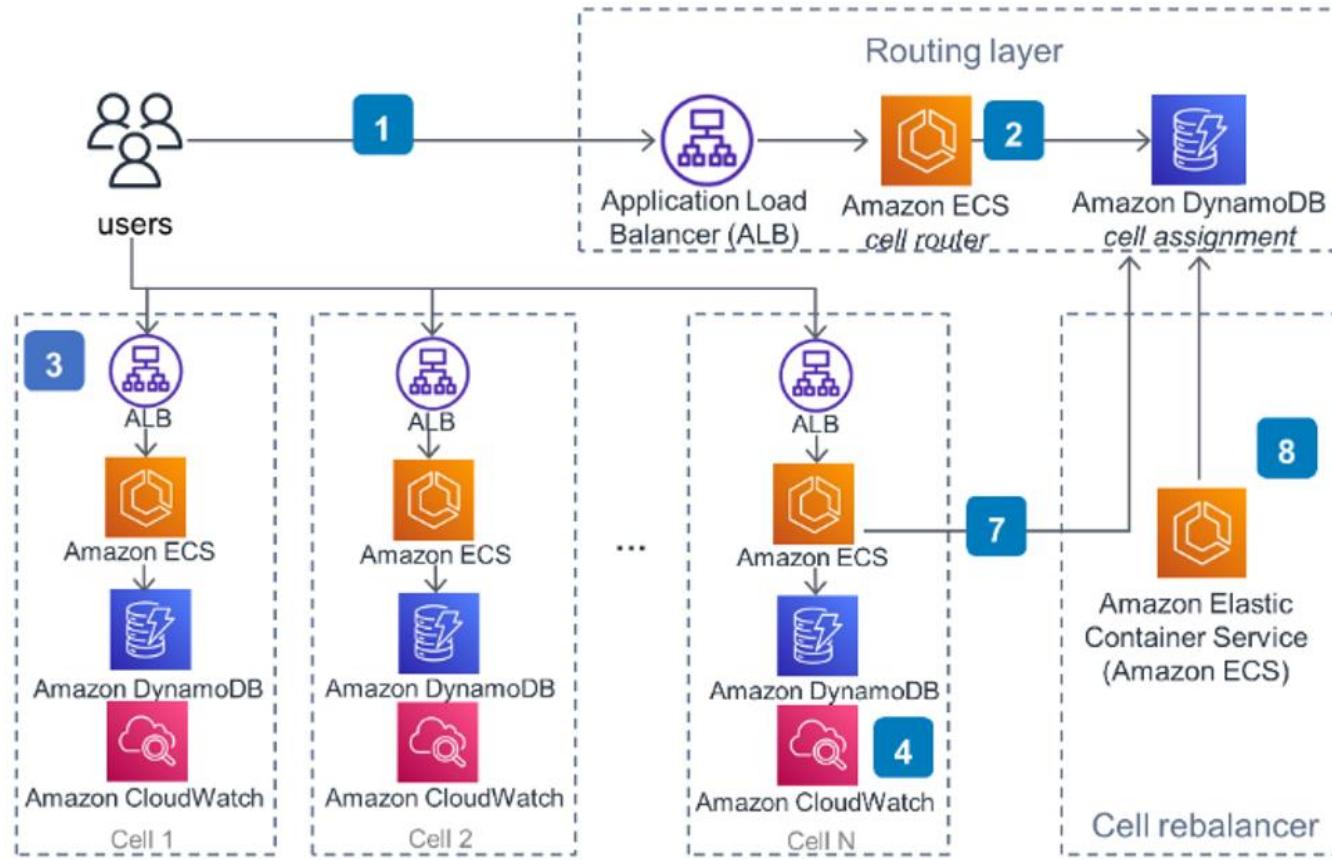
## Cell Based Architecture



1/3 of the customers unavailable

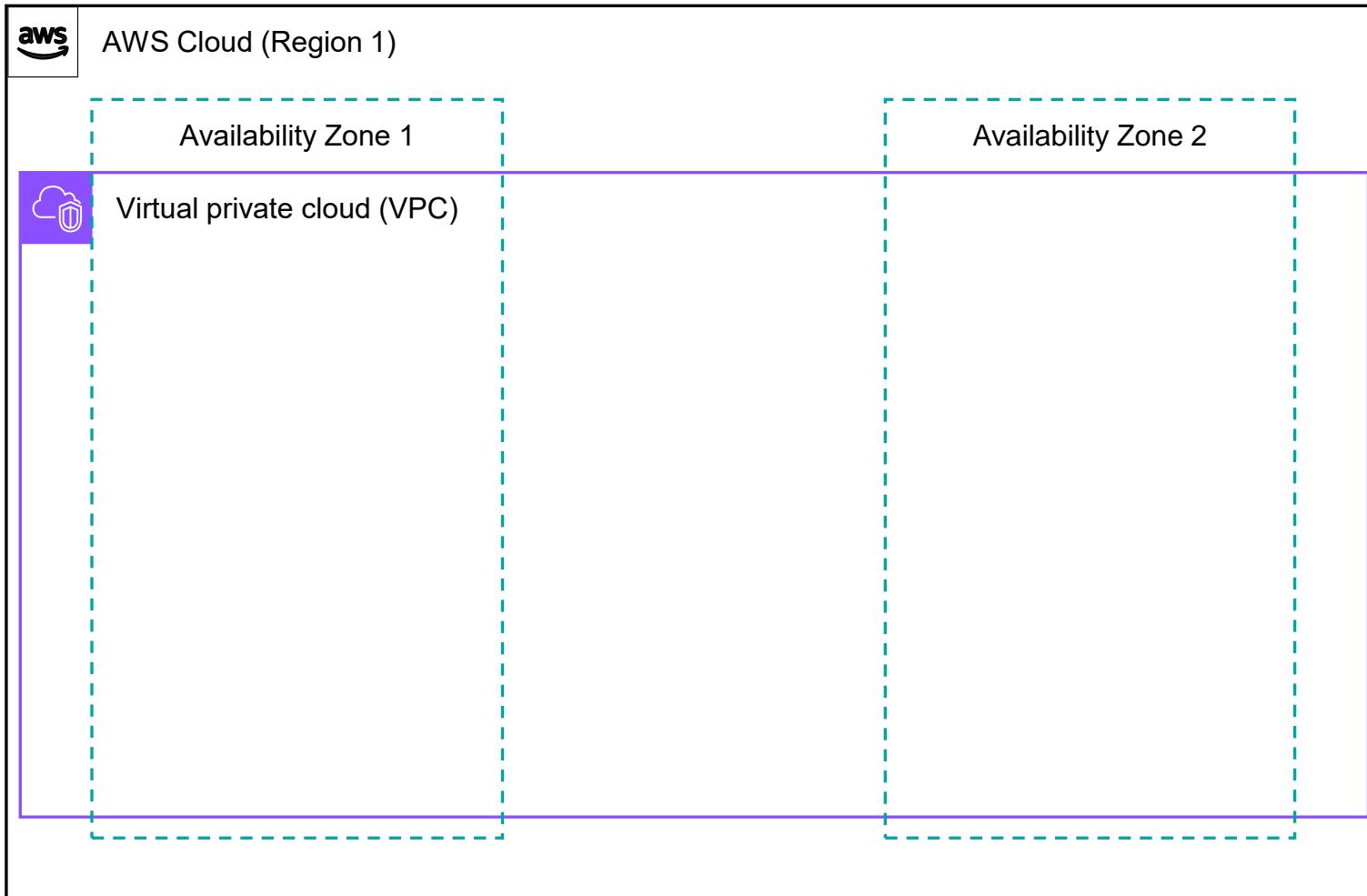
# Cell Based Architecture - Example

# Cell Based Architecture - Implementation

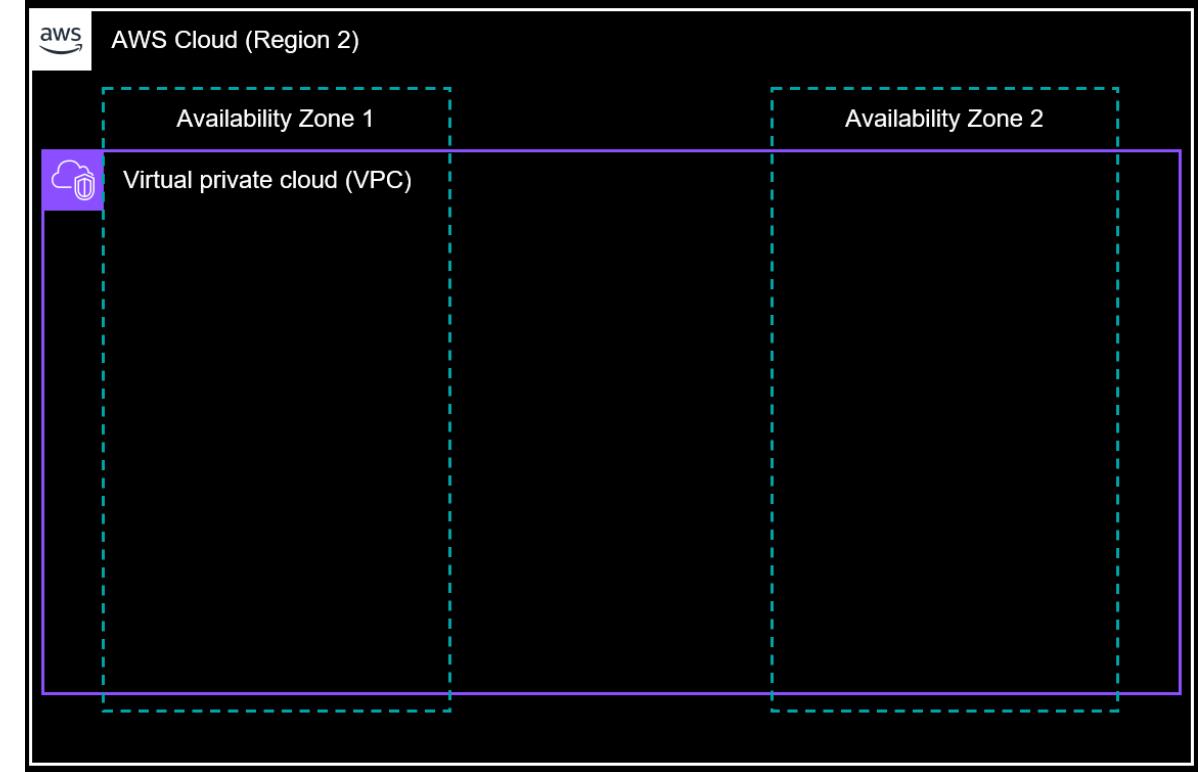
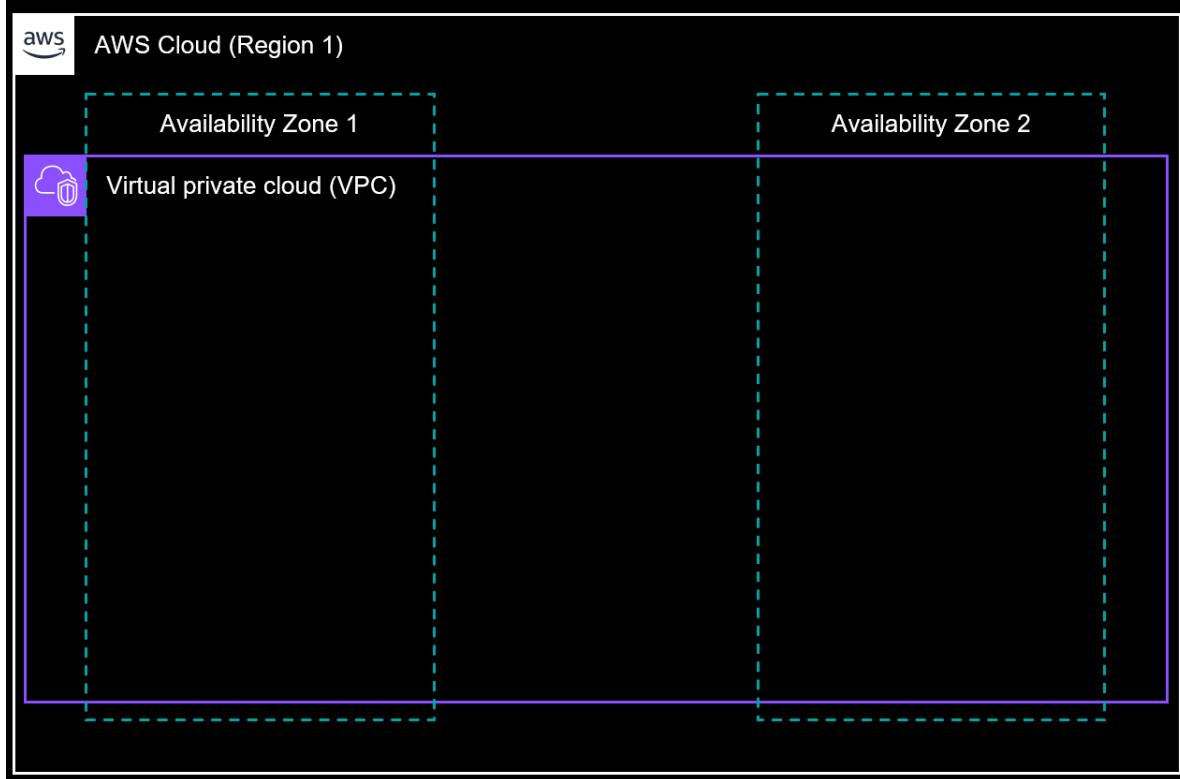


<https://aws.amazon.com/solutions/guidance/cell-based-architecture-on-aws/>

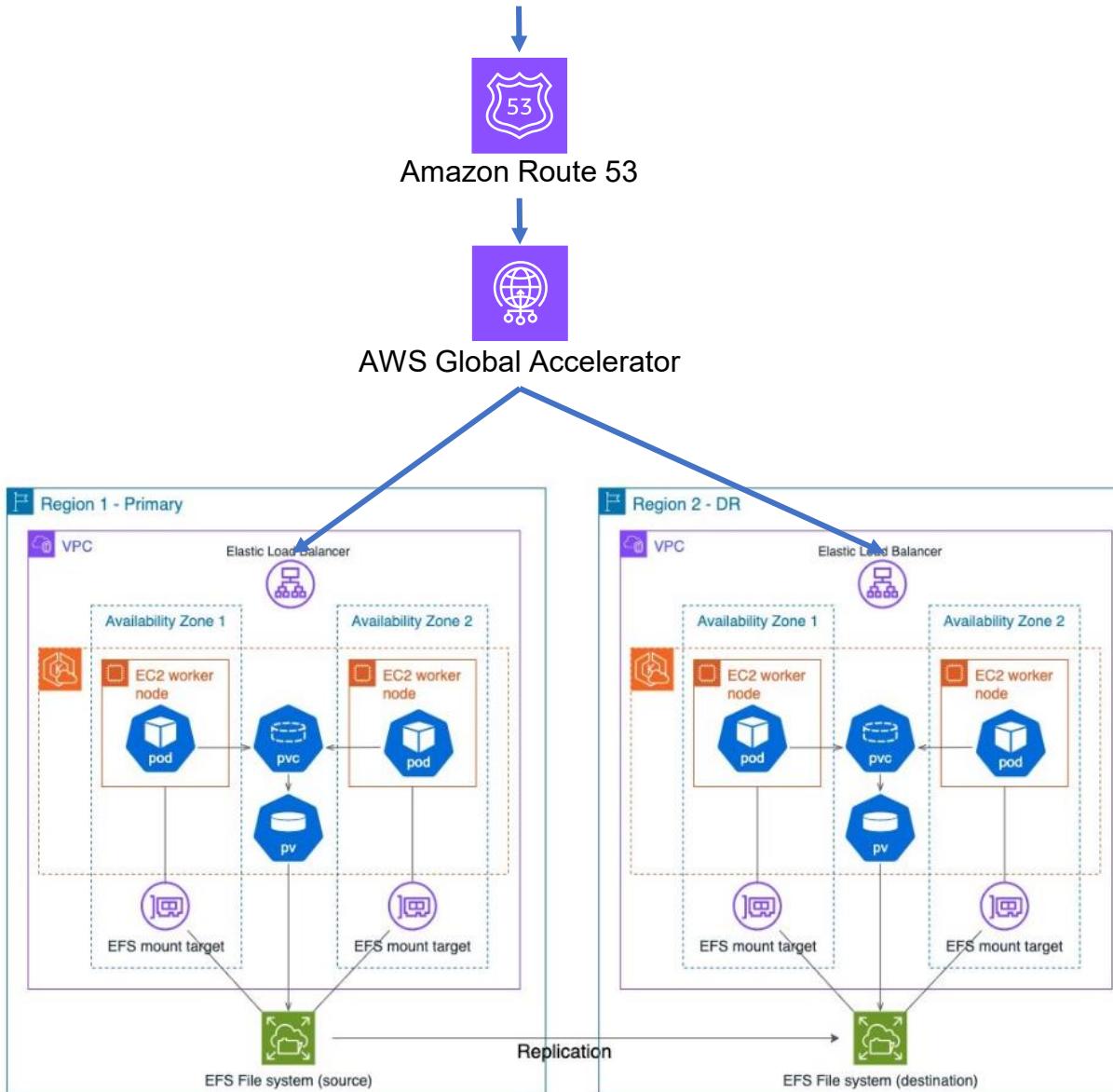
# Cell Based Architecture – What is a cell?



# Cell Based Architecture – What is a cell?

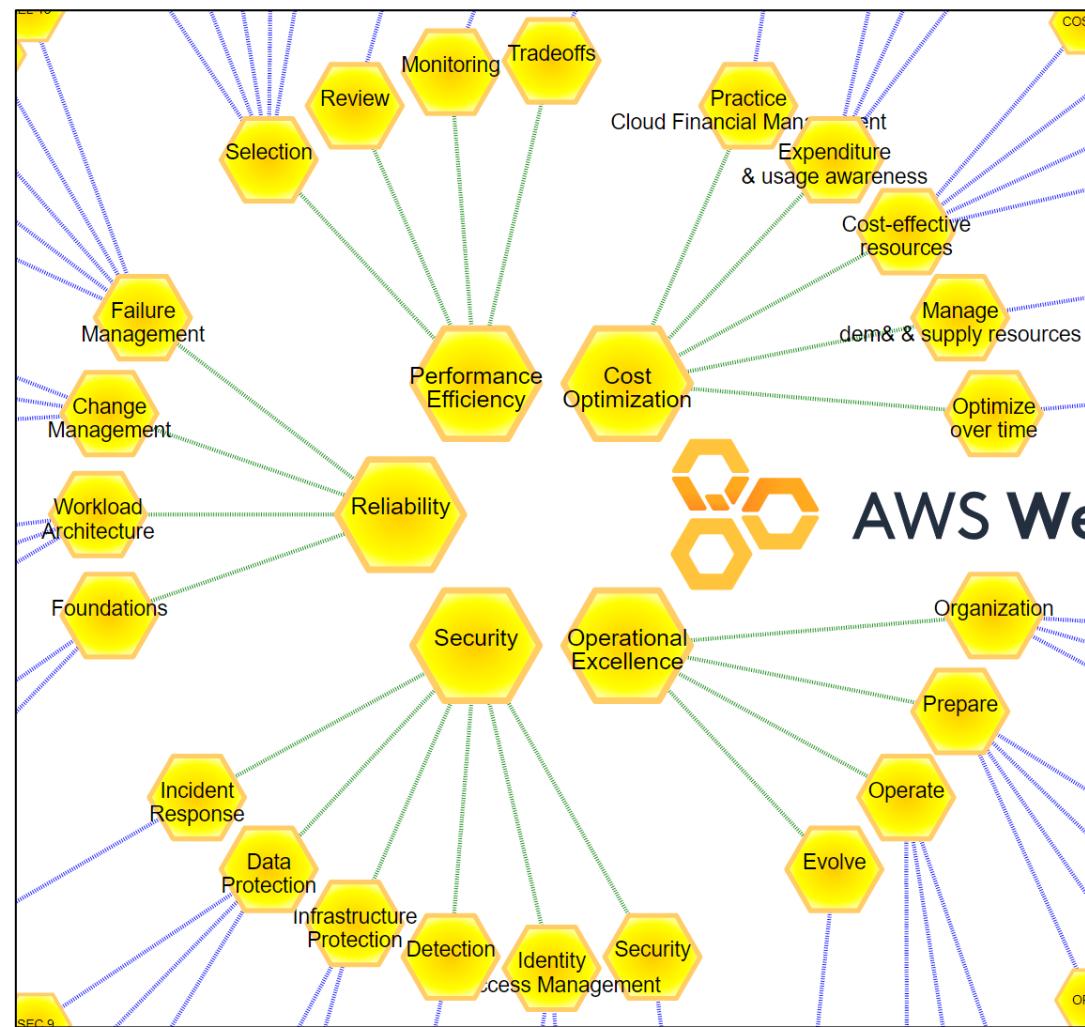


# Cell Based Architecture – Popular Implementation



# SECTION 2 – REUSABLE PARTS OF SYSTEM DESIGN

# 5 Pillars of AWS Well Architected Framework (Not just for AWS)



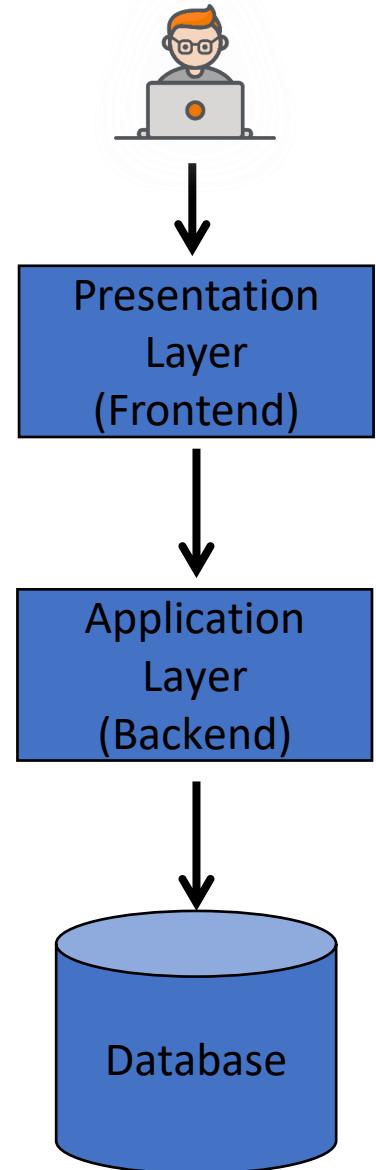
<https://wa.aws.amazon.com/map.html>

# Impact on System Design

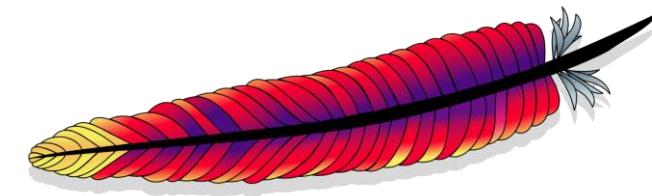
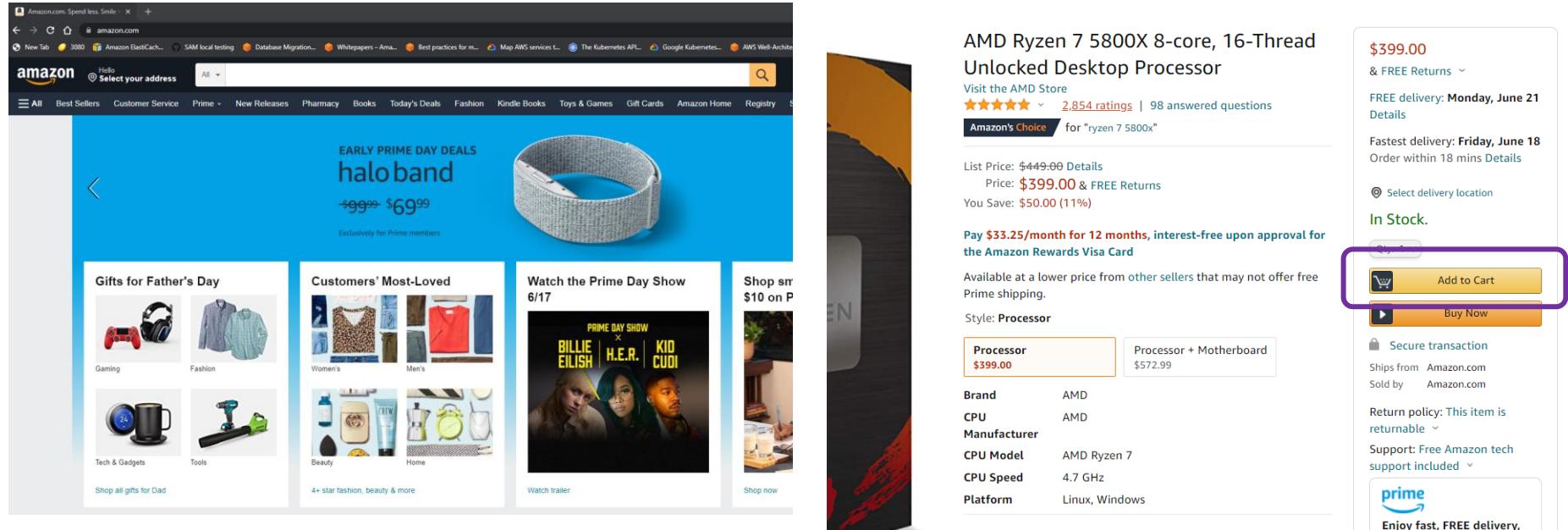
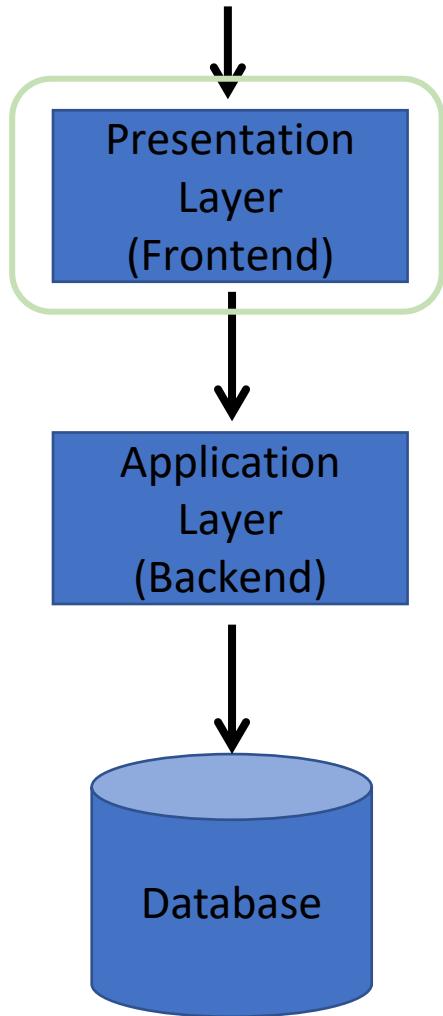
- Use this to answer the interview q – “How do you ensure your design is good?”
- Understand the priority for the application
- Well Architected Review (WAR)

# Three-Tier Architecture

# 3 Tiers

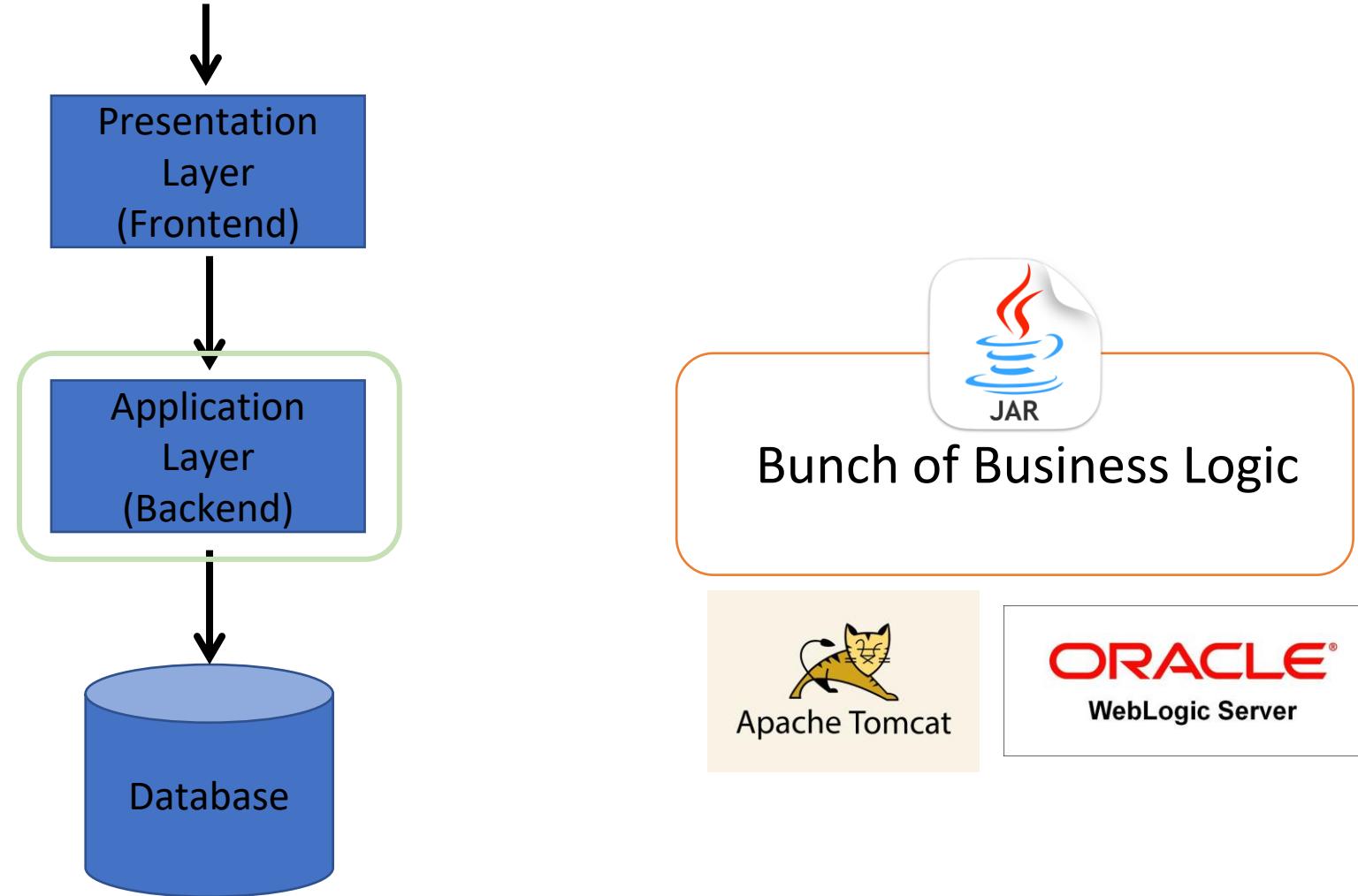


# 3 Tiers

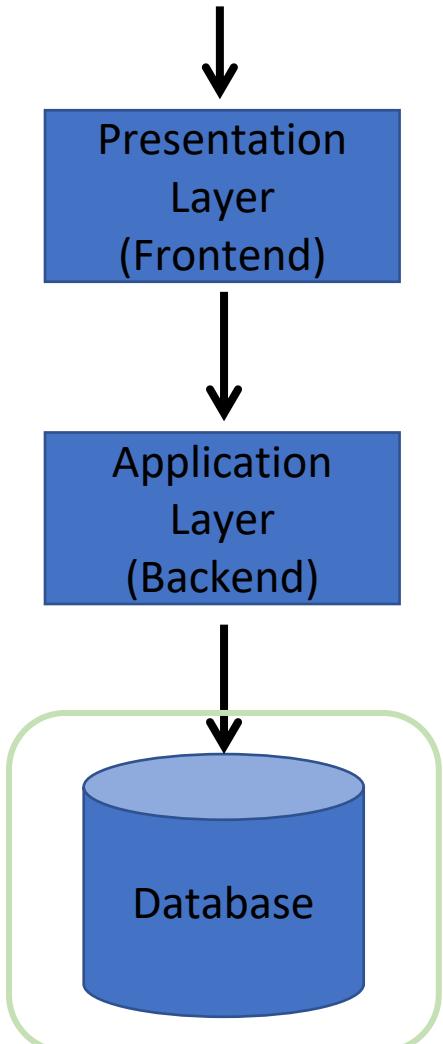


Apache Web Server

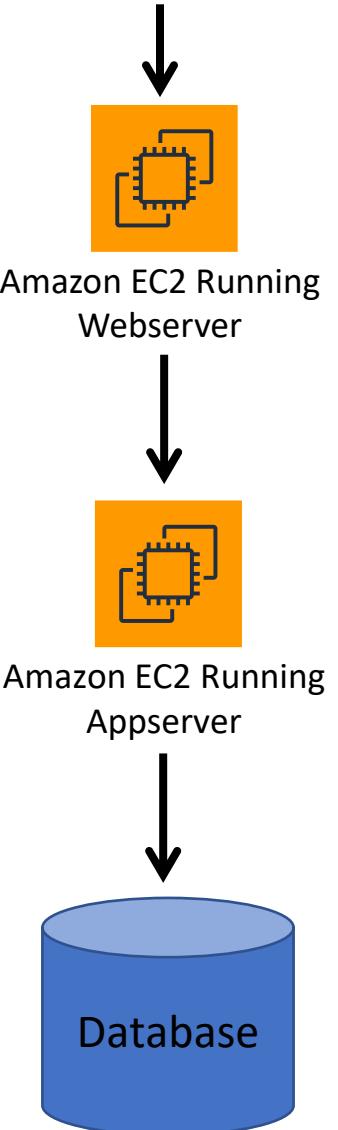
# 3 Tiers



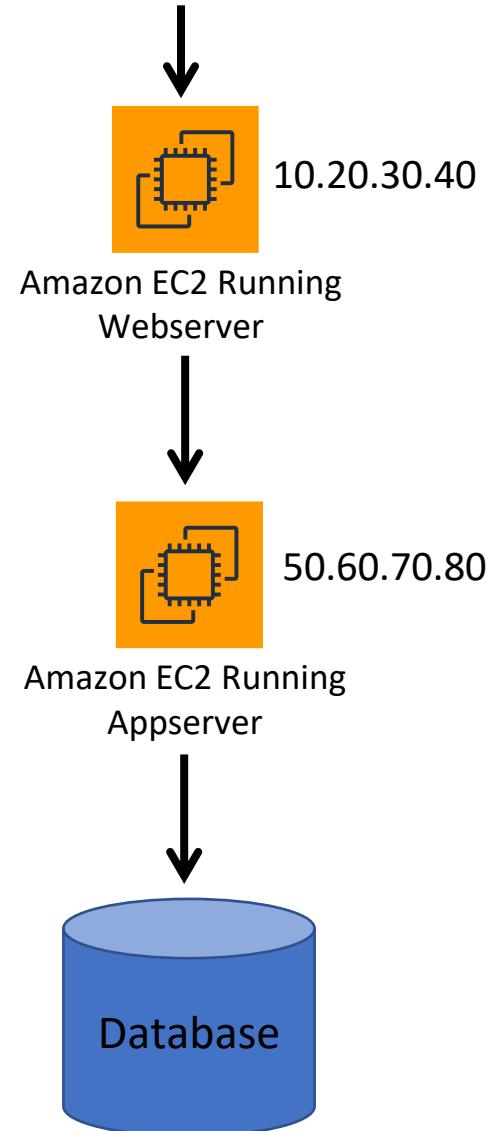
# 3 Tiers



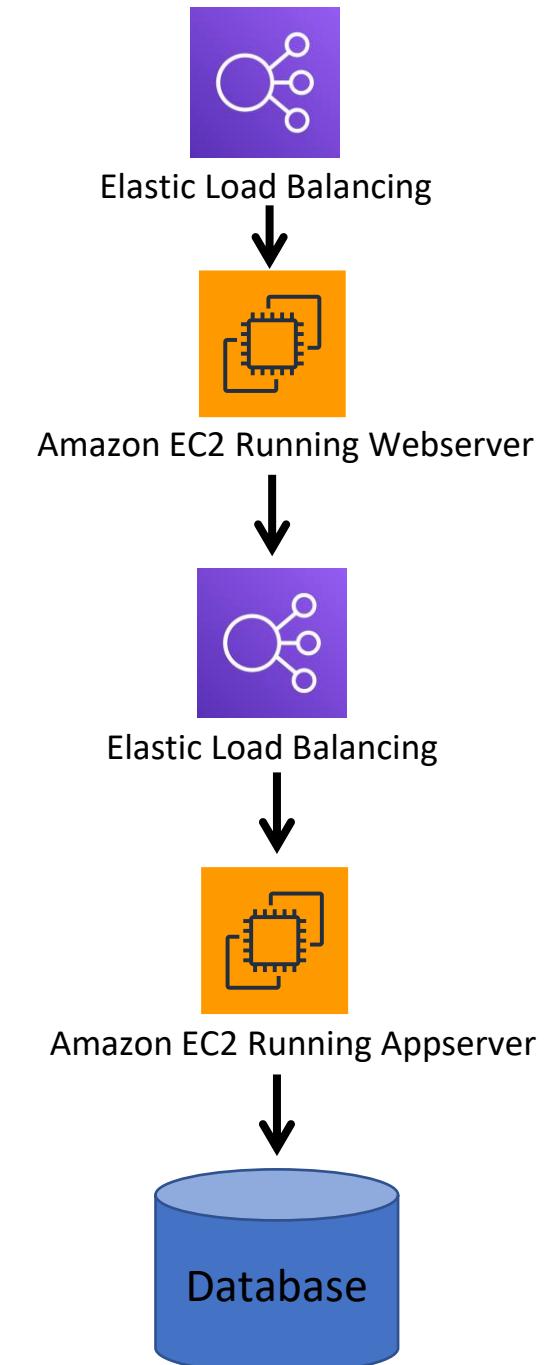
# 3 Tiers



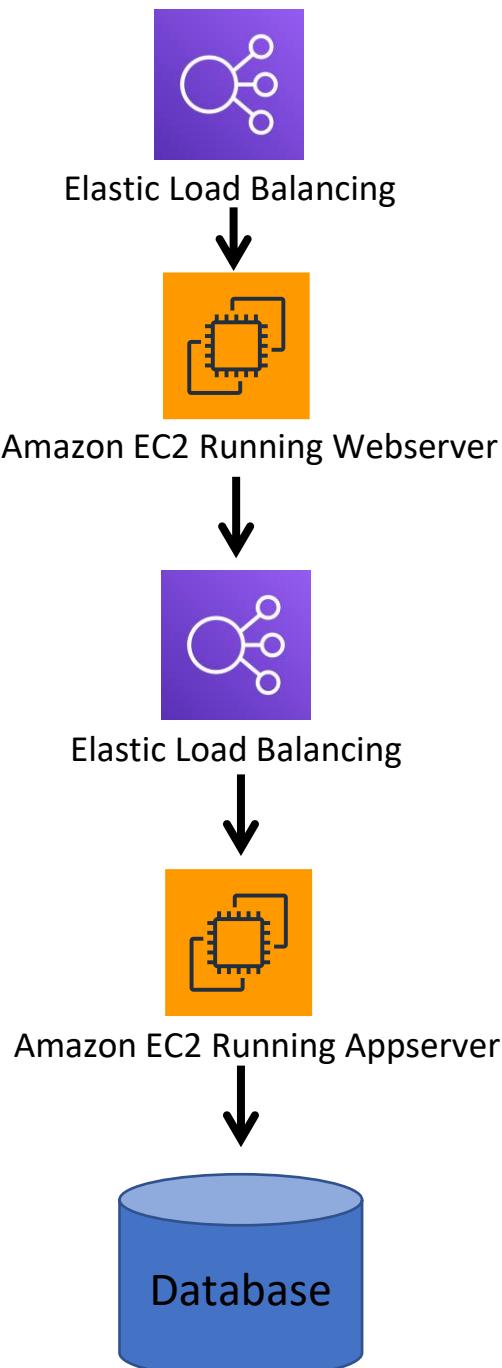
# 3 Tiers



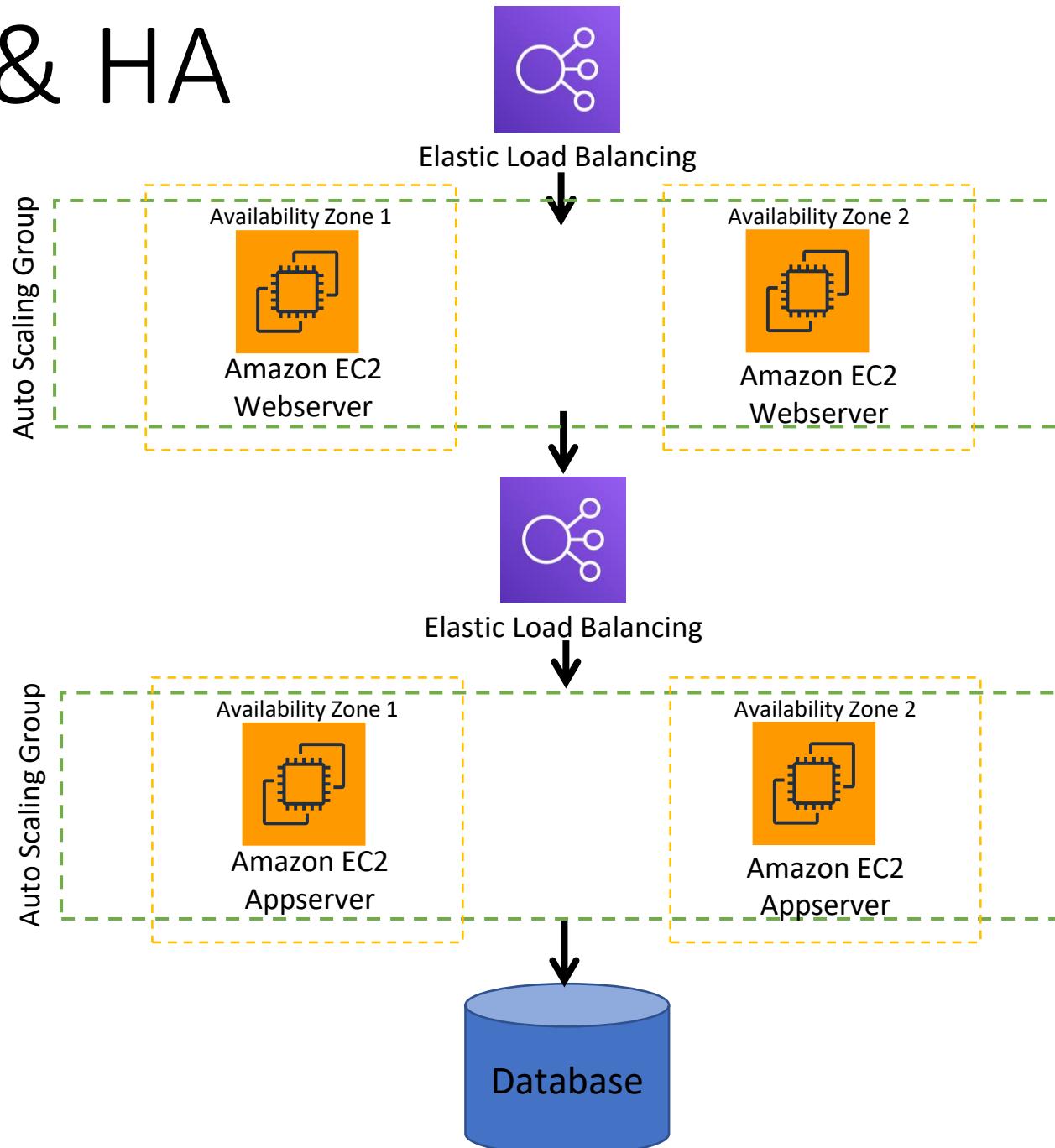
# 3 Tiers



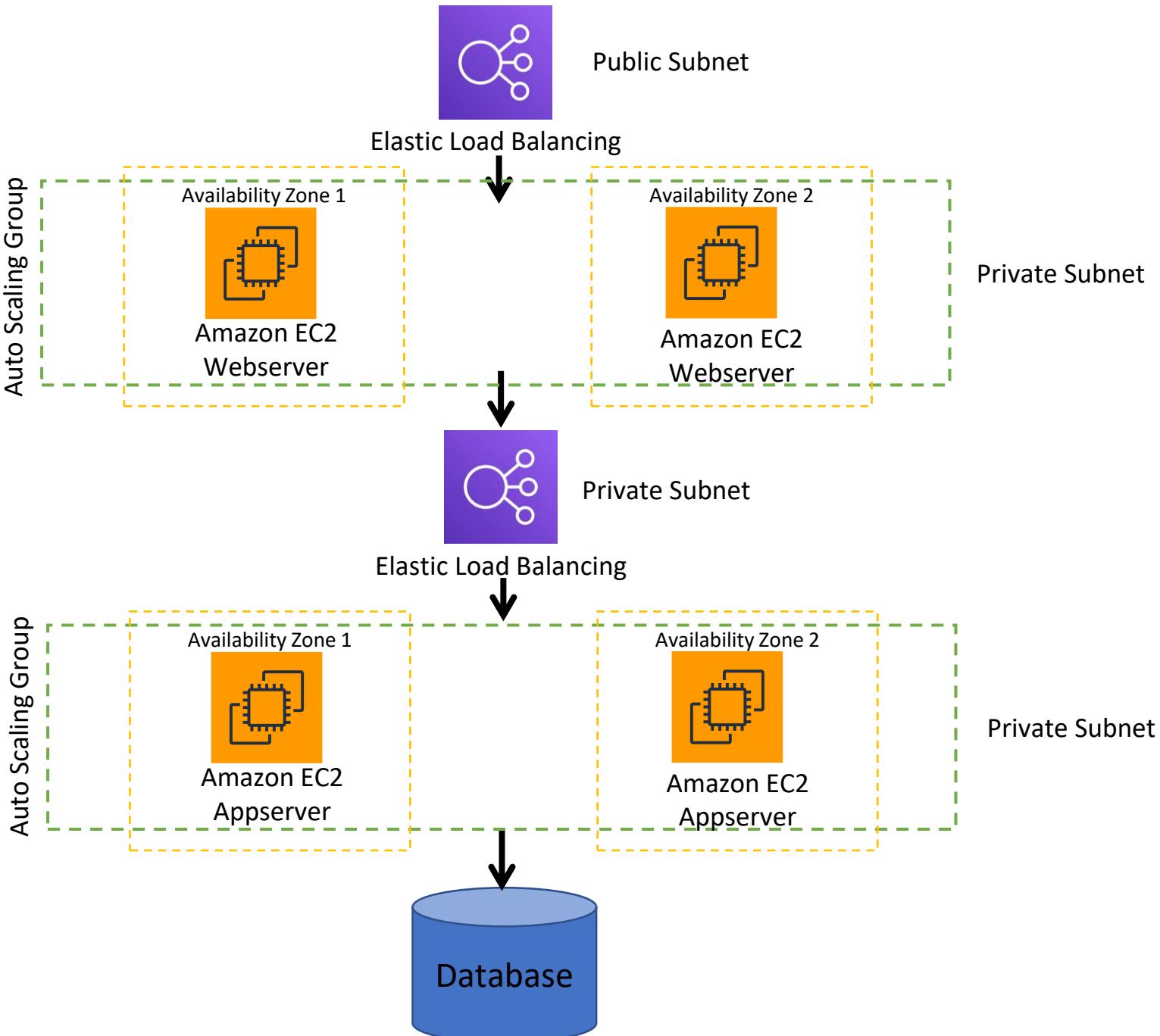
# Single Points Of Failure



# Scalable & HA

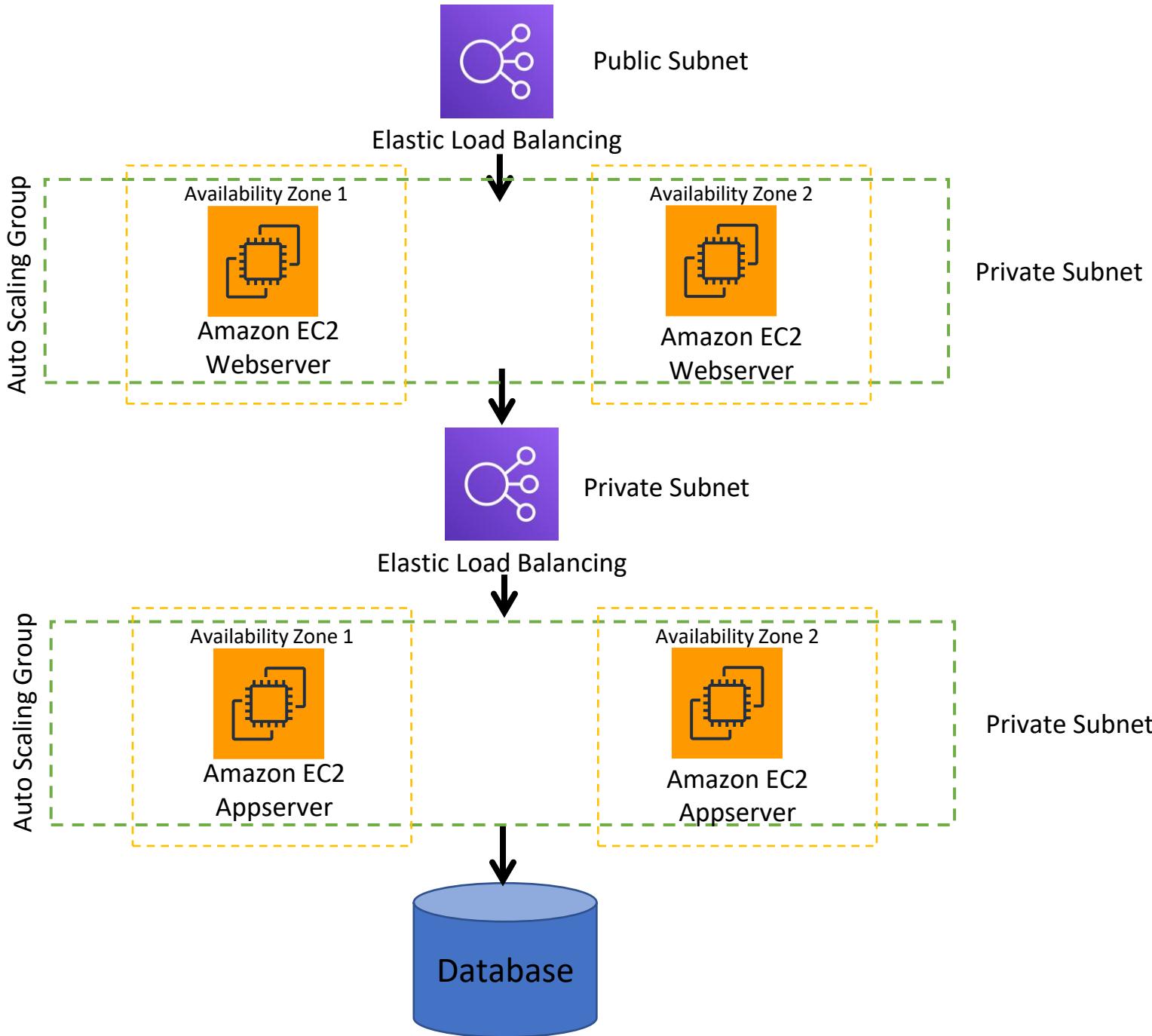


# Network Security



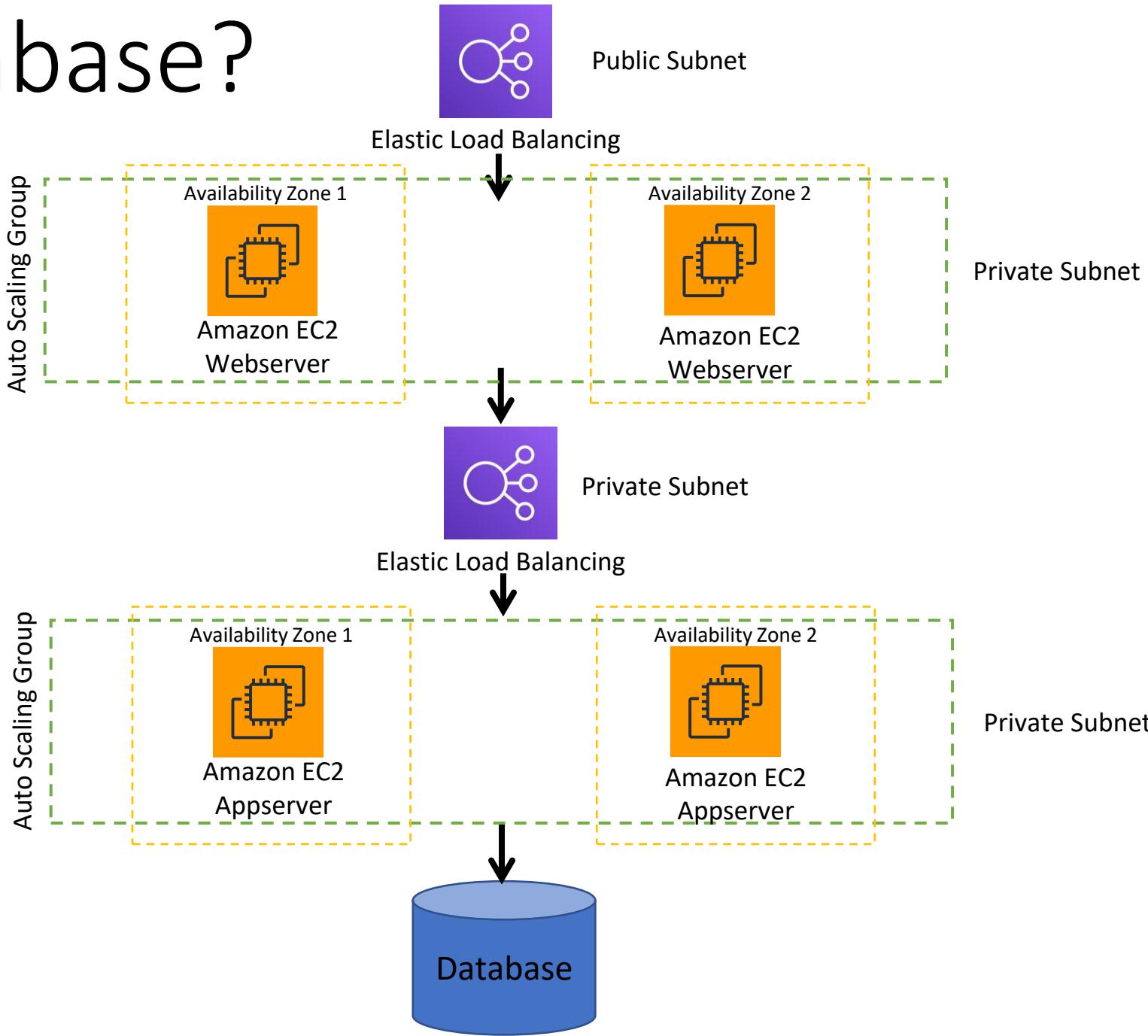
# Network Security

- NACL
- Security Group
- WAF with Load Balancer



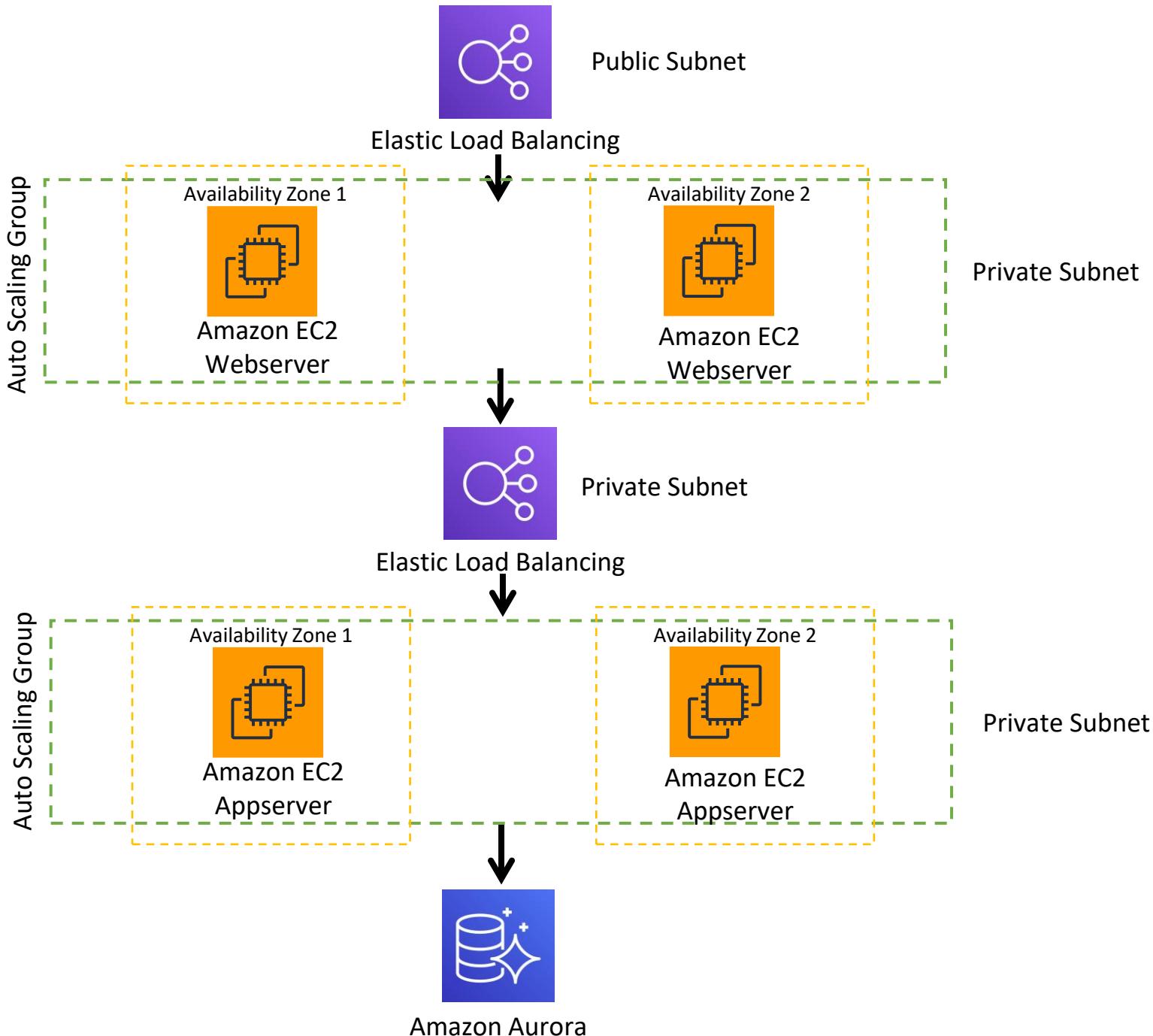
# How about Database?

- SQL Vs. NoSQL
- Use AWS Native Databases



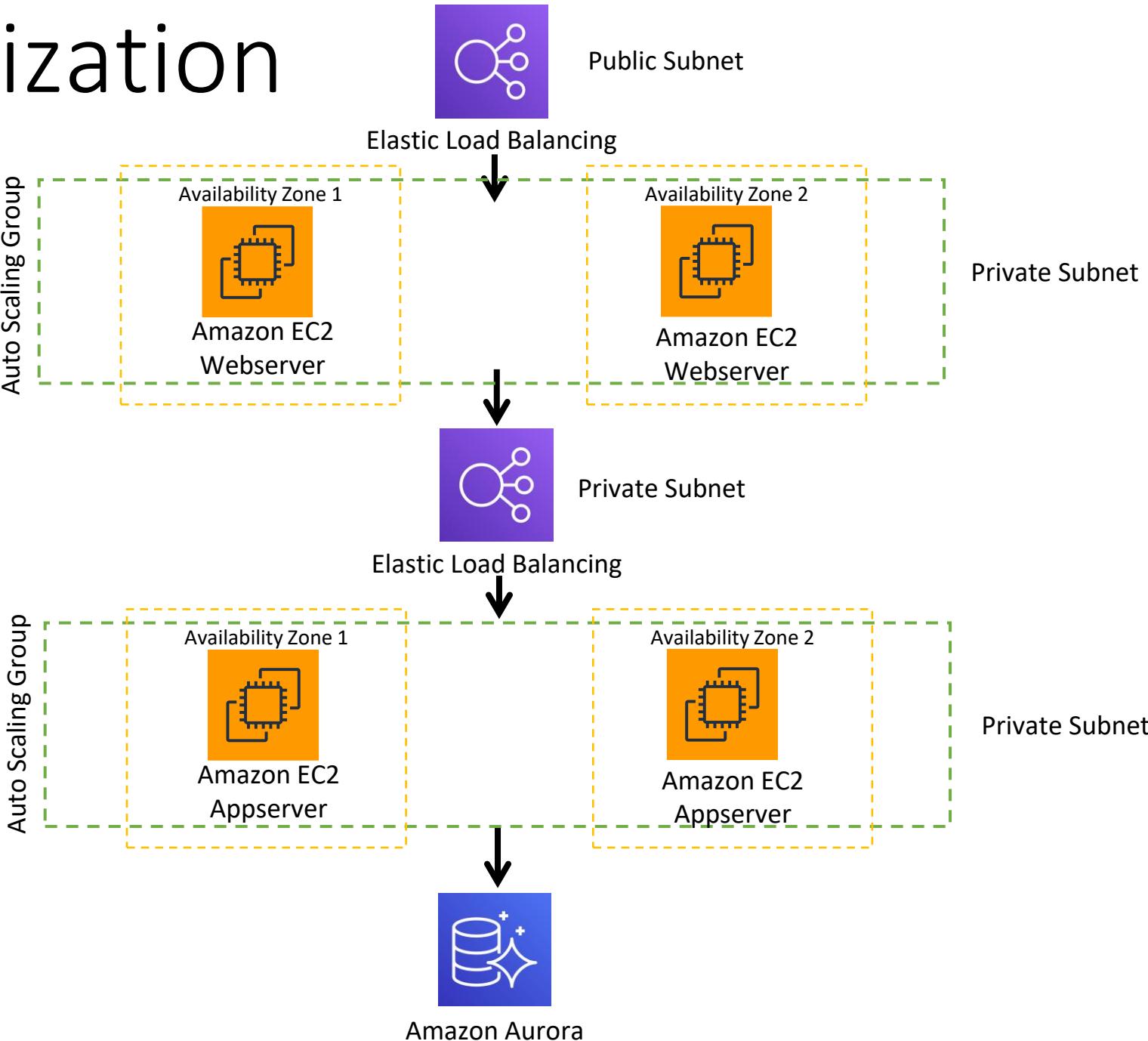
# Database HA

- Multi-AZ
- Global Database (Replication)



# Database Optimization

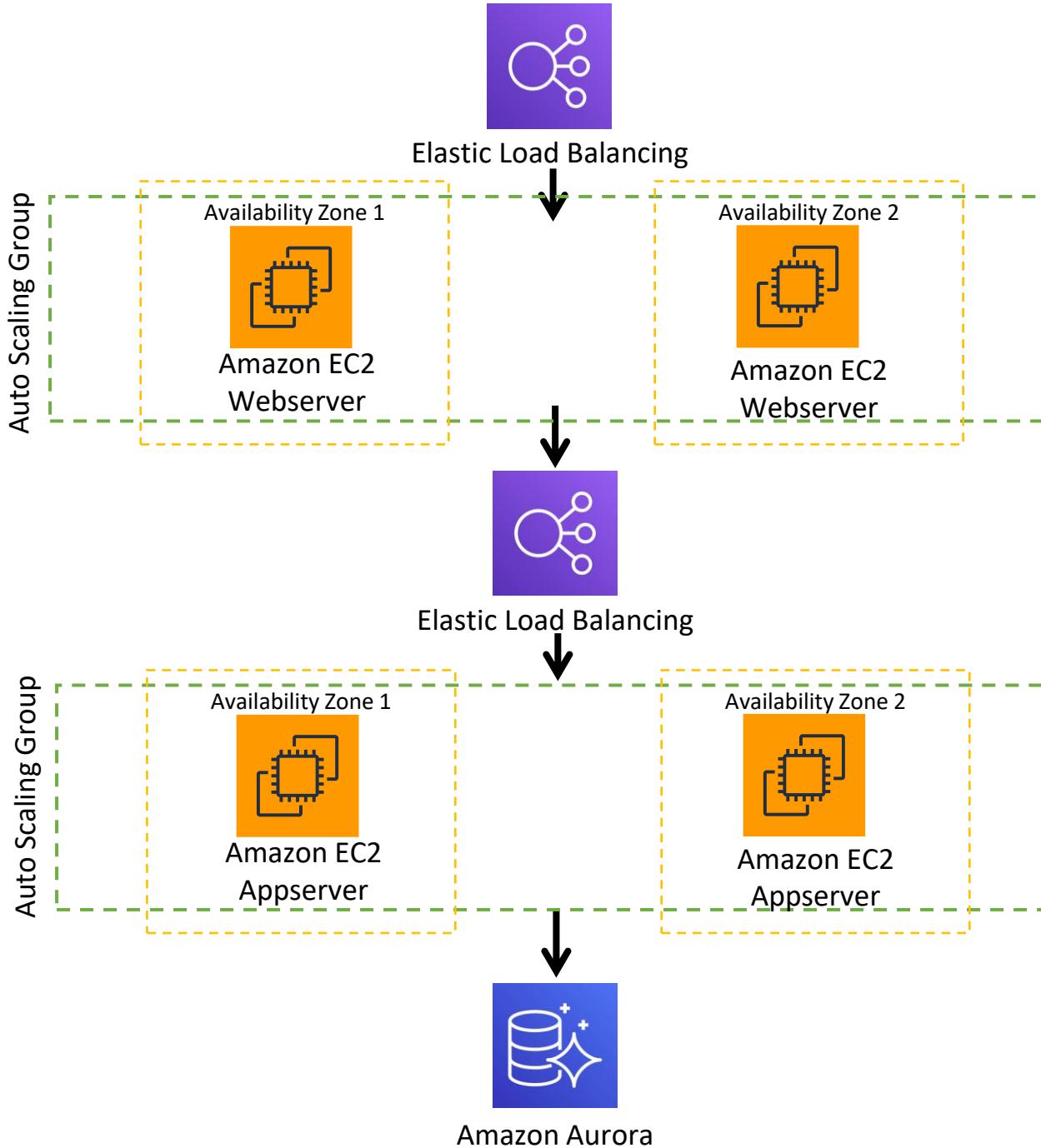
- Read Replica
- Caching Layer
- Query Tuning



## Presentation Layer

## Application Layer

## Database

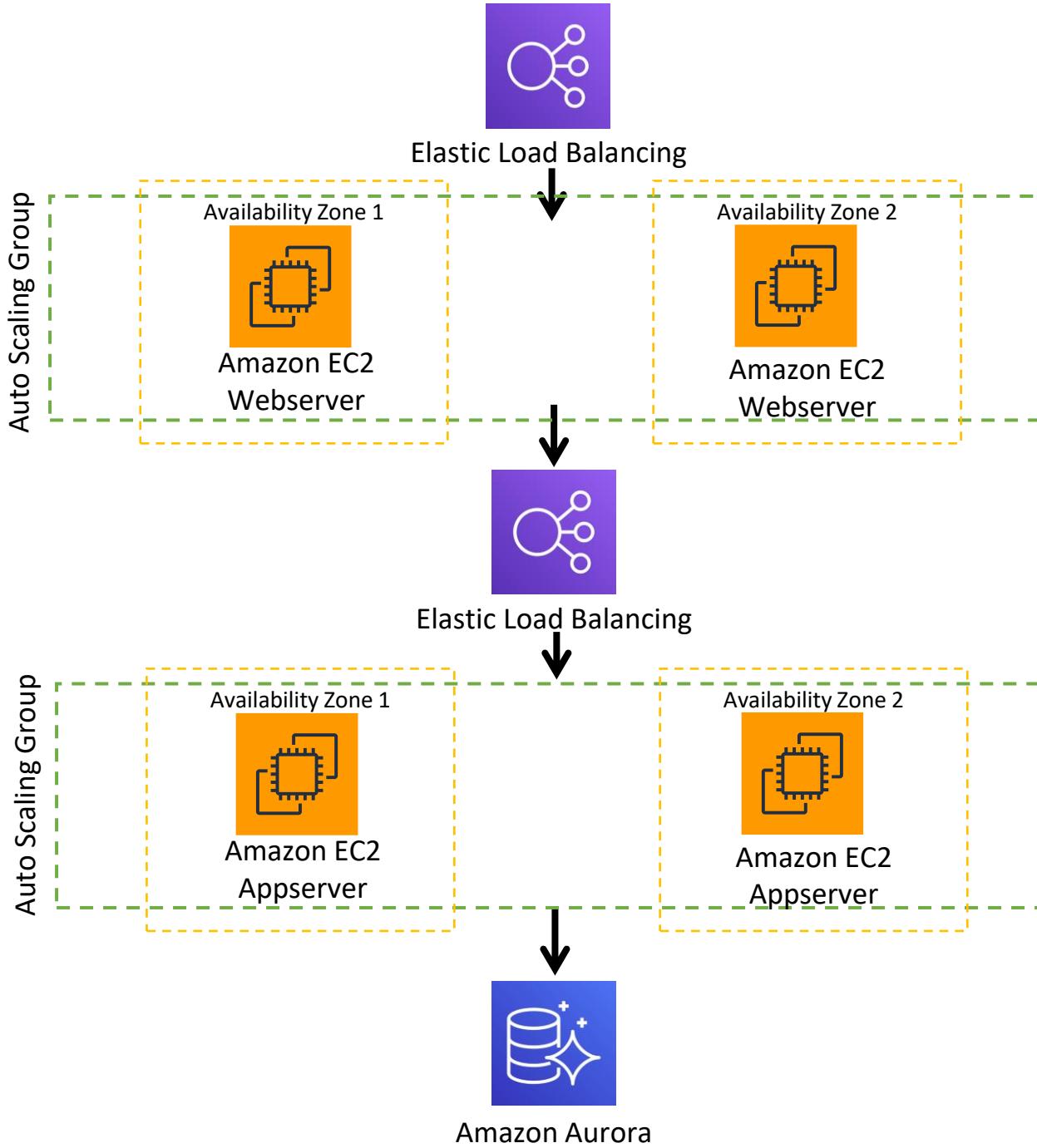


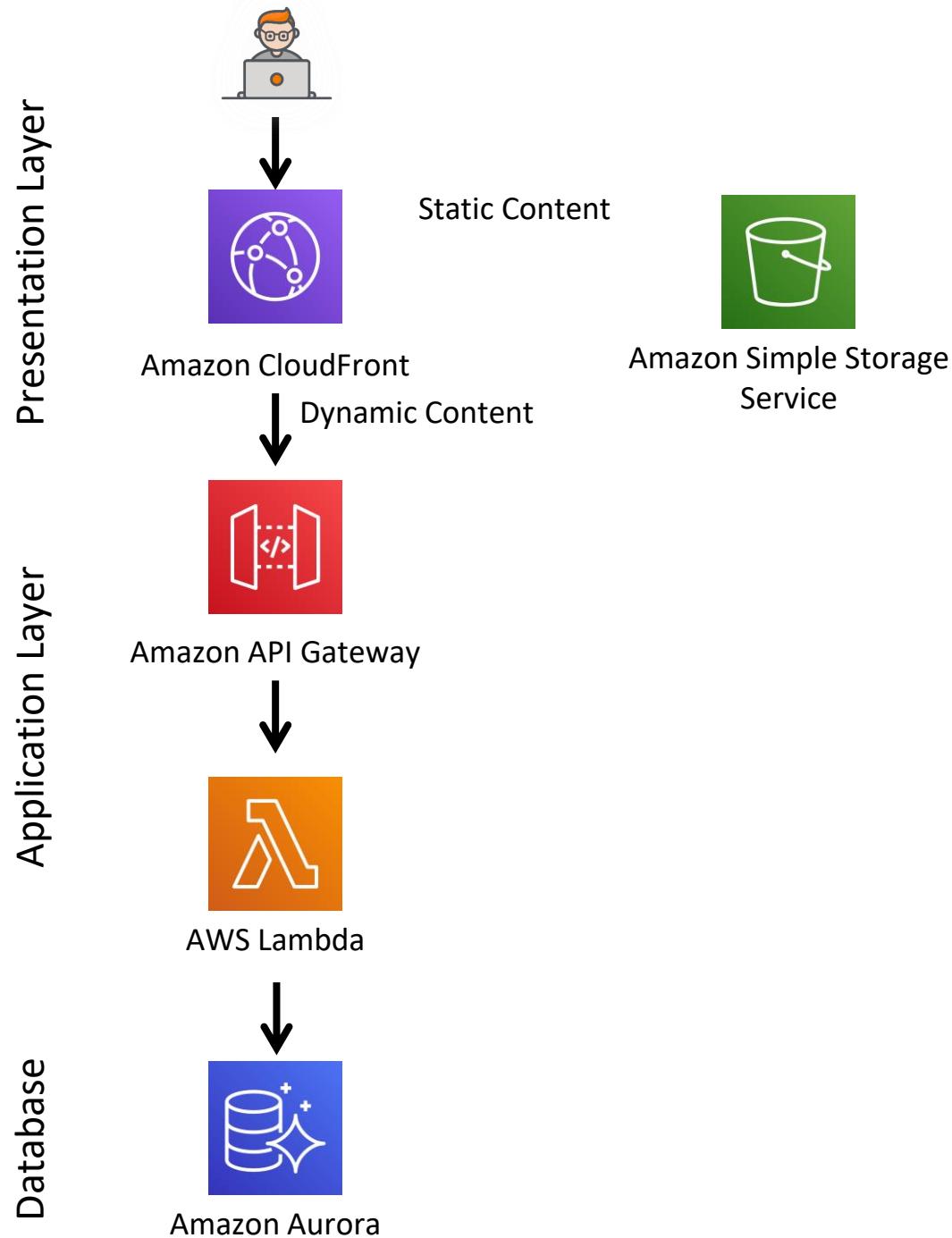
# Three-Tier Architecture with Serverless

## Presentation Layer

## Application Layer

## Database



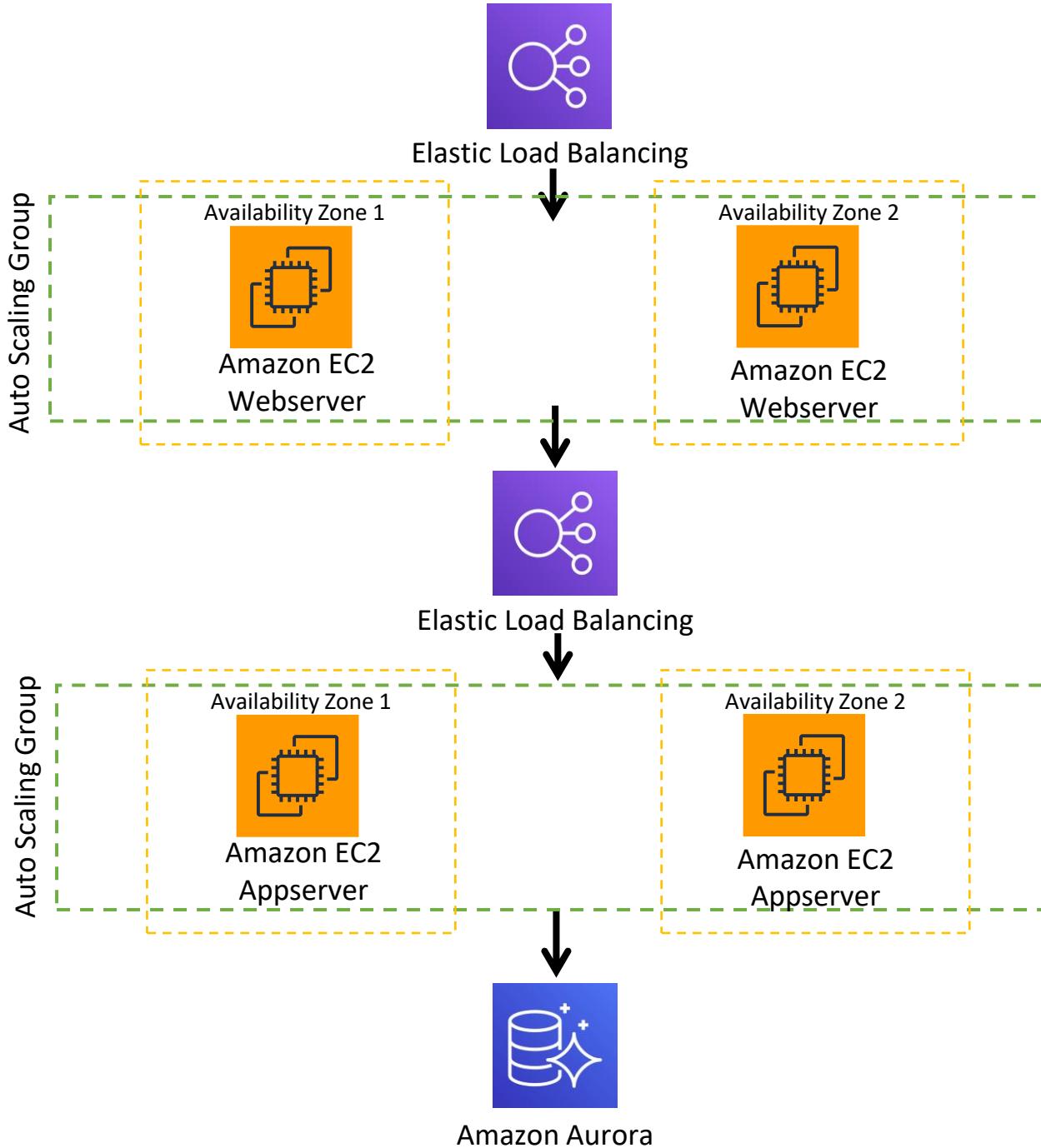


# Three-Tier Architecture with Kubernetes

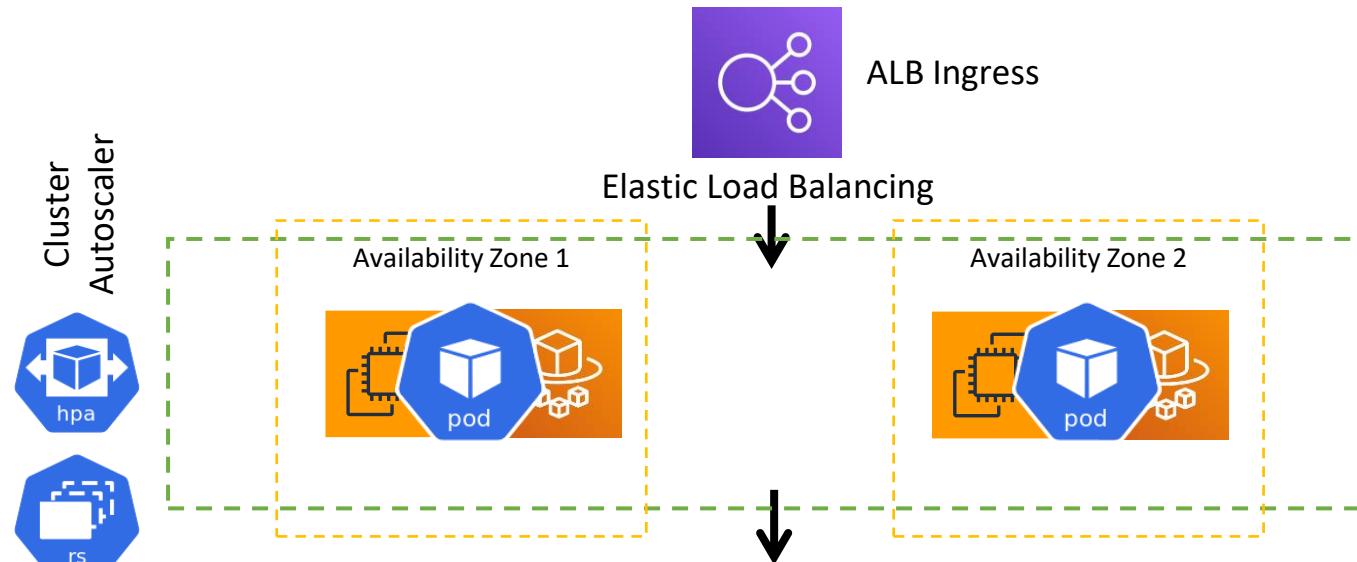
## Presentation Layer

## Application Layer

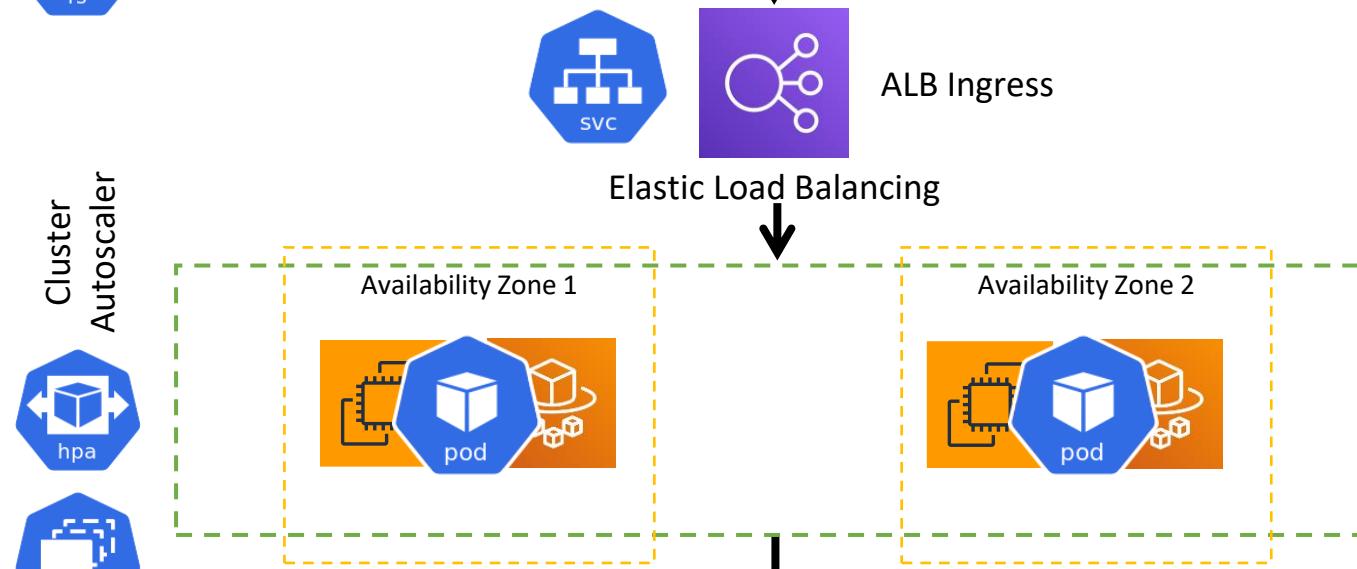
## Database



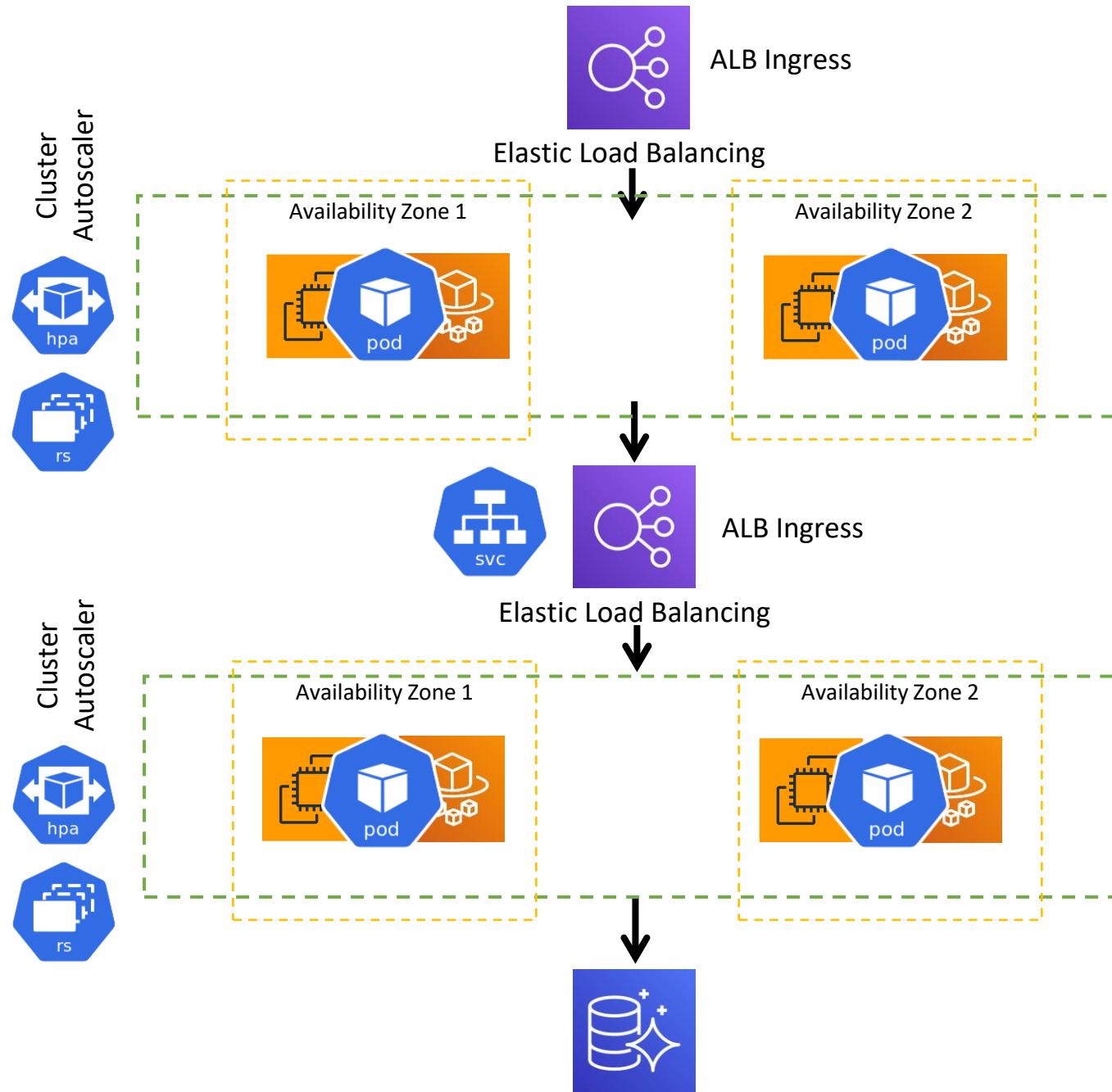
## Presentation Layer



## Application Layer



## Database

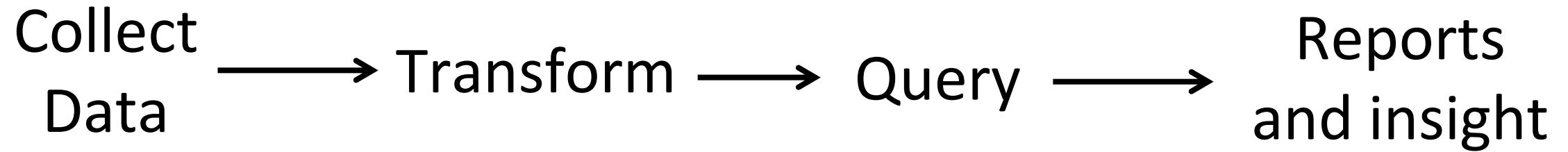


Amazon Aurora

Copyright © Rajdeep Saha - All Rights Reserved

# Data Analytics System Design on AWS

# Steps of Data Analytics

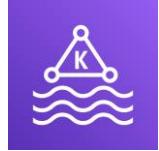


# Steps of Data Analytics

Collect  
Data



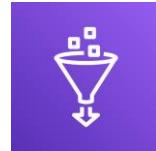
Amazon Kinesis



Amazon Managed  
Streaming for Kafka

→ Transform → Query →

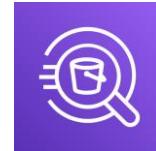
Reports  
and insight



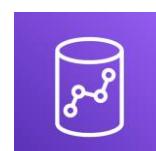
AWS Glue



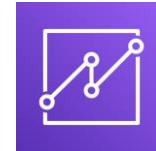
Amazon EMR



Amazon Athena



Amazon Redshift



Amazon QuickSight



Amazon EMR



Amazon Simple Storage  
Service



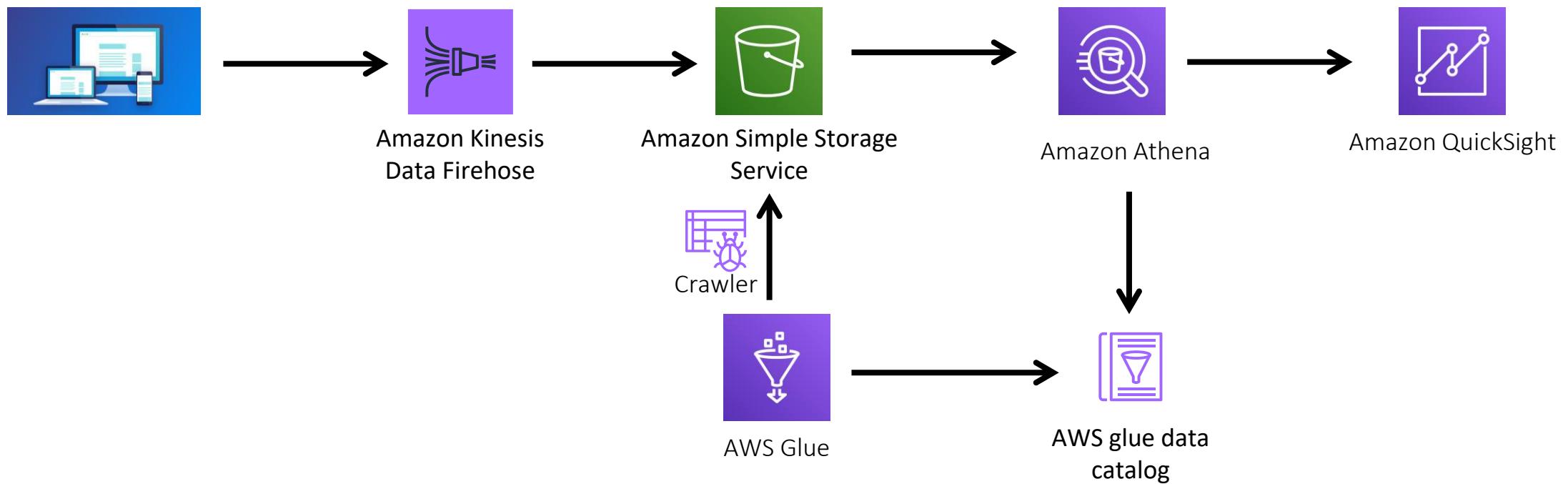
Amazon Elasticsearch  
Service



Amazon SageMaker

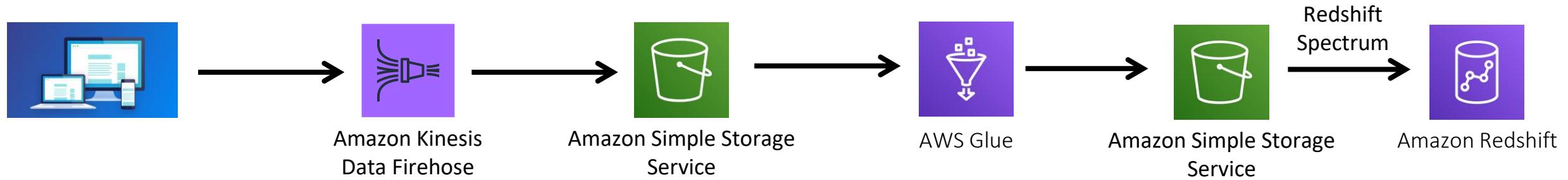
# Sample Architecture #1

## Query and report on click stream



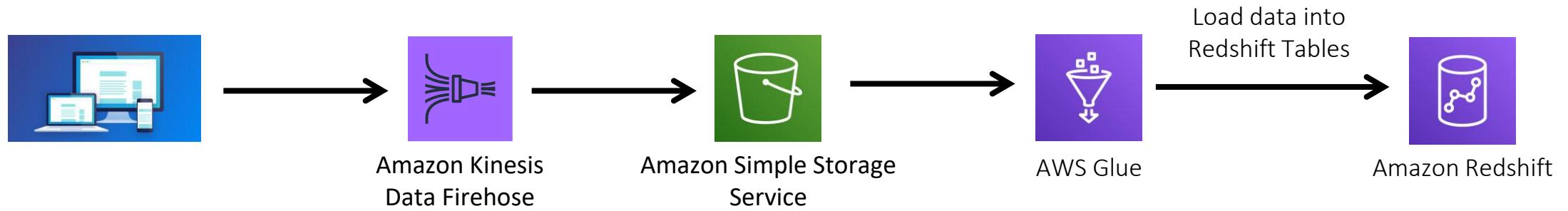
# Sample Architecture #2a

## ETL and data warehouse



# Sample Architecture #2b

## ETL and data warehouse



# Quick Detour into AWS Glue



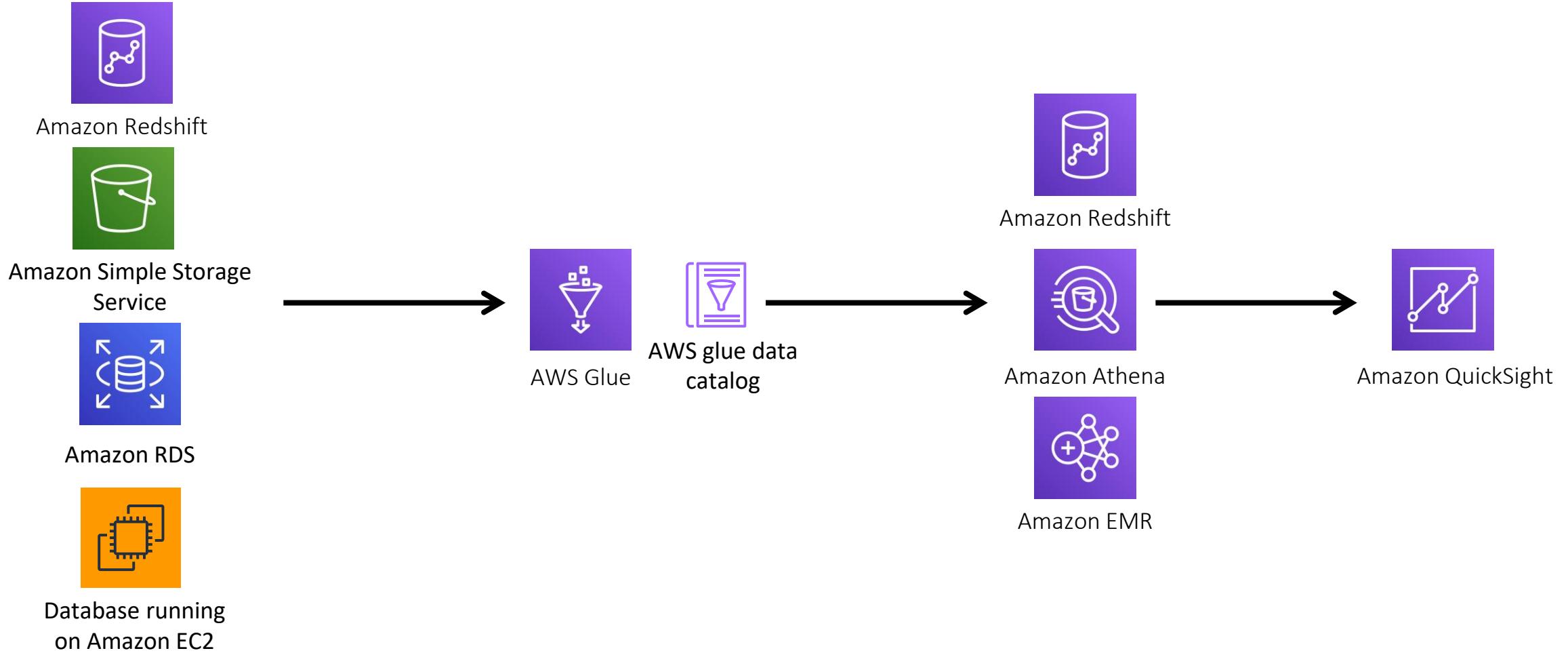
AWS Glue



- Serverless data integration tool
- Glue crawlers can run on data and create metadata
- Visually create ETL flow (Supports Python/Spark and Scala)
- Enrich, clean, and normalize data without writing code (Glue Databrew)
- Replicate data across various sources (Glue Elastic Views)

# Sample Architecture #3

## Unified catalog across multiple data stores



# Amazon EMR

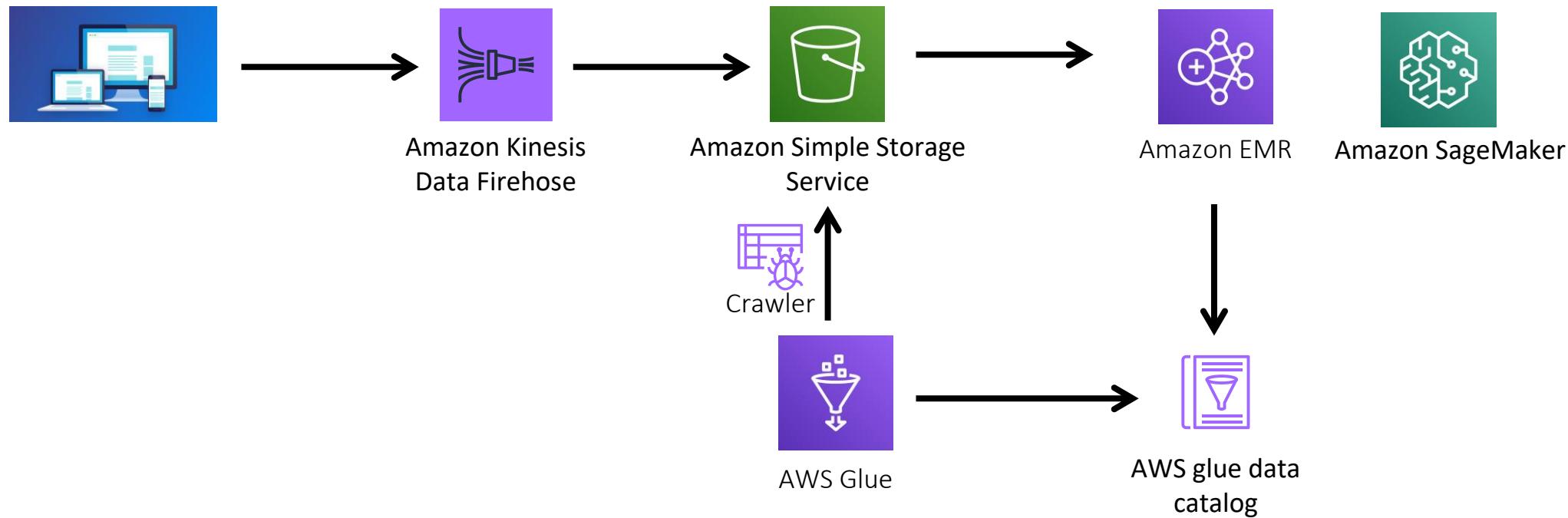


Amazon EMR

- Managed big data platform from AWS
- Runs open-source tools – Apache Spark, Apache Hive, Apache Hbase, Apache Flink, Apache Hudi, and Presto
- Run on EC2 or EKS (Elastic Kubernetes Service), or on-prem using EMR on Outposts

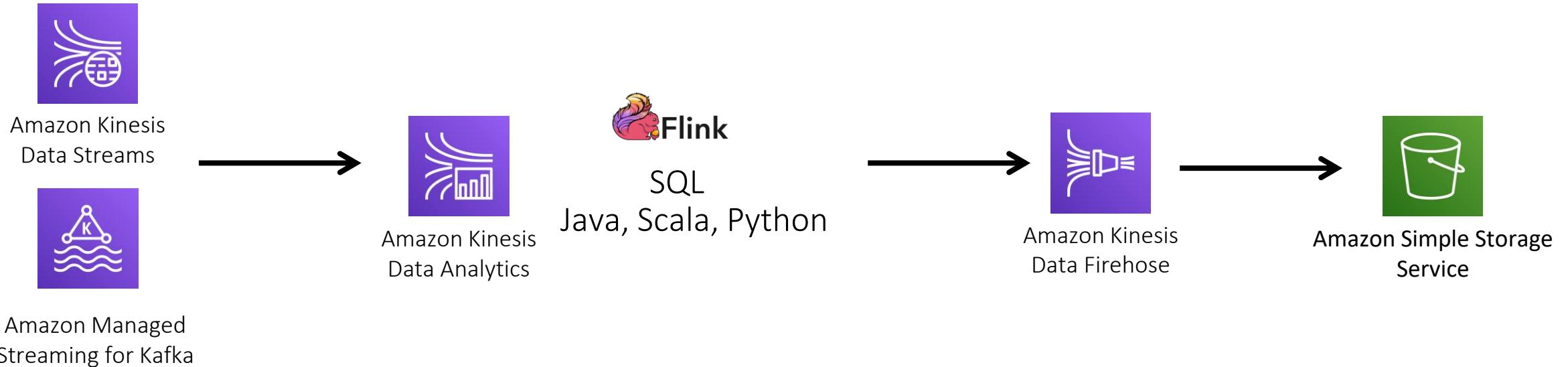
# Sample Architecture #4

## Big data analysis of click stream data

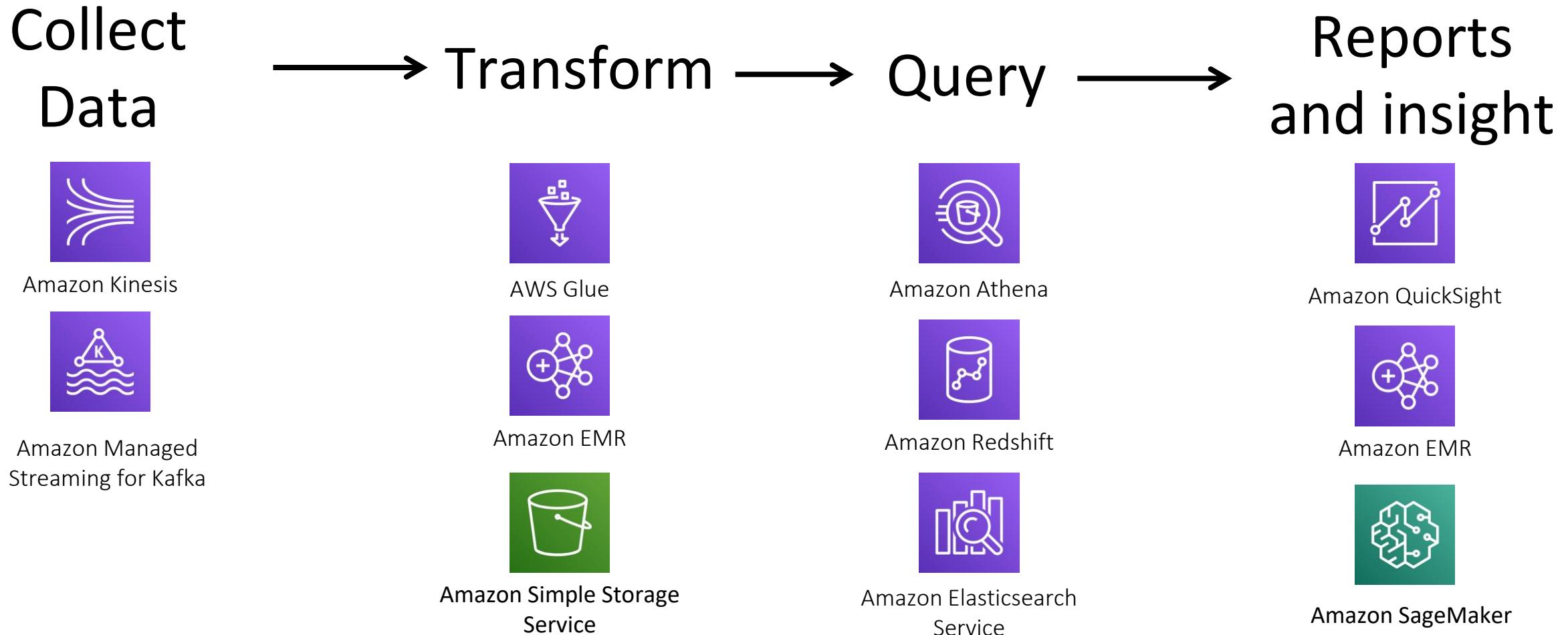


# Sample Architecture #5

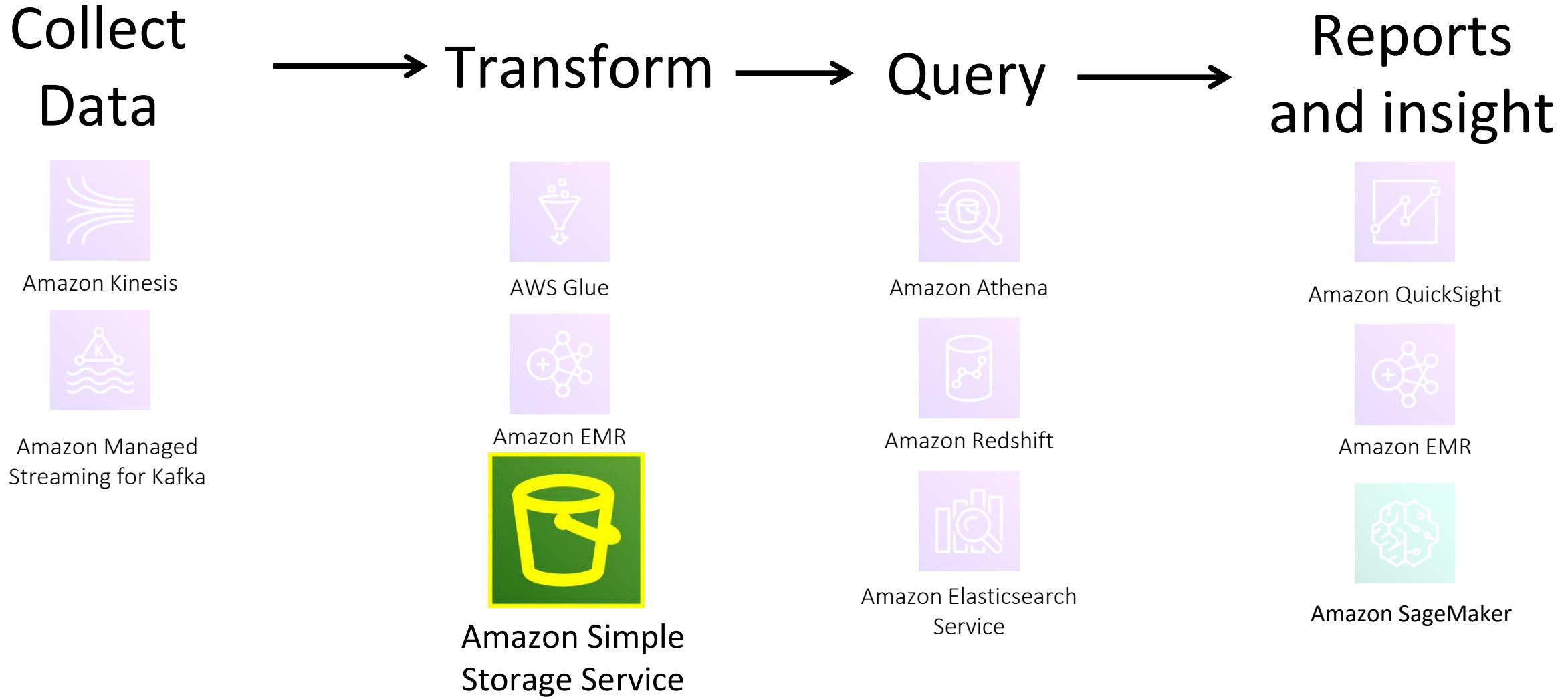
## In stream querying and ETL



# The Illusive Data Lake



# The Illusive Data Lake



# Using Well Architected for Performance/Cost Optimization (Challenge Faced Question)

# Tackling Any Tuning/Troubleshooting

- Monitor
- Measure
- Remediate

# Tackling Any Tuning/Troubleshooting

- Monitor
  - Logs
  - Metrics
  - Traces
- Measure
  - Define KPI
  - Send alarms
- Remediate
  - Configuration
  - Code

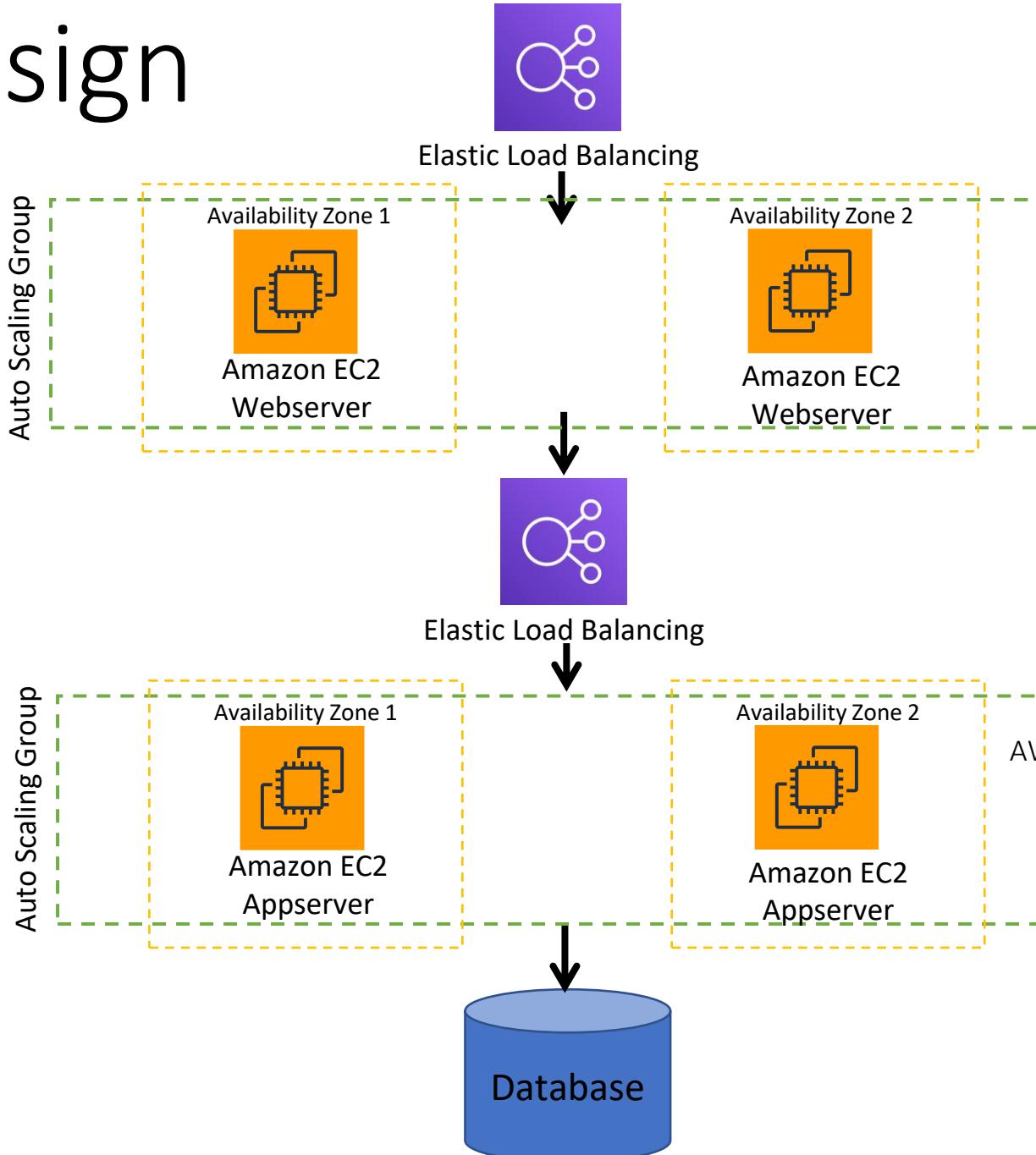
# EC2 Based Application

- Monitor
  - Logs
  - Metrics – CPU/Memory Utilization on CloudWatch
  - Traces
- Measure
  - Define KPI
  - Send alarms – CloudWatch Alarm
- Remediate
  - Configuration – Used home grown algorithm/compute optimizer to optimize EC2 capacity
  - Code

# Lambda Based Application

- Monitor
  - Logs
  - Metrics – Lambda is throttling (but already have high memory)
  - Traces – Enabled X-Ray trace
- Measure
  - Define KPI – found which section is taking longer
  - Send alarms
- Remediate
  - Configuration
  - Code – Moved database connection to global section

# 3-Tier Design



Amazon CloudWatch



AWS X-Ray



AWS Compute Optimizer  
Identifies whether your AWS resources are optimal, and offers recommendations to improve cost and performance



AWS Cost Explorer



spot instances

CloudHealth  
by vmware®



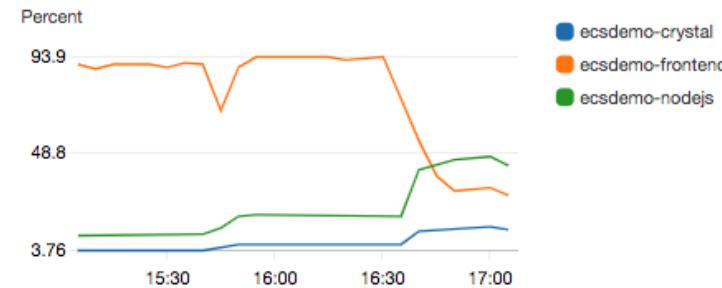
ECS Services ▾

fargate-demo-ECSClust... ▾

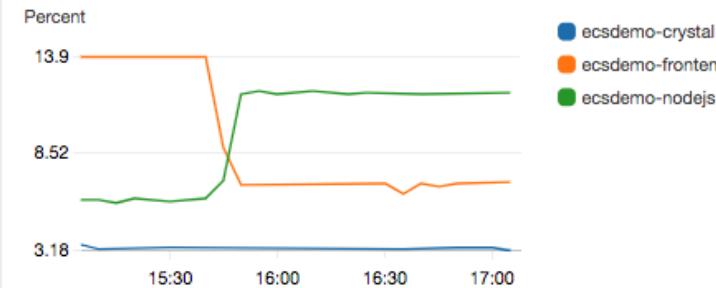
Filters: Filter services...

CloudWatch has recently announced the open preview of Container Insights to monitor your EKS and Kubernetes clusters. Please provide feedback through this link. You can also send email directly to [containerinsightsfeedback@amazon.com](mailto:containerinsightsfeedback@amazon.com).

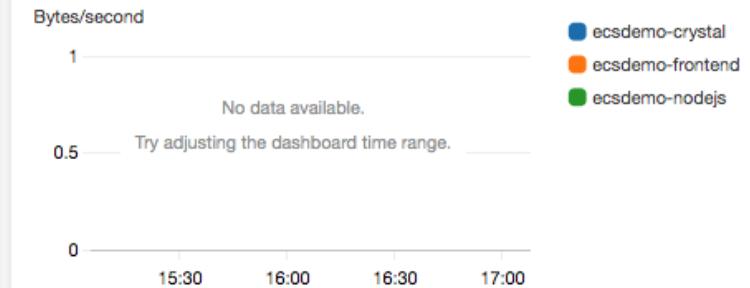
### CPU Utilization



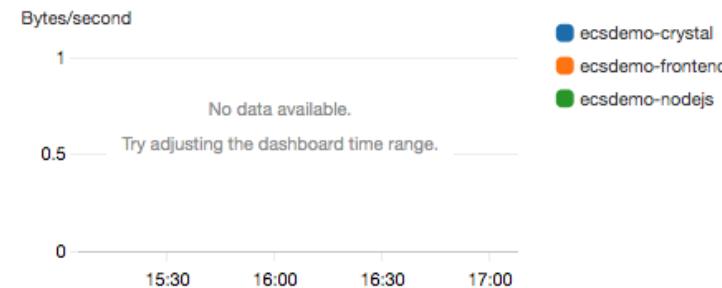
### Memory Utilization



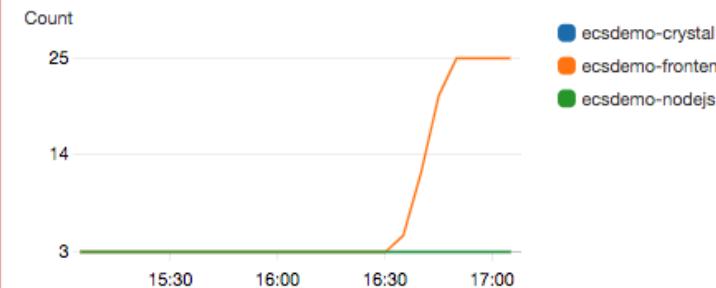
### Network TX



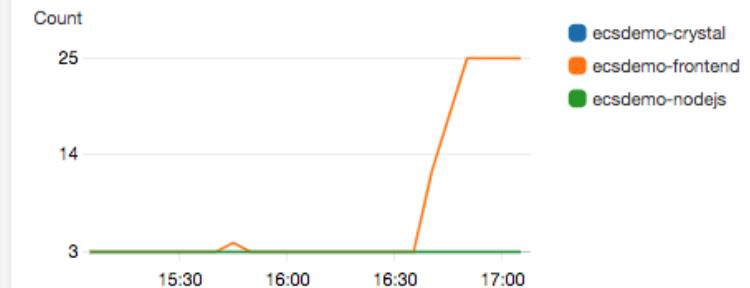
### Network RX



### Number of Desired Tasks



### Number of Running Tasks



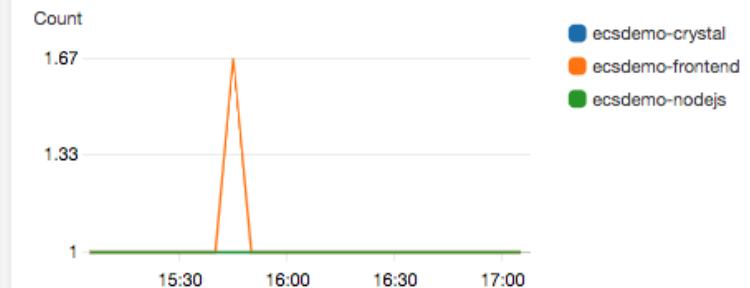
### Number of Pending Tasks



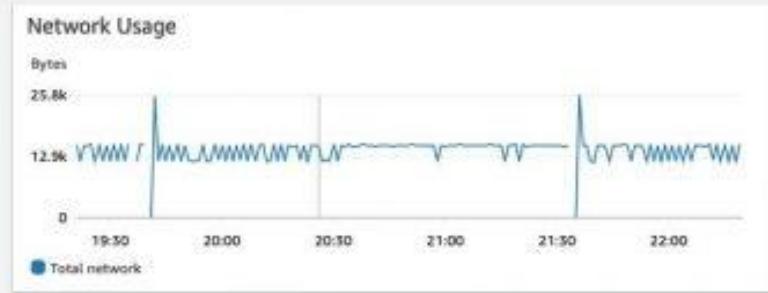
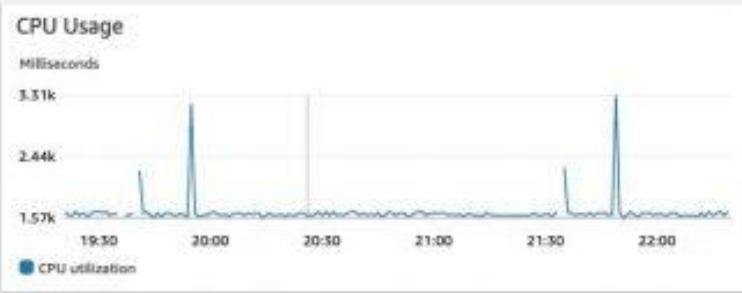
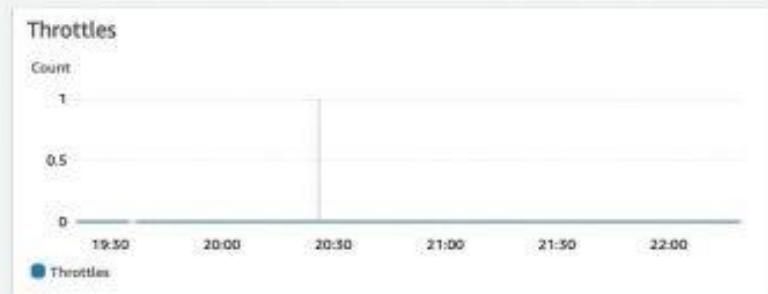
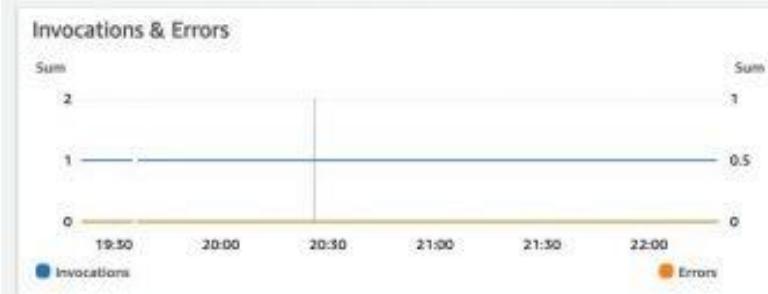
### Number of Task Sets



### Number of Deployments



## Performance monitoring

 1h 3h 12h 1d 3d 1w Custom  Add to dashboard  
Single function 
 ⚠ In alarm 0  Insufficient data 0  OK 0

[Invocations](#) [Application logs](#)

## Most recent 1000 invocations (177)

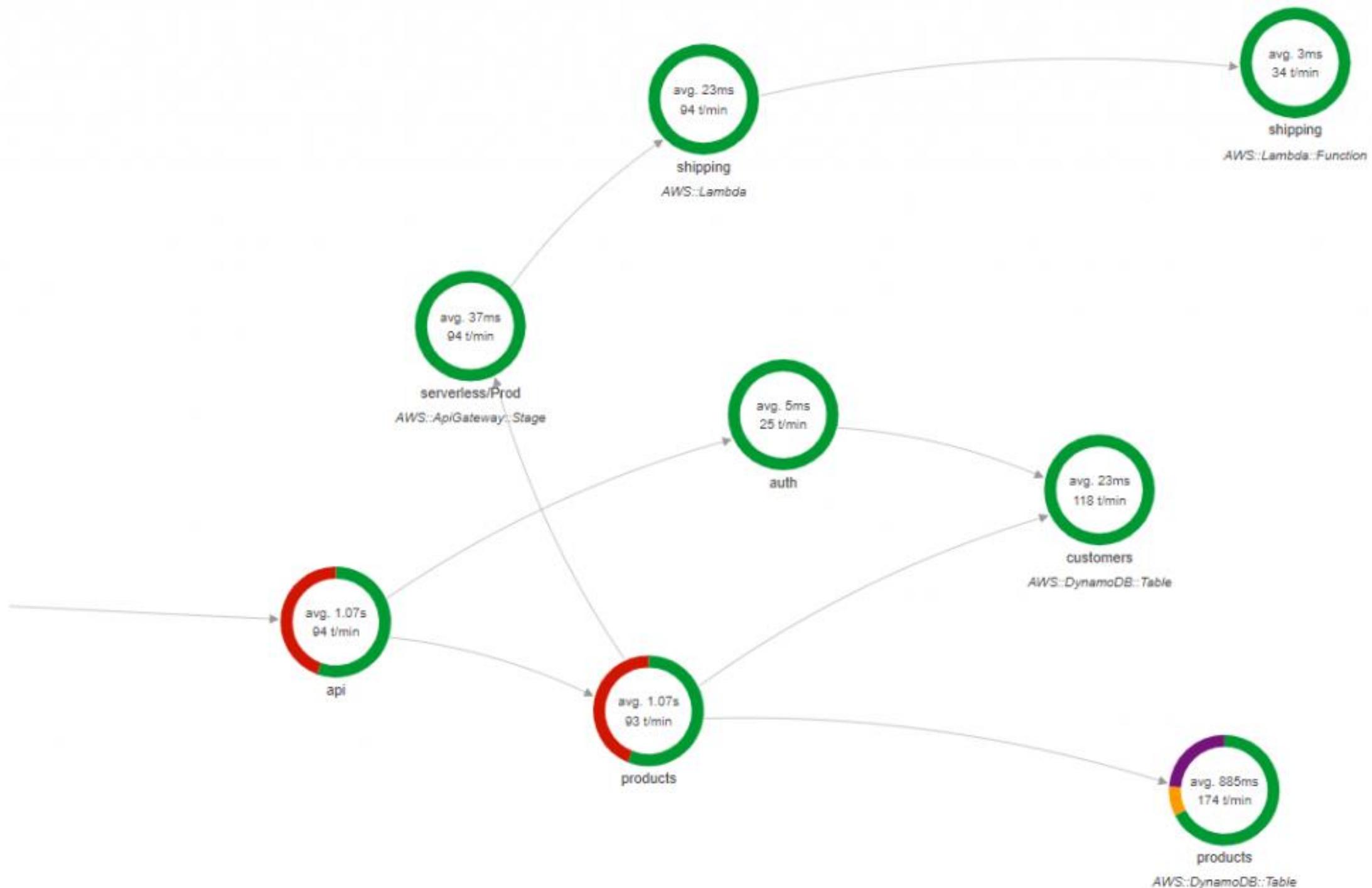
[View logs](#) 

 < 1 2 3 > 

<input type="checkbox"/>	Timestamp	Request ID	Trace	Memory %	Network IO	CPU time	Cold start
<input type="checkbox"/>	2020-10-21 02:18:43 (UTC-04:00)	6e7d206c-180c-4301-8eff-ad9262...	<a href="#">View</a> <input type="button"/>	<div style="width: 128%;"><div style="width: 100%;">128%</div></div>	11 kB	1600ms	-
<input type="checkbox"/>	2020-10-21 02:17:43 (UTC-04:00)	75652a4f-4a26-4703-a7db-de362...	<a href="#">View</a> <input type="button"/>	<div style="width: 128%;"><div style="width: 100%;">128%</div></div>	14 kB	1650ms	-
<input type="checkbox"/>	2020-10-21 02:16:43 (UTC-04:00)	21e0c8a7-cb97-47b1-88ae-31ef0...	<a href="#">View</a> <input type="button"/>	<div style="width: 128%;"><div style="width: 100%;">128%</div></div>	11 kB	1640ms	-
<input type="checkbox"/>	2020-10-21 02:15:43 (UTC-04:00)	24a1fcf1-0812-459d-a3c7-91d59...	<a href="#">View</a> <input type="button"/>	<div style="width: 128%;"><div style="width: 100%;">128%</div></div>	14 kB	1600ms	-
<input type="checkbox"/>	2020-10-21 02:14:43 (UTC-04:00)	04d56aa2-4a64-4309-88f0-50603...	<a href="#">View</a> <input type="button"/>	<div style="width: 128%;"><div style="width: 100%;">128%</div></div>	11 kB	1650ms	-



Clients





## AWS X-Ray

Getting Started

Service map

## Traces

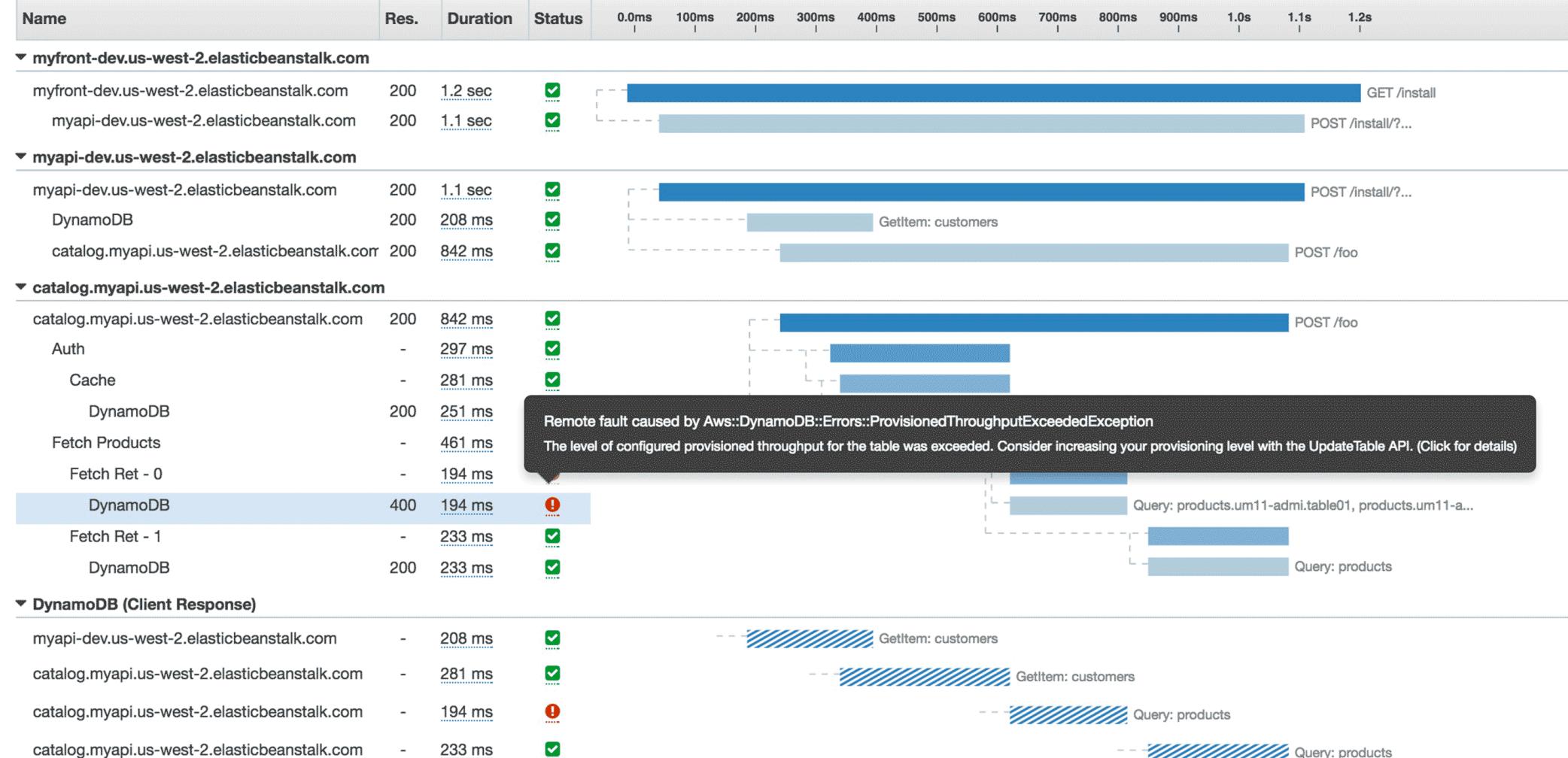
Enter Service Name, Annotation, Trace ID or click the Help icon for additional details



Traces &gt; 1-58214aaa-26811b4a16897a938c977b5e

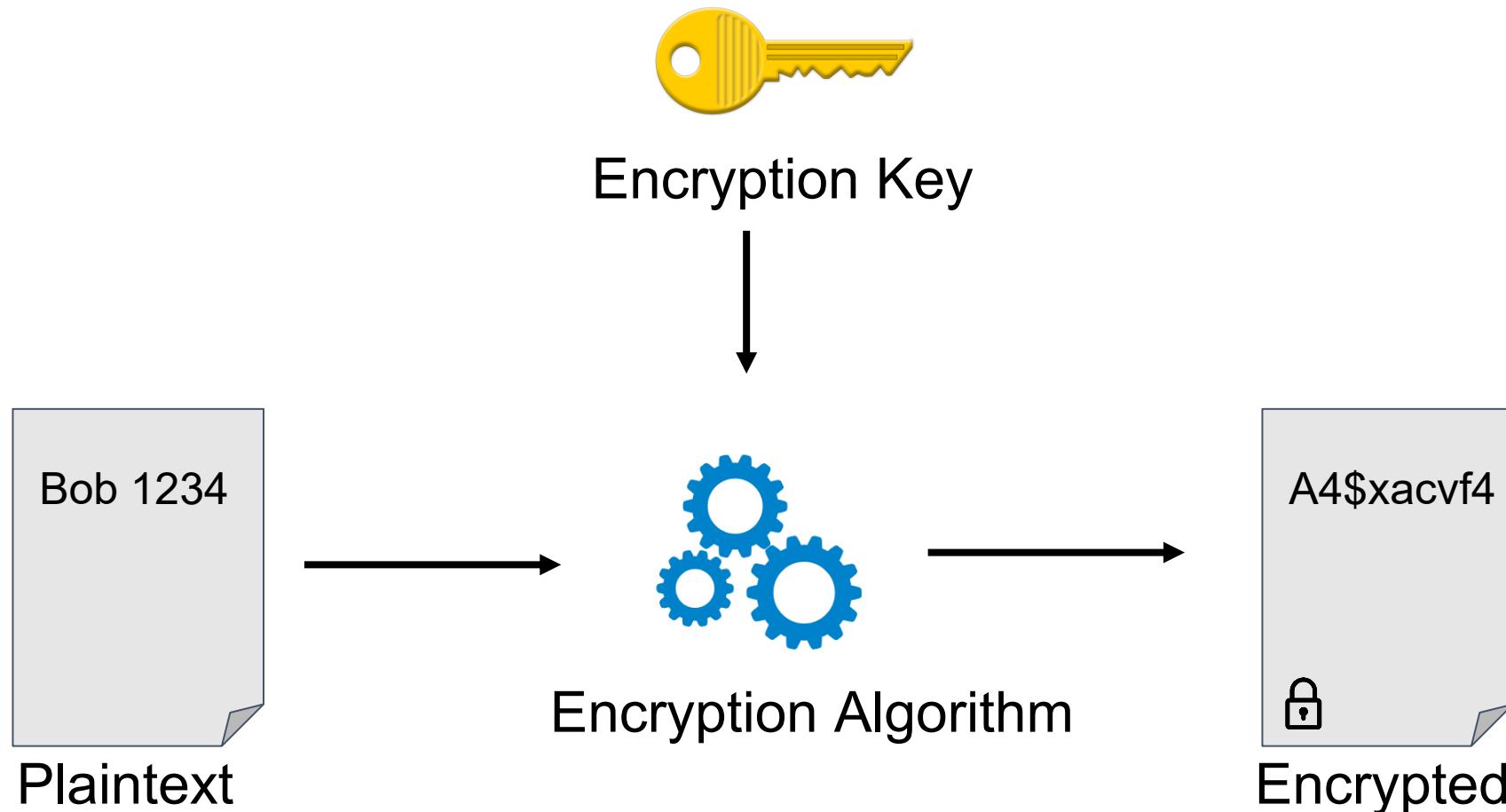
Timeline

Raw

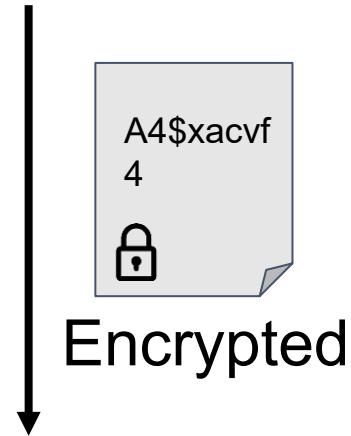
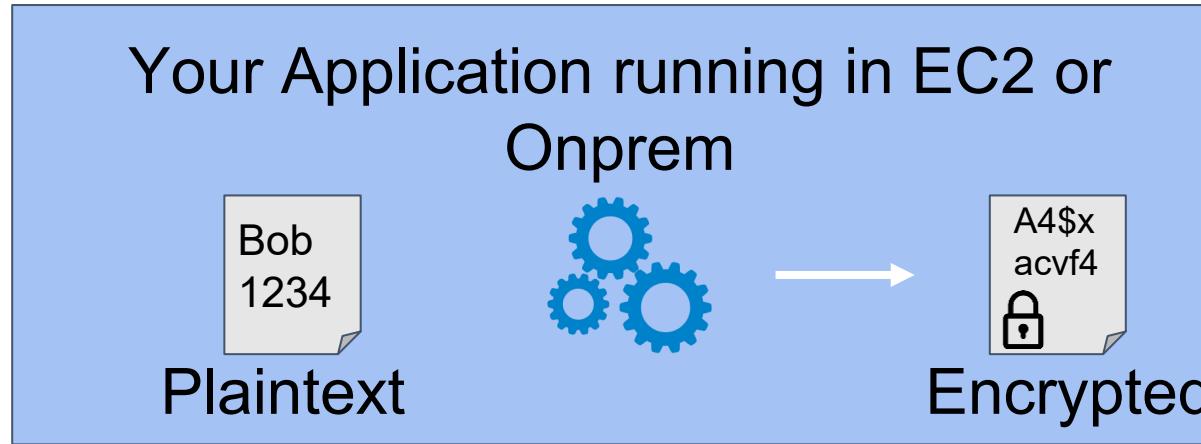


# Understanding Encryption at Rest & Client/Server Side Encryption

# Encryption Flow

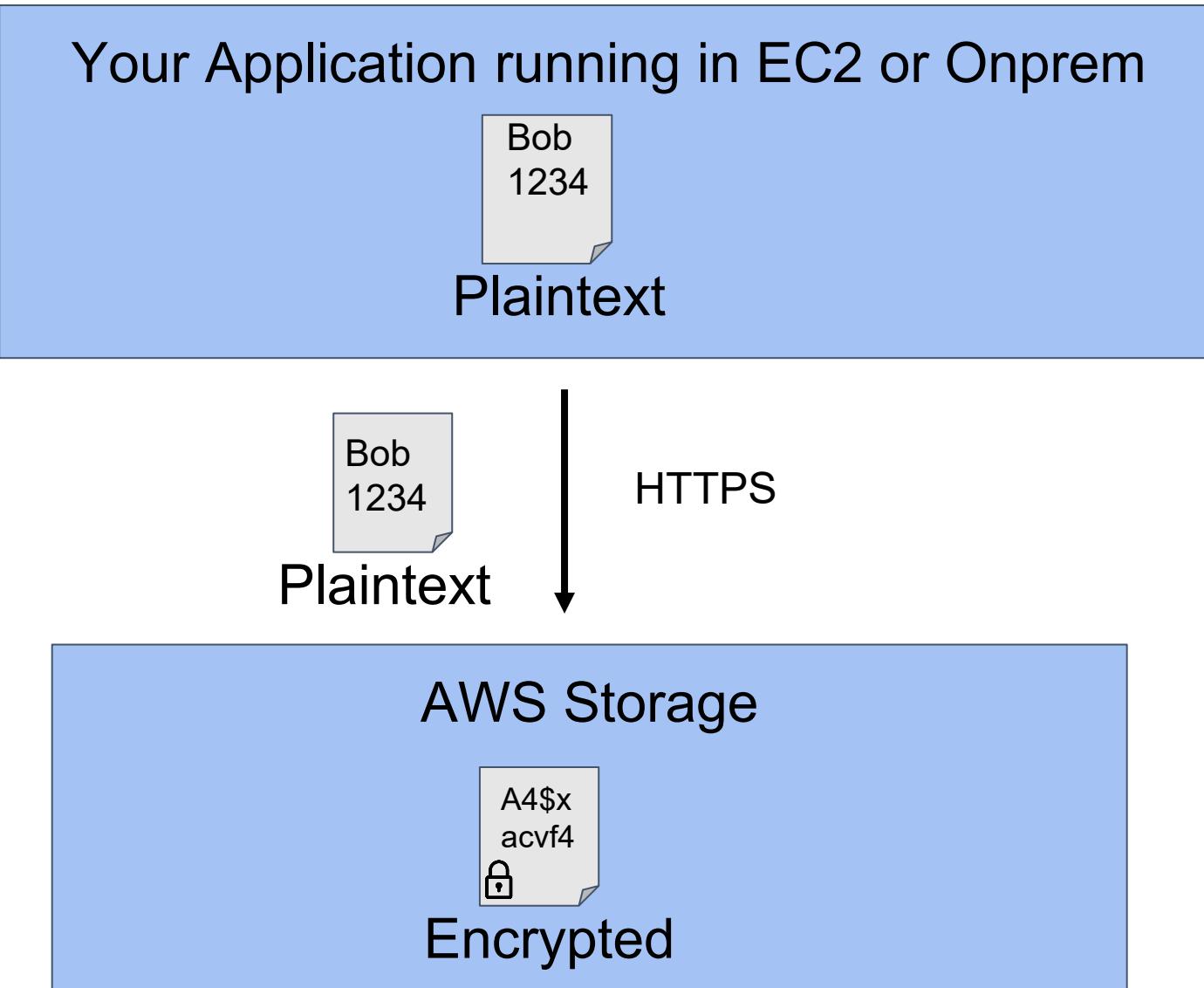


# Client Side Encryption

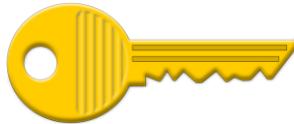


AWS Storage  
S3, EBS, RDS etc.

# Server Side Encryption



# Managing Key Yourselves

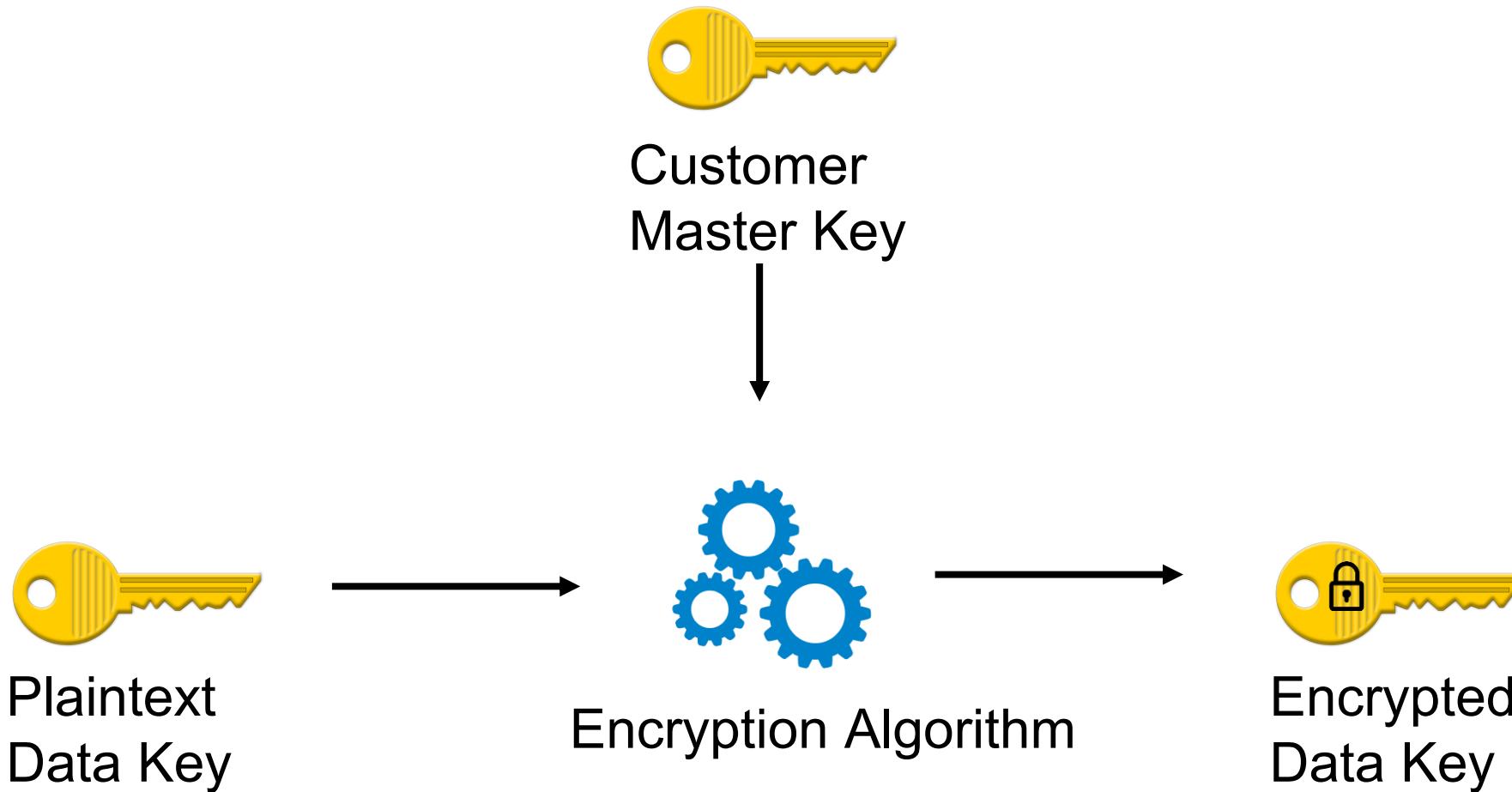


Encryption Key

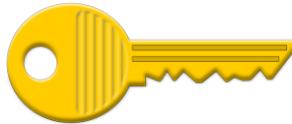
Keys need to be rotated periodically

Making it harder to obtain the key for intruders

# Envelope Encryption



# Managing Key Yourselves



Encryption Key

Keys need to be rotated periodically

Making it harder to obtain the key for intruders

Track and log your keys usage, detect anomaly

# AWS KMS (Key Management System)

Fully Managed

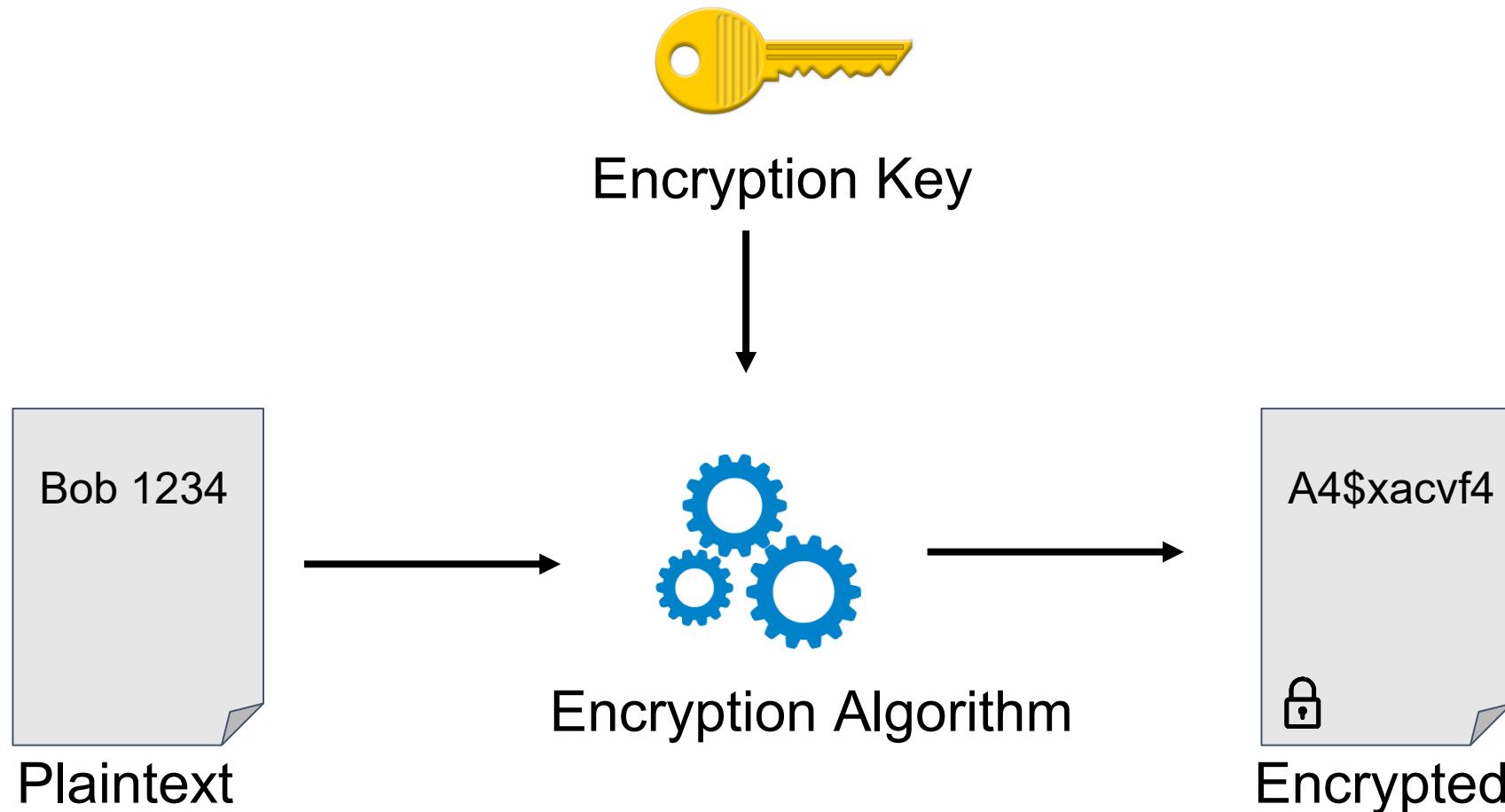
Centralized Key Management

Integration with AWS Services

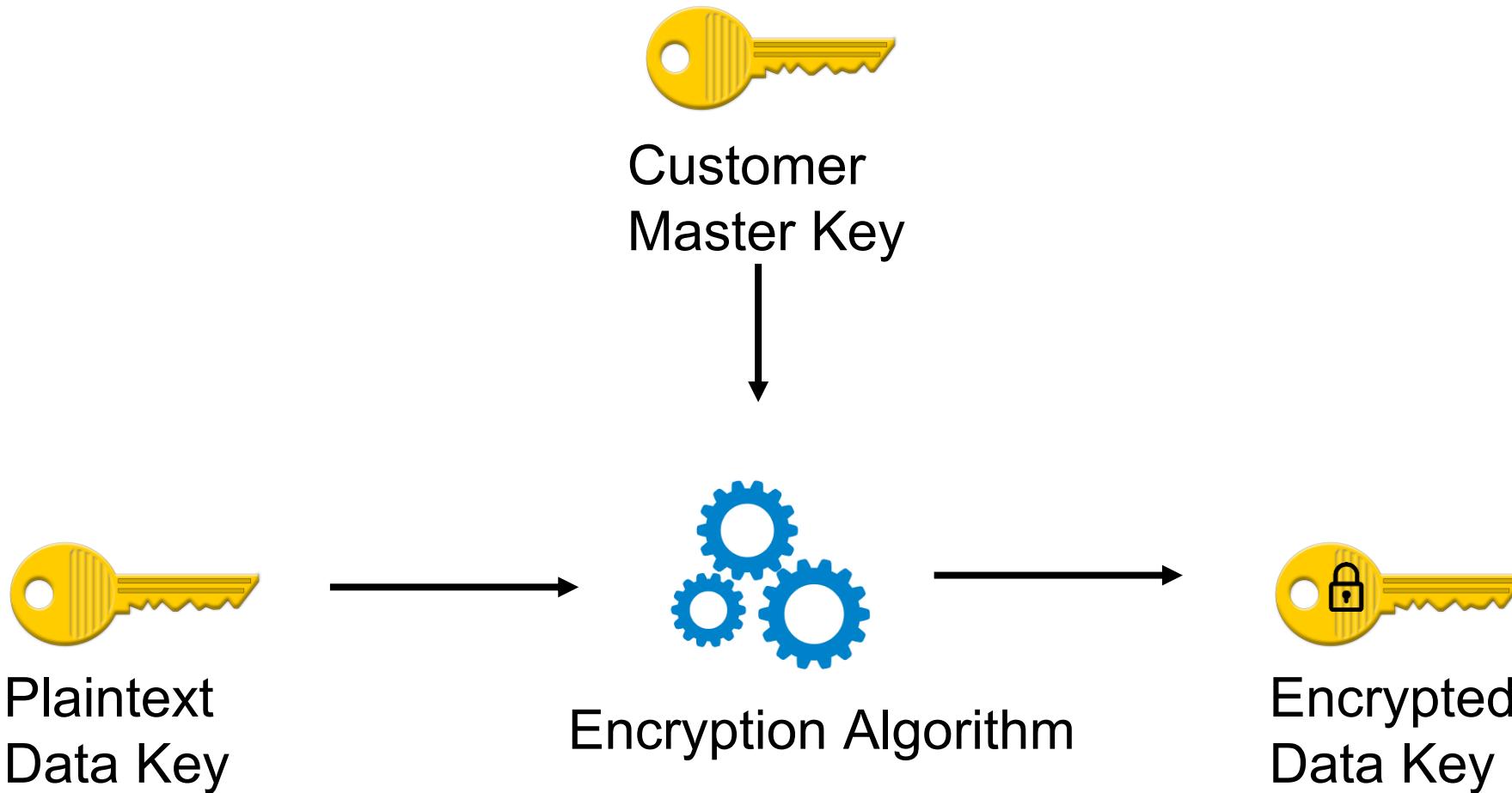
Built in Auditing

Secure and Compliant

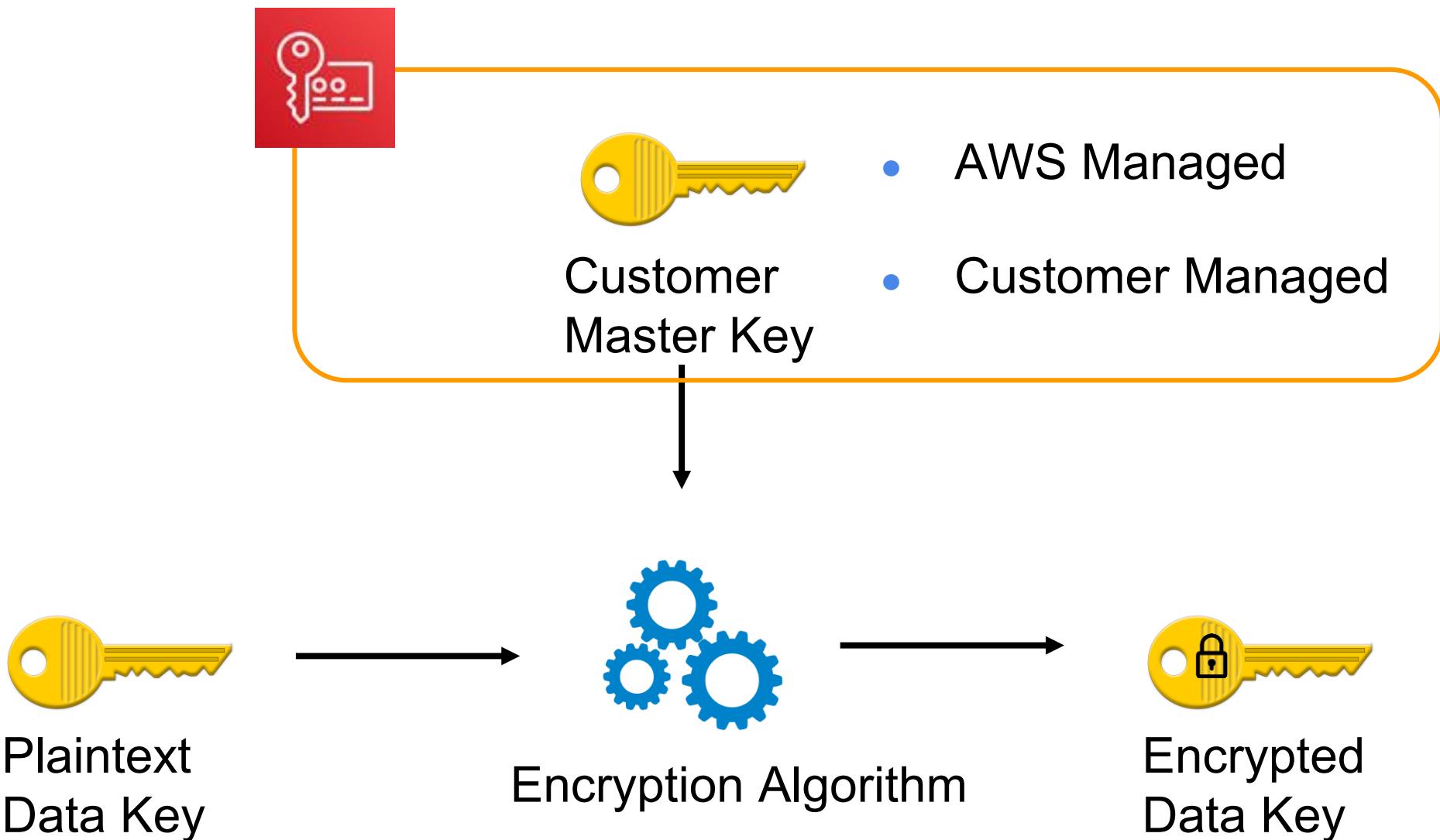
# Encryption Flow



# Envelope Encryption



## AWS KMS



# AWS Managed CMK

Identified by  
*aws/servicename*

AWS generated

Can't be deleted

Can't be baked into  
custom roles

Rotated once every 3  
years automatically

# Customer Managed CMK

Can be given any name

Customer created

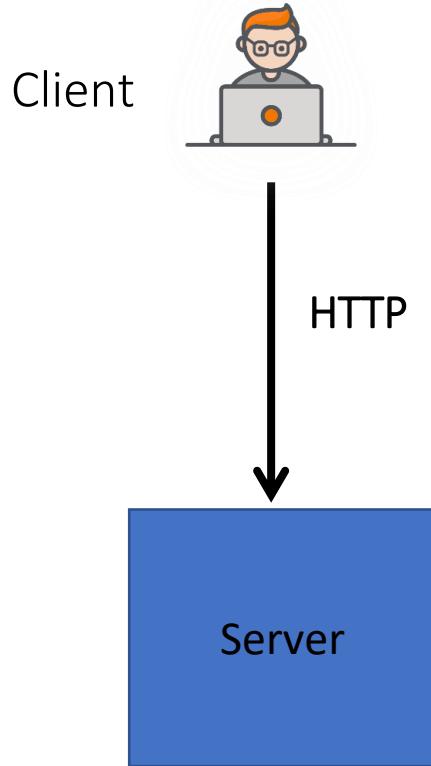
Can be  
deleted/enabled/disabled

Can be baked into custom  
roles

Rotated once a year  
automatically or manually

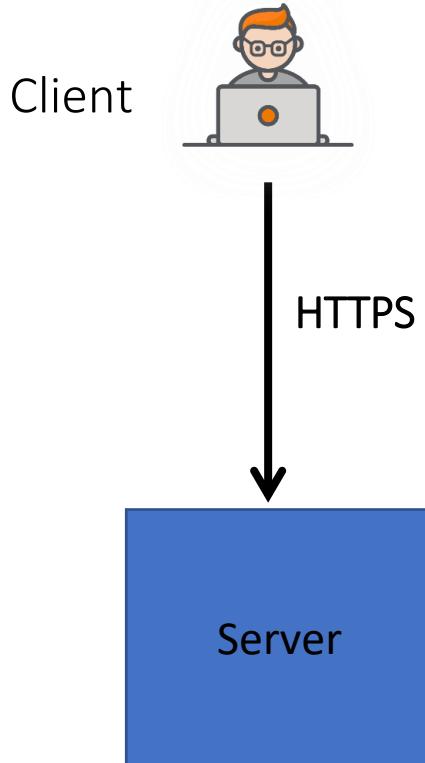
# Security at Transit

# Data in Transit



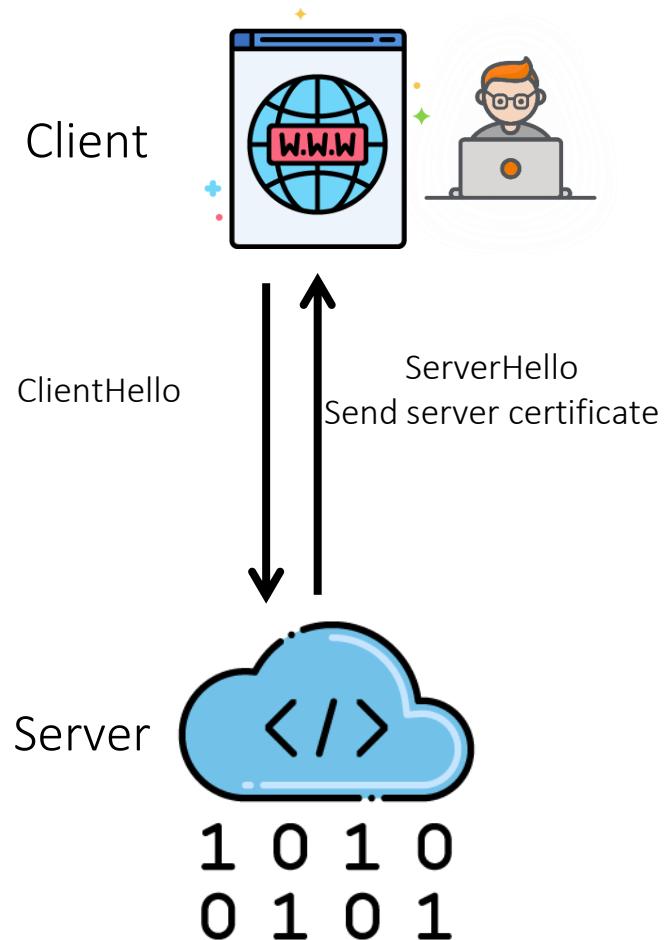
- Hyper Text Transfer Protocol
- All information is sent in clear text
- Vulnerable to attack
- Not used in real world systems

# Data in Transit

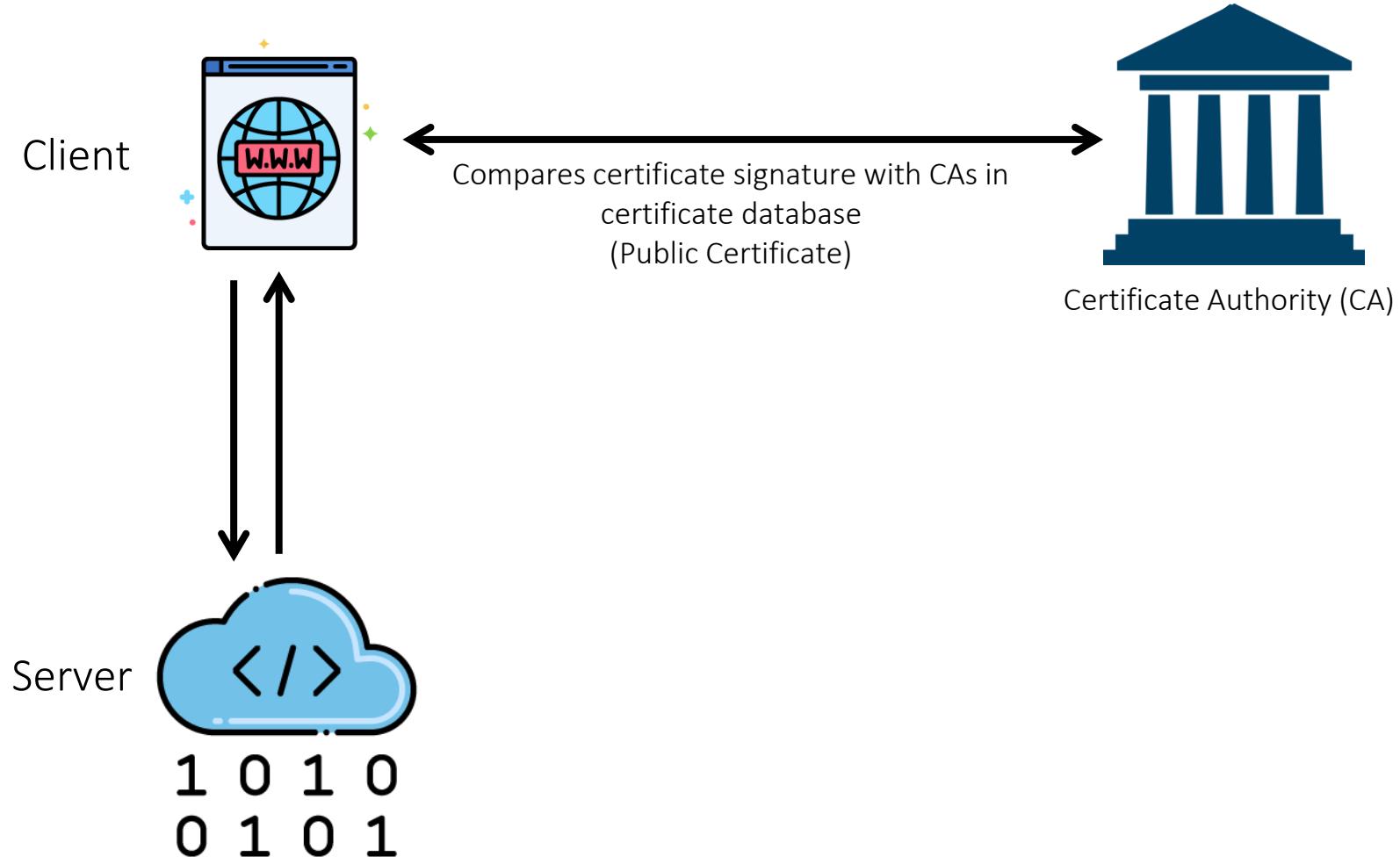


- Hyper Text Transfer Protocol Secure
- All information is encrypted
- Uses one of the two protocols:
  - SSL (Secure Socket Layer)
  - TLS (Transport Layer Security)
    - MTLS (Mutual TLS)
  - TLS is faster, newer, and built on SSL

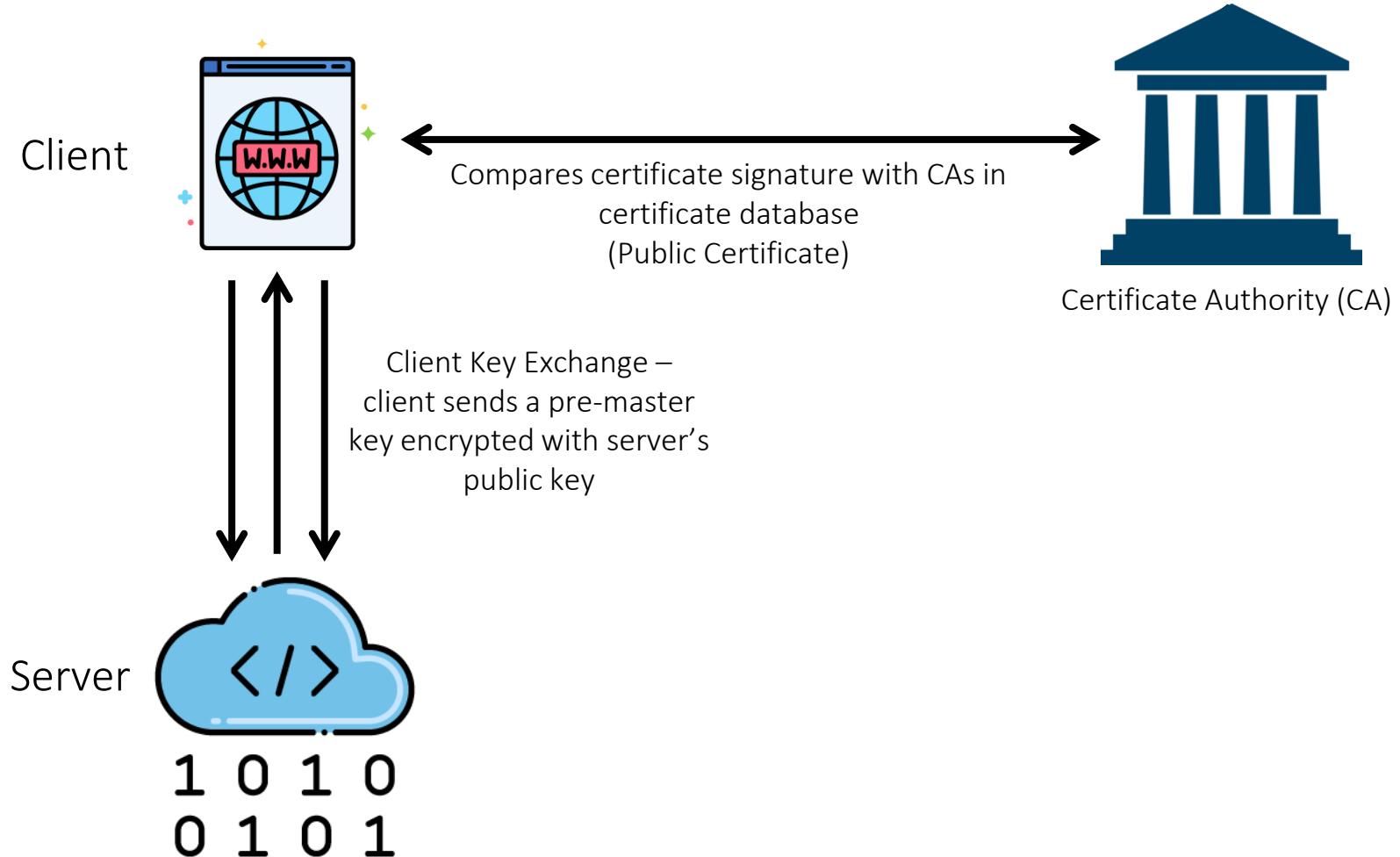
# SSL/TLS Flow



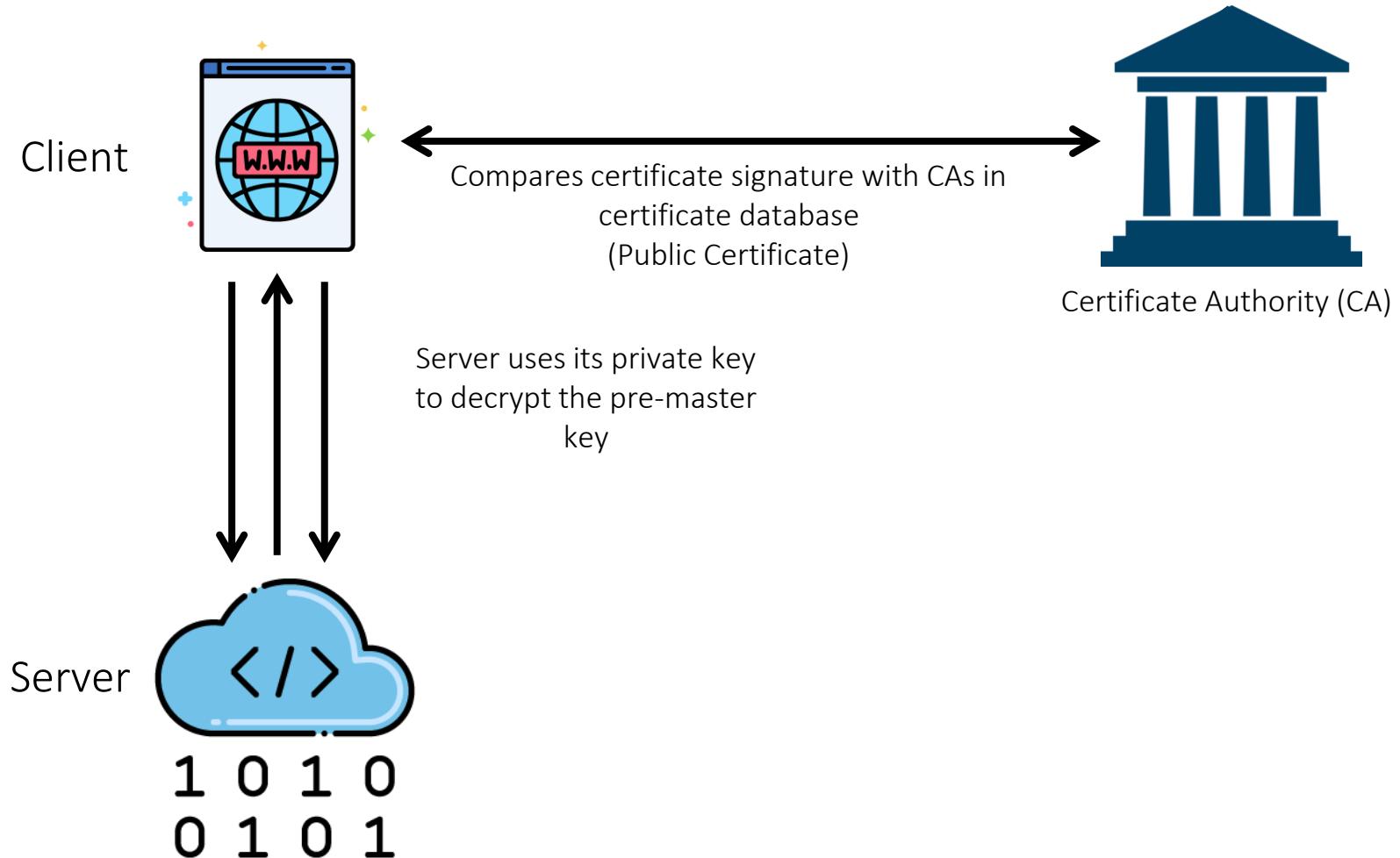
# SSL/TLS Flow



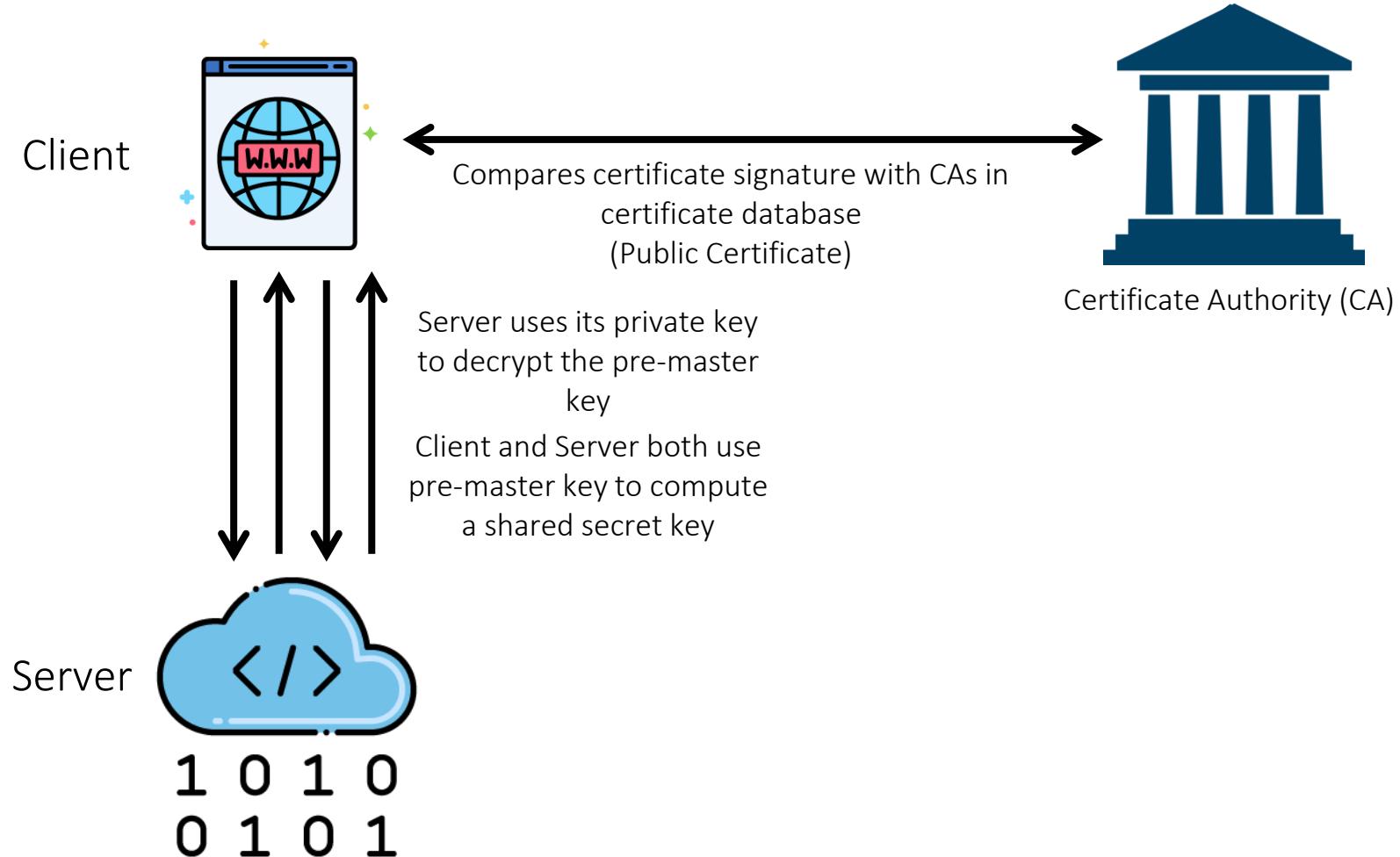
# SSL/TLS Flow



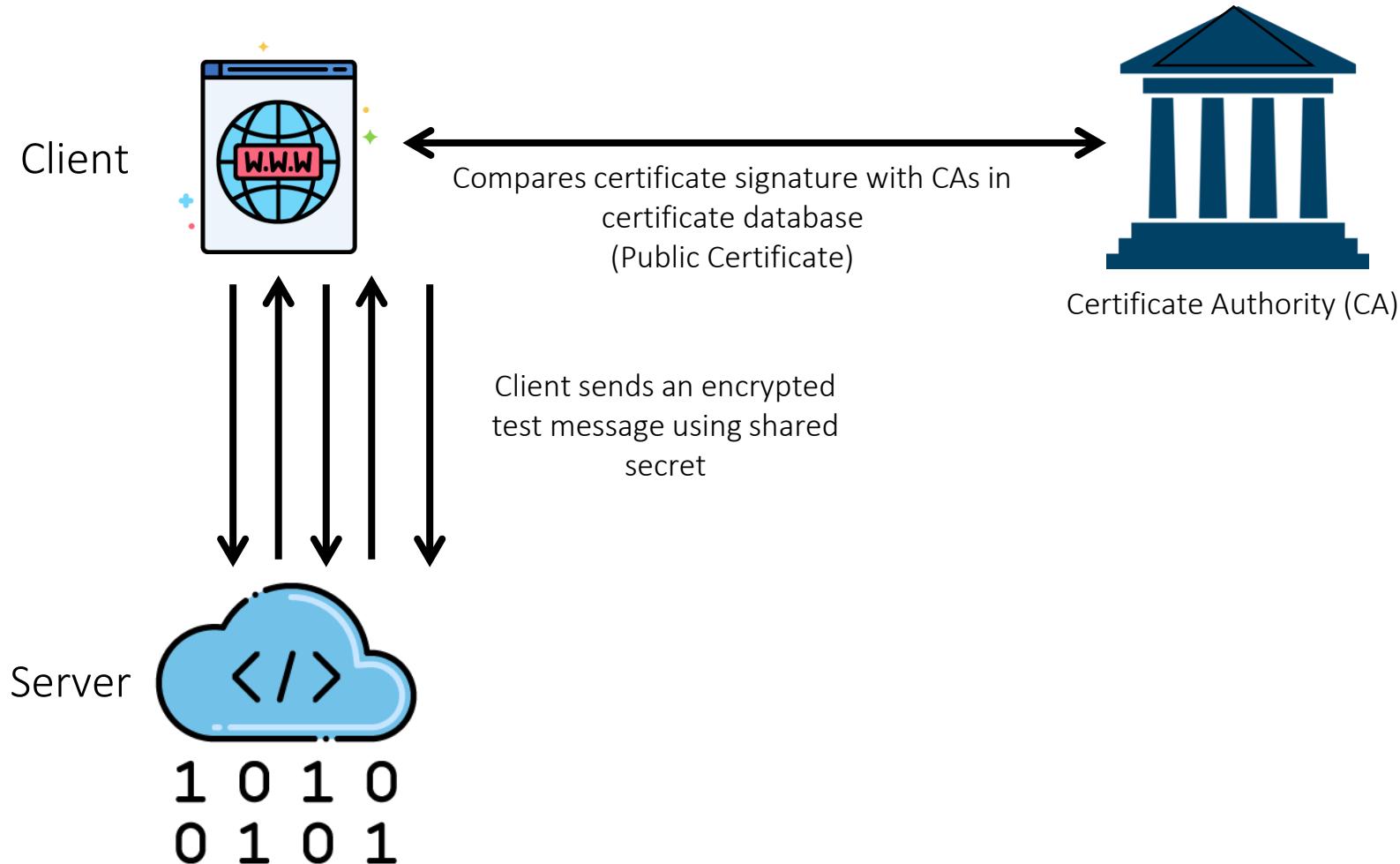
# SSL/TLS Flow



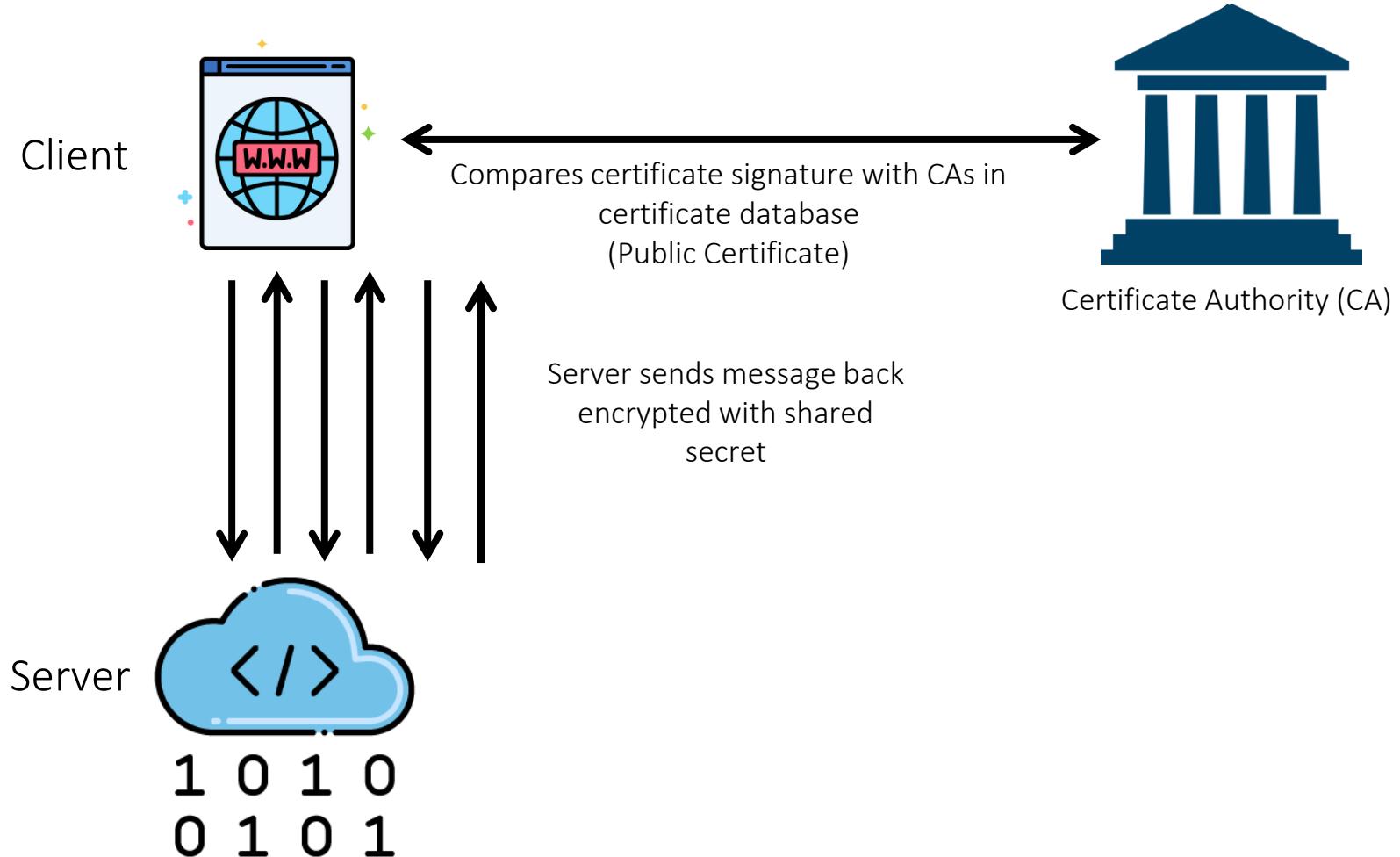
# SSL/TLS Flow



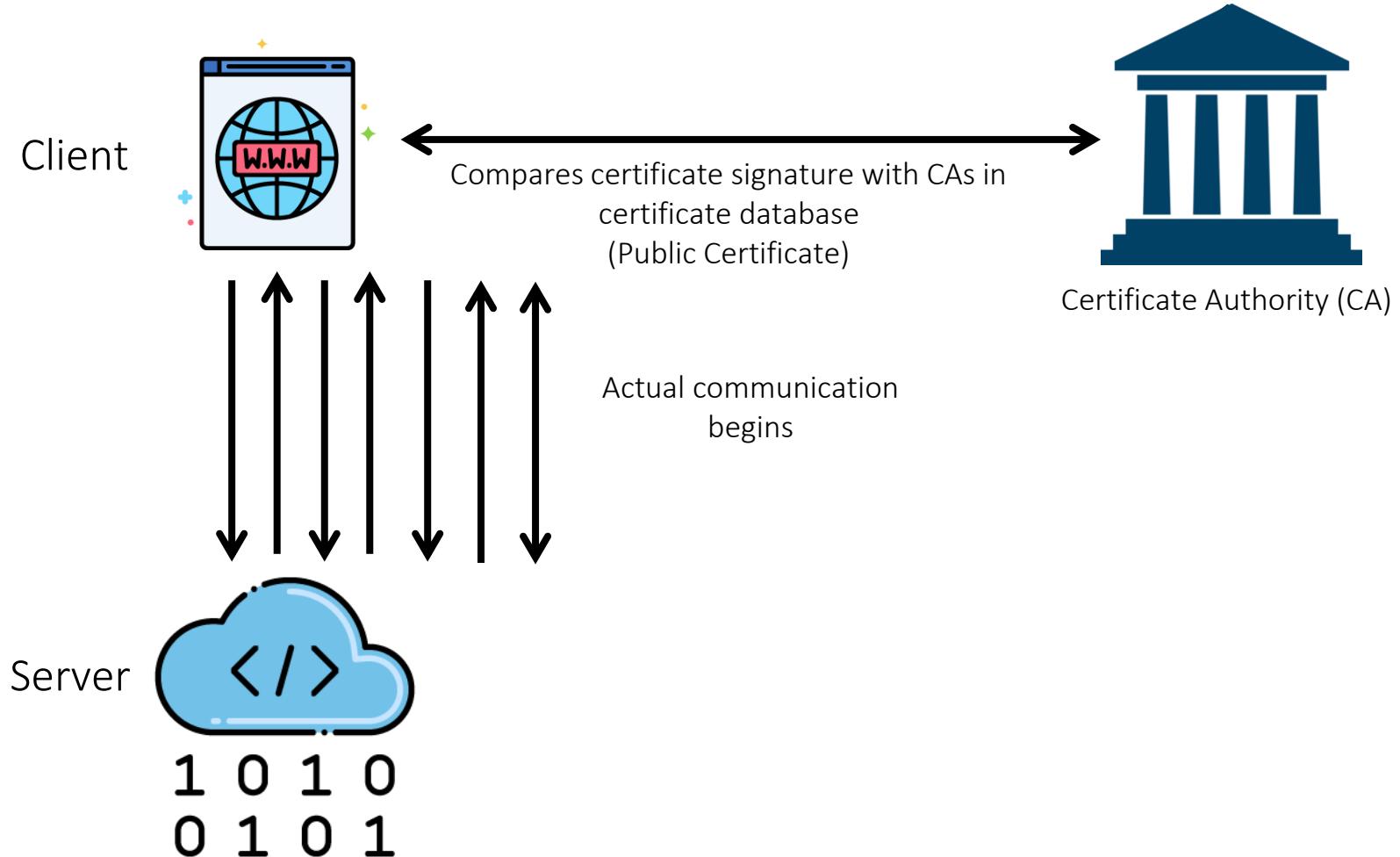
# SSL/TLS Flow



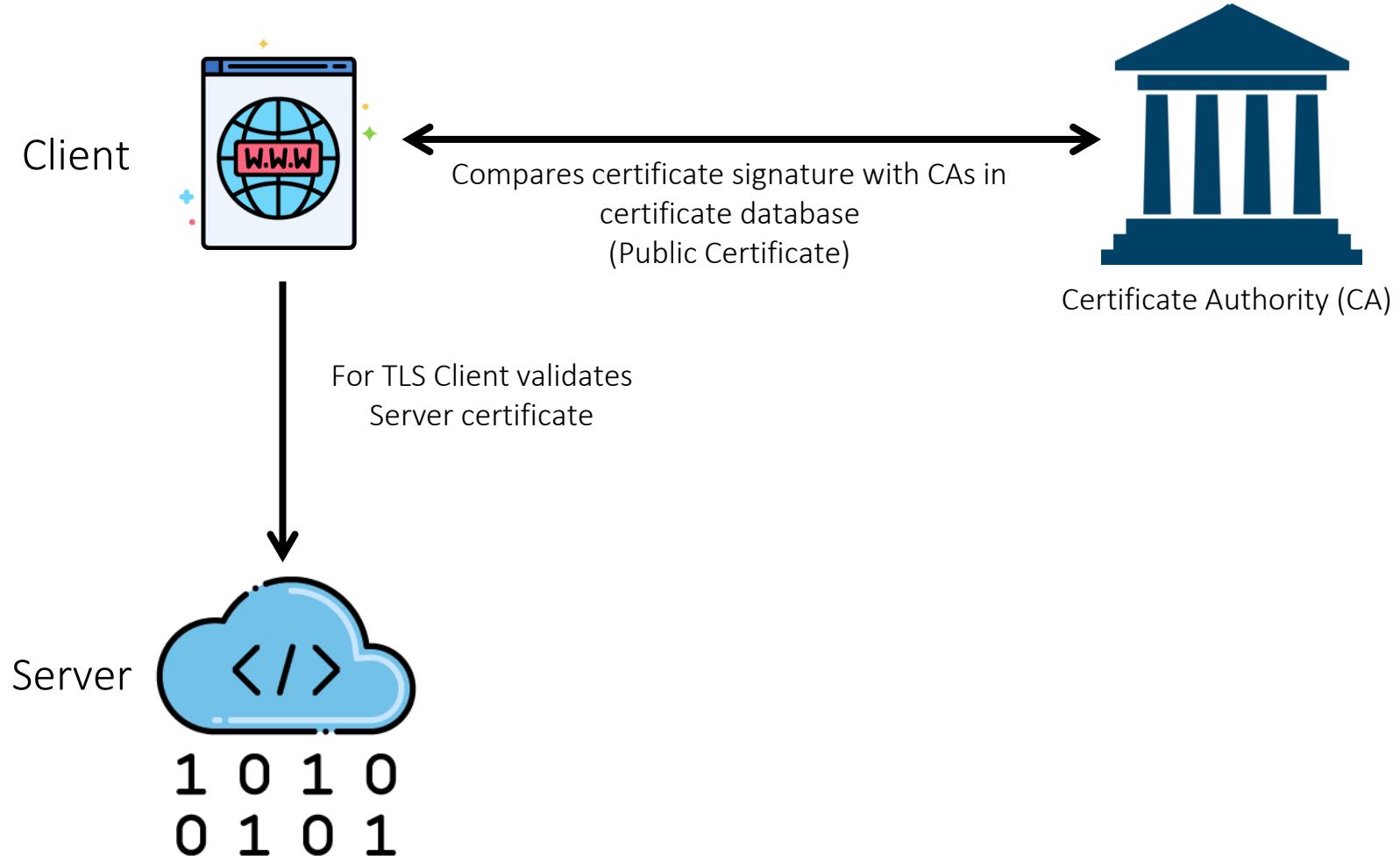
# SSL/TLS Flow



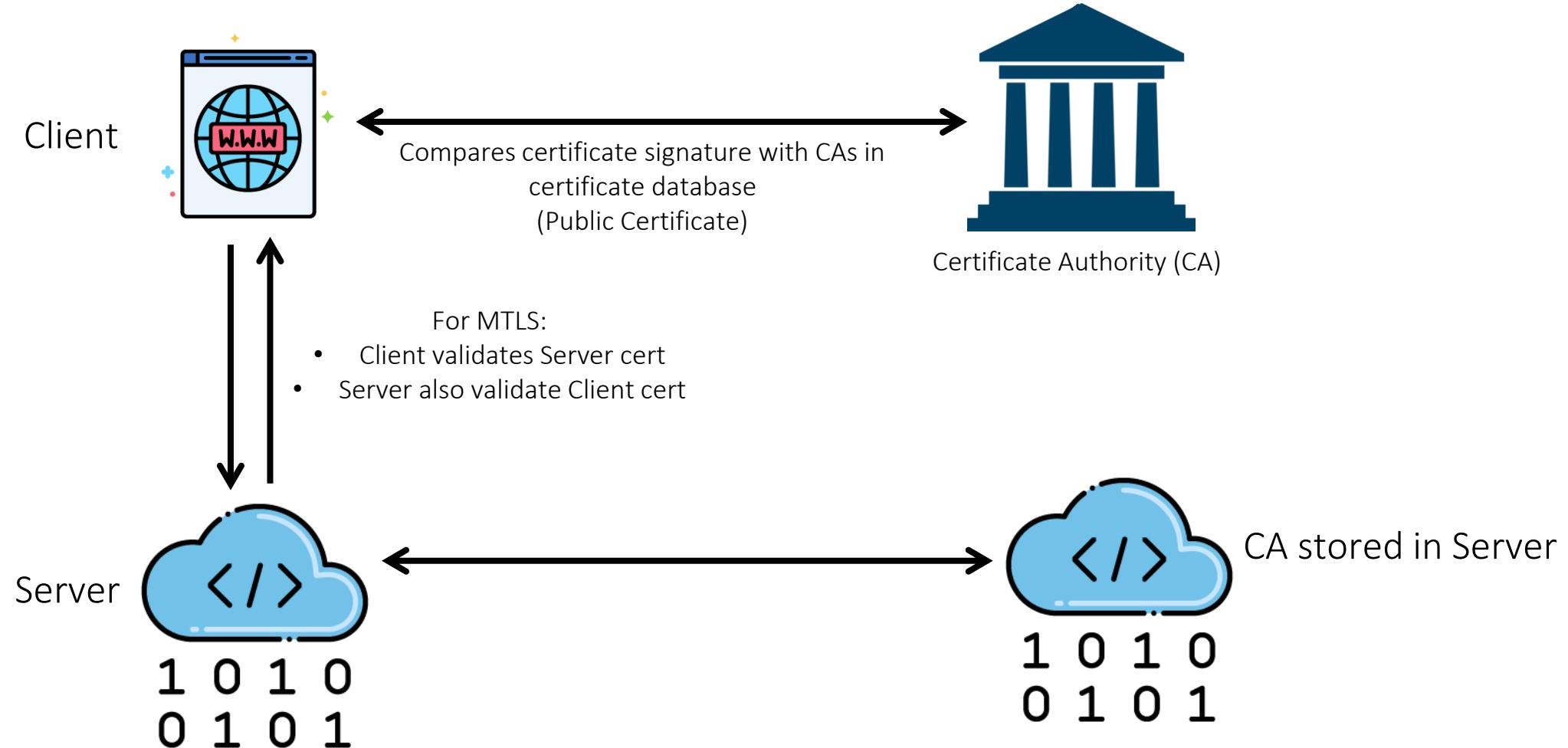
# SSL/TLS Flow



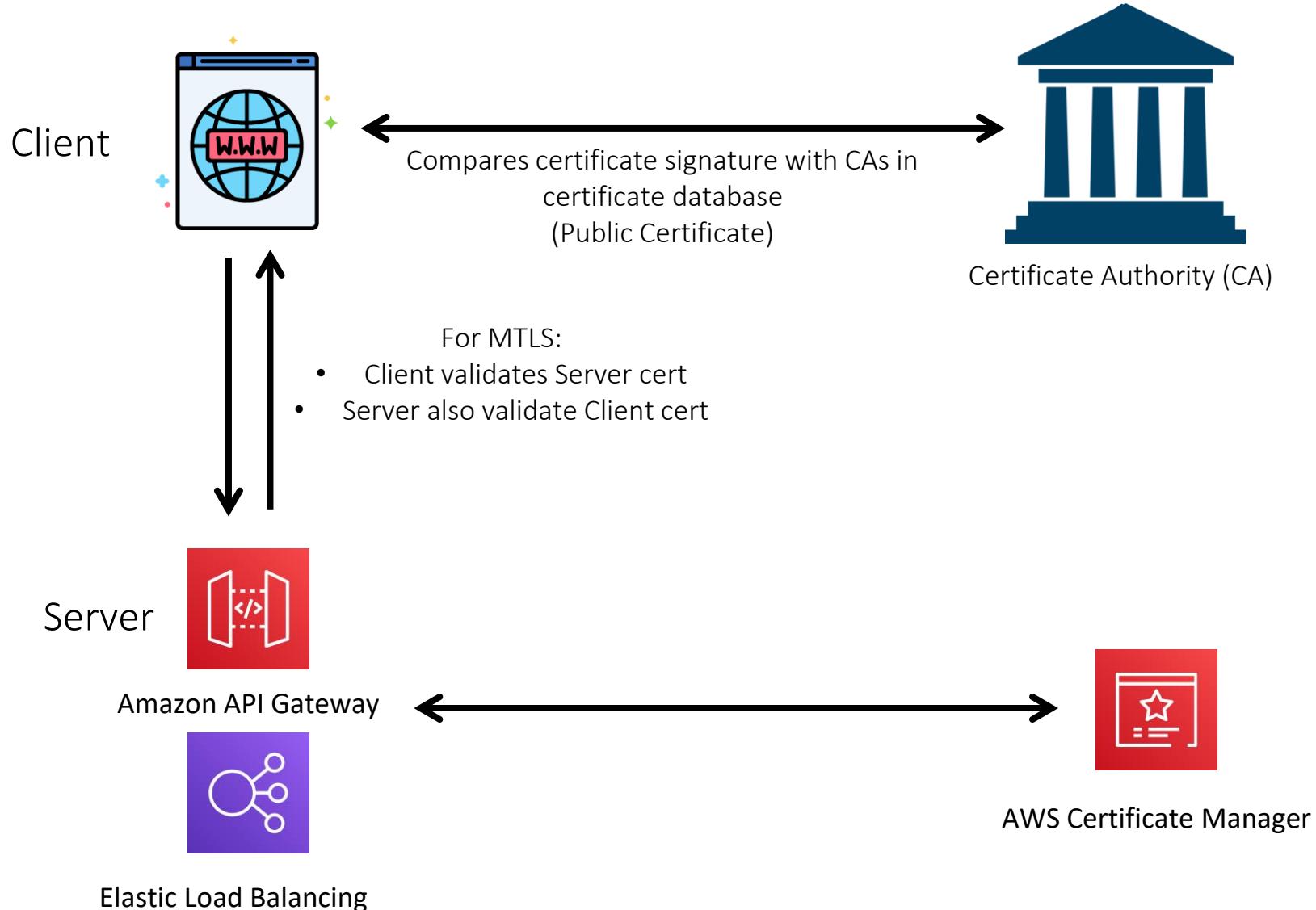
# Mutual TLS



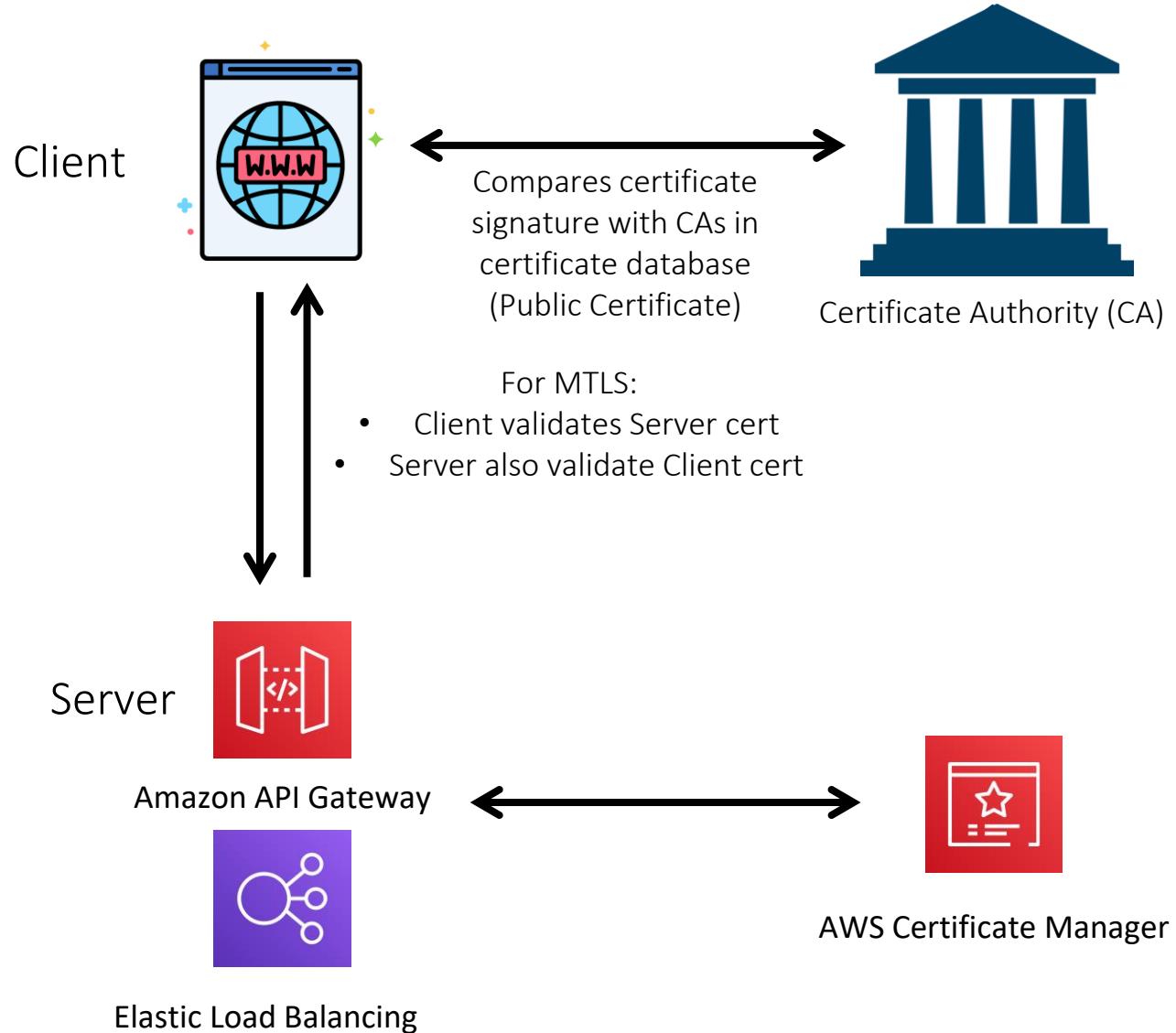
# Mutual TLS



# Mutual TLS in AWS



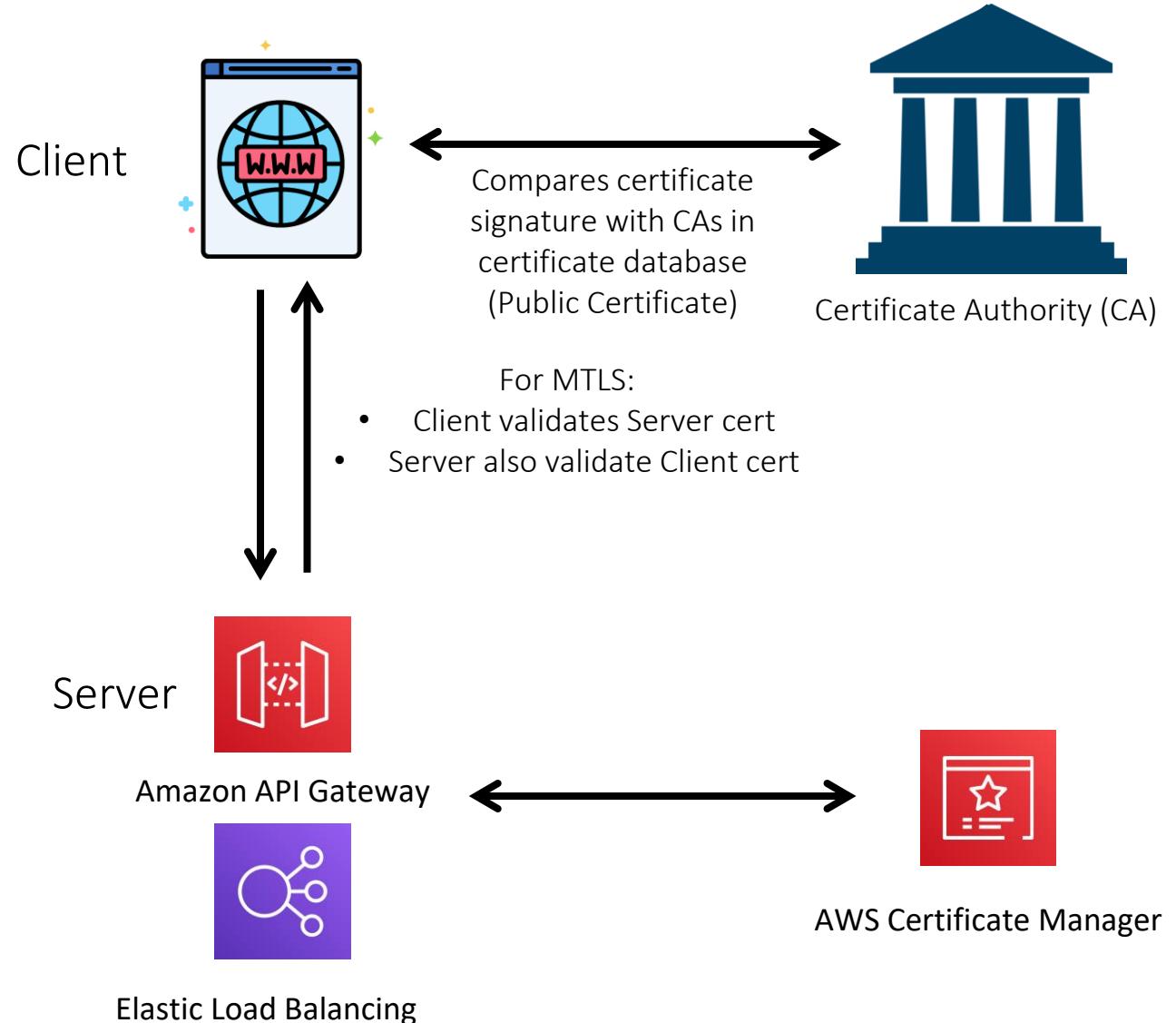
# Mutual TLS in AWS



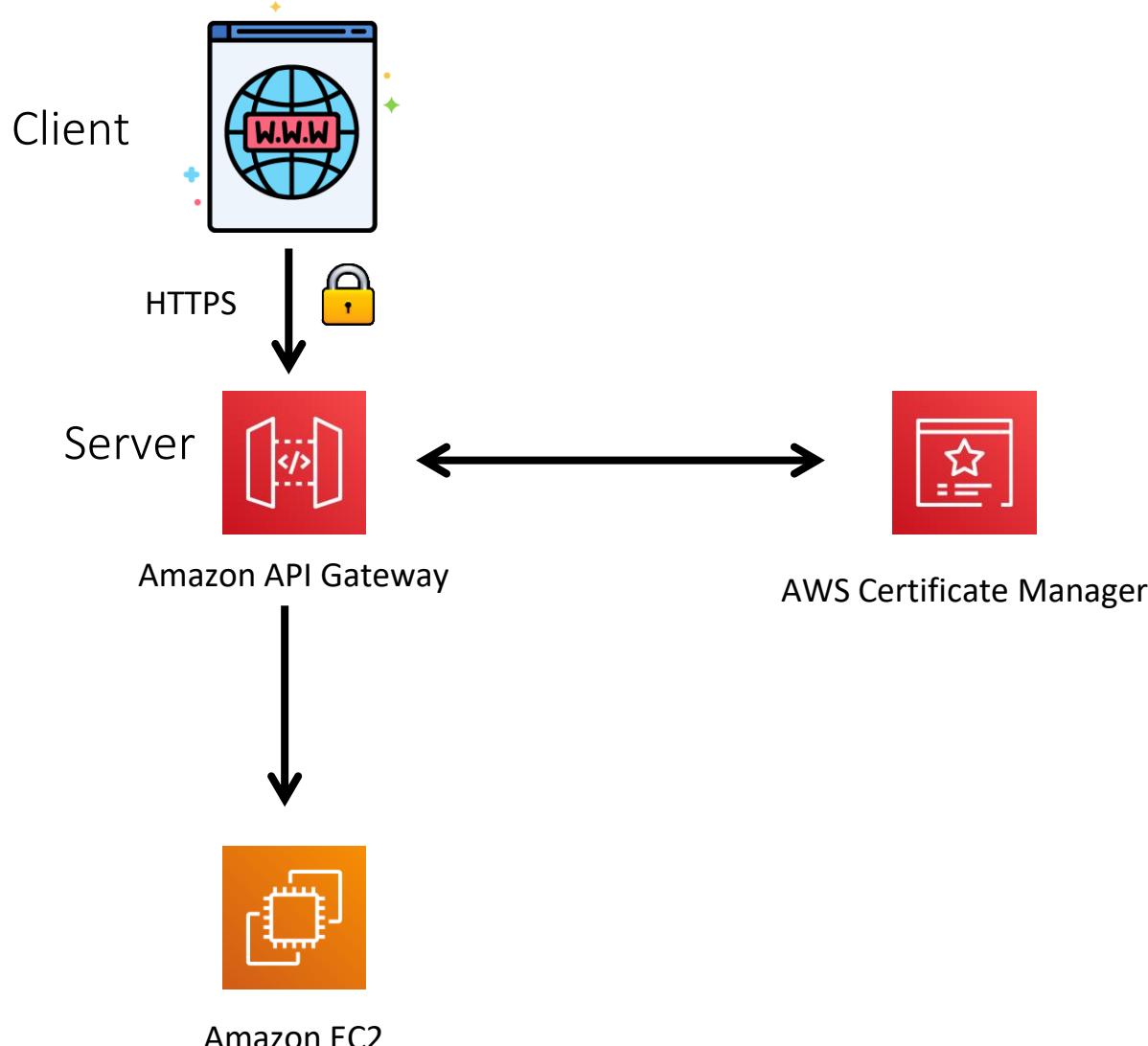
- MTLS is used for B2B
- TLS is used with thin clients (web browsers)

# Real World TLS with ALB, NLB, API Gateway

# Mutual TLS in AWS

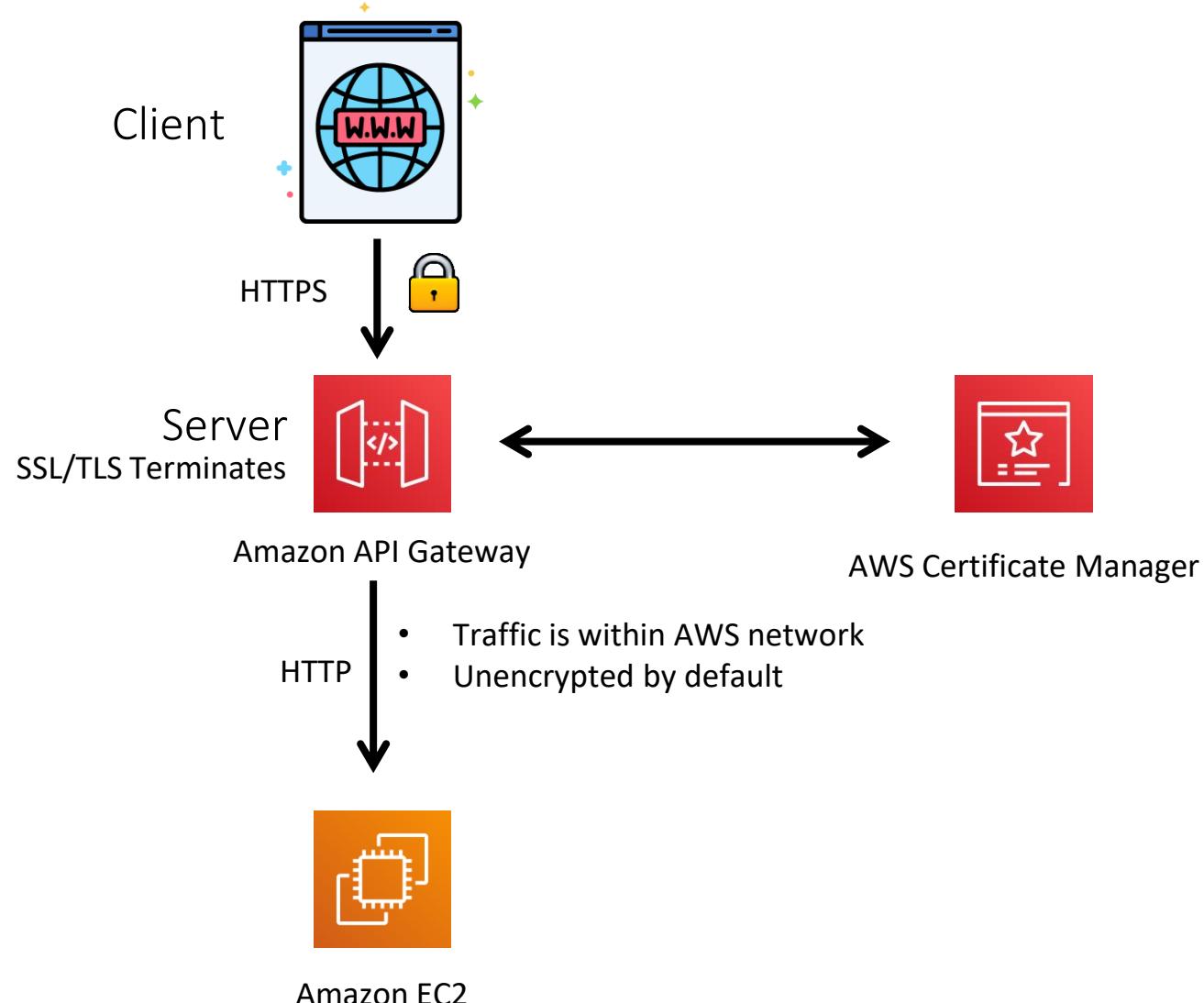


# TLS with API Gateway

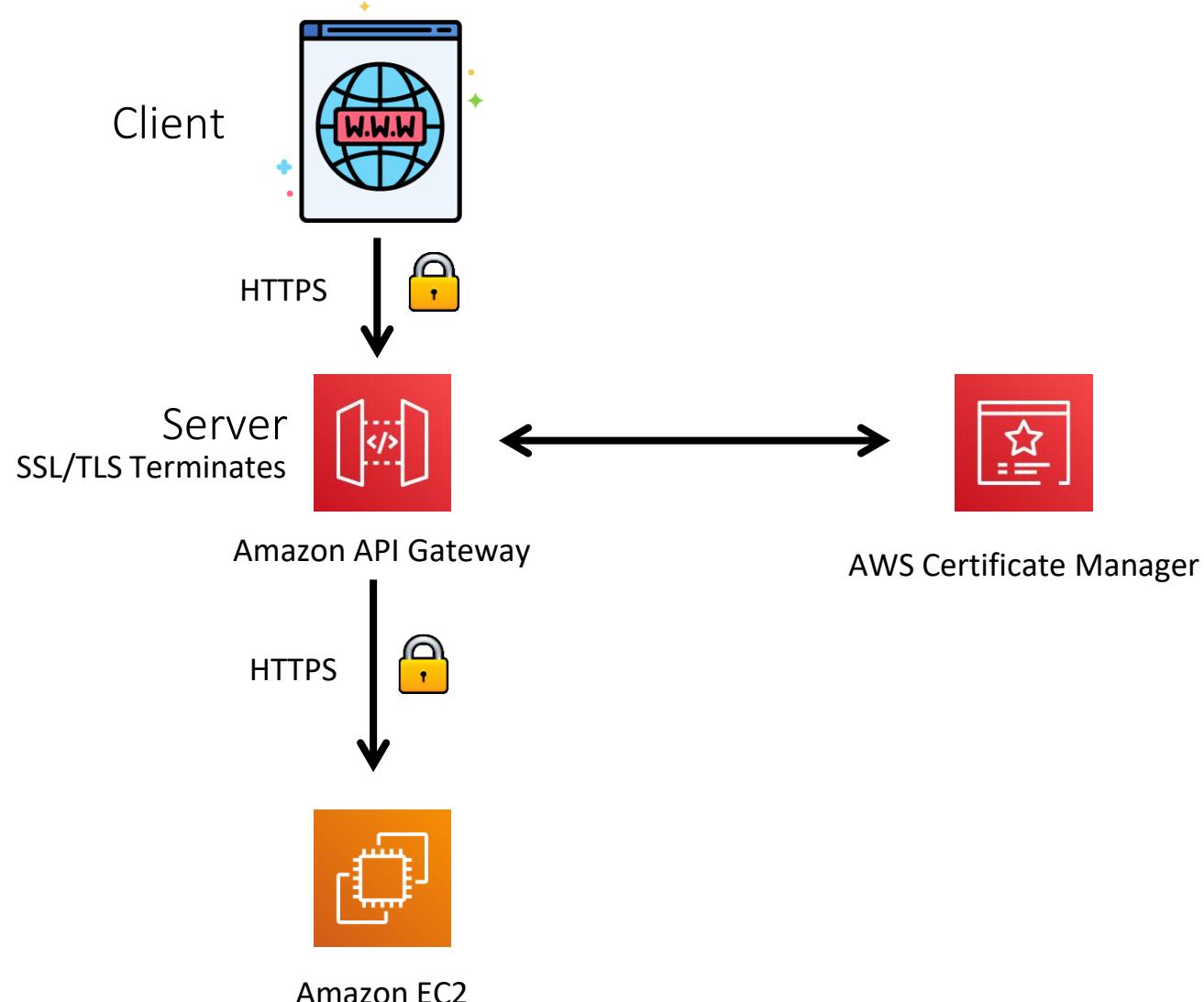


- API Gateway must have HTTPS
- By default, API Gateway will use AWS default cert
- With custom domain, you can bring in your own cert

# TLS with API Gateway

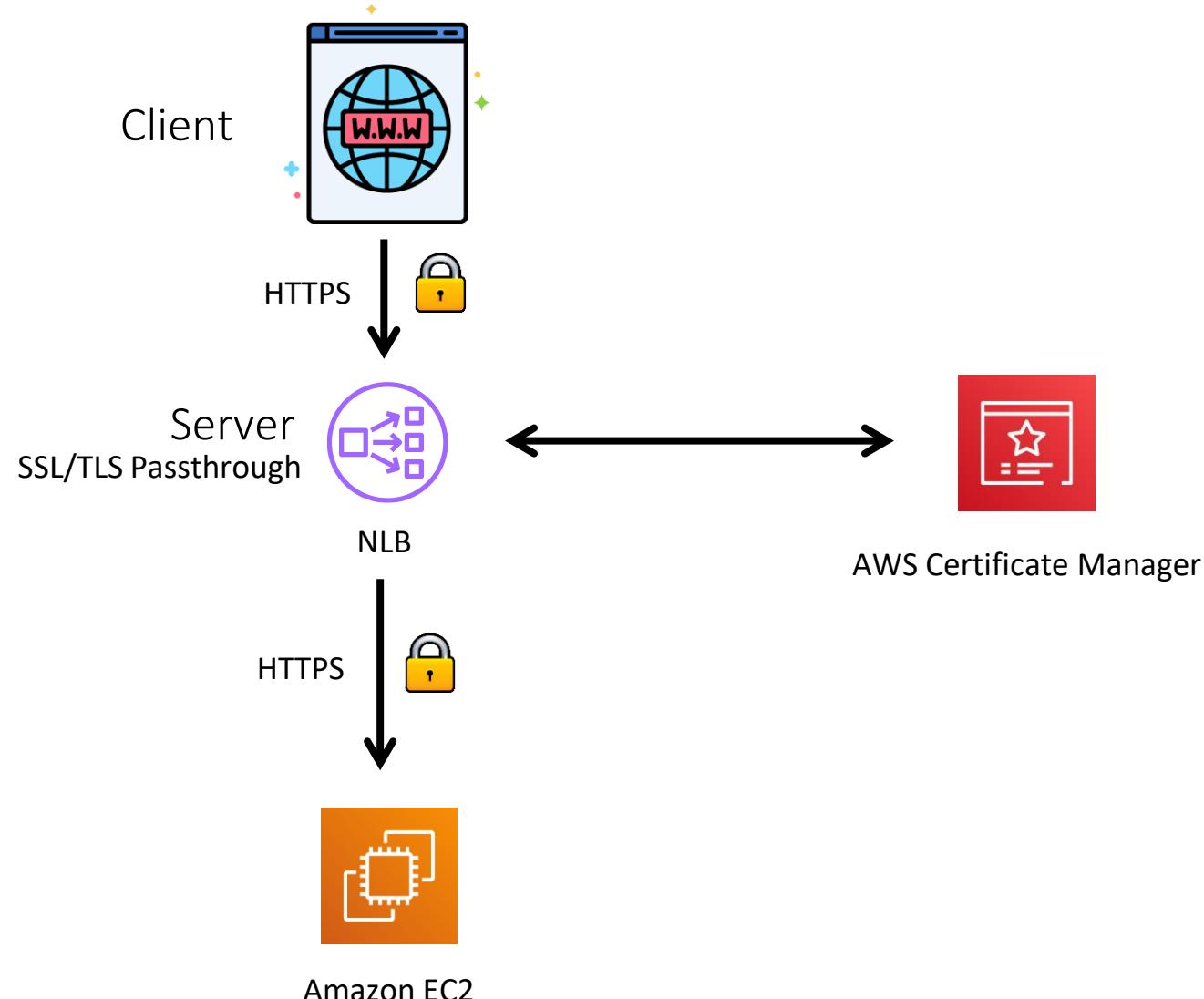


# TLS with API Gateway



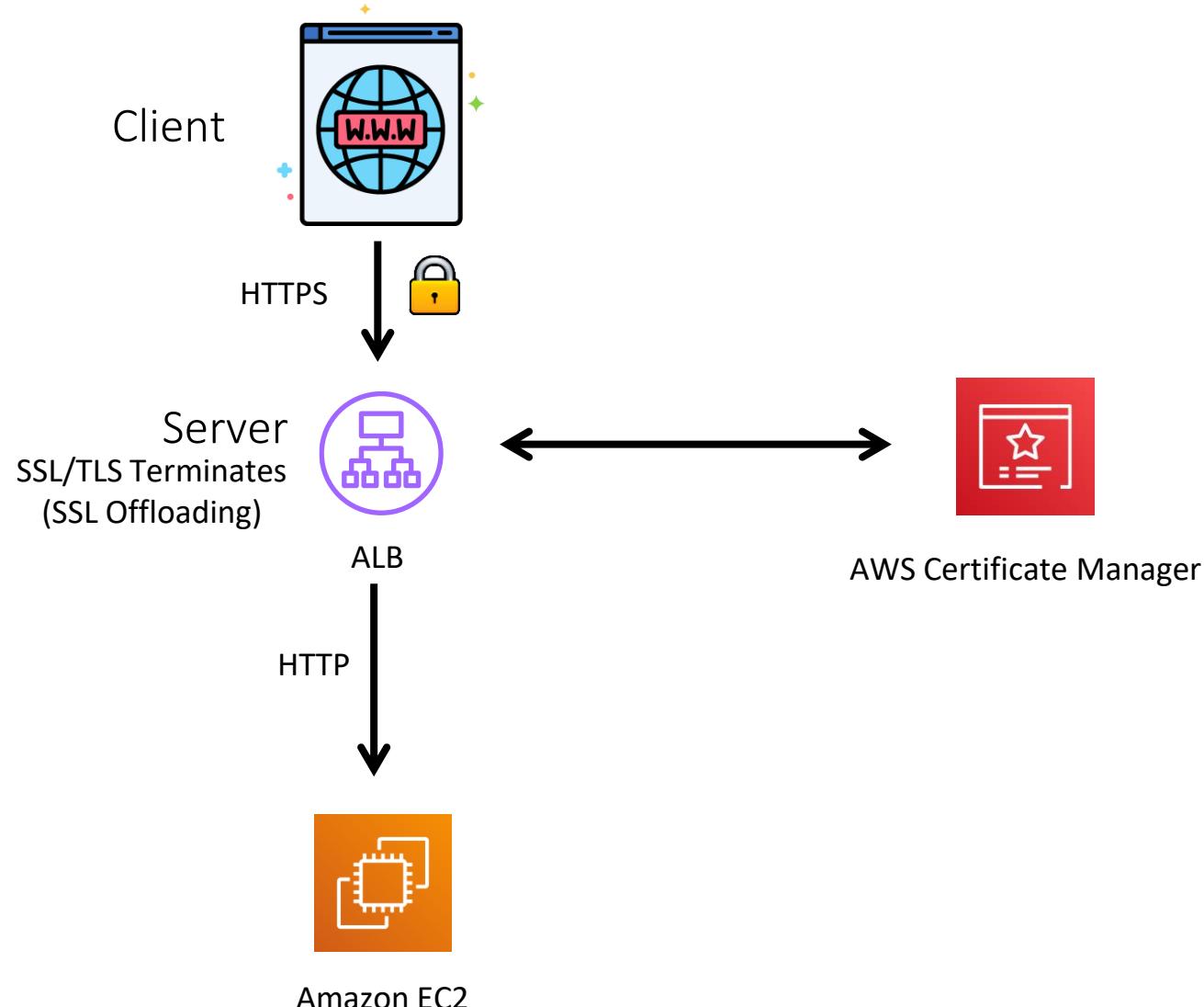
- API Gateway can create an SSL cert for backend
- Backend server needs to validate the cert

# TLS with NLB



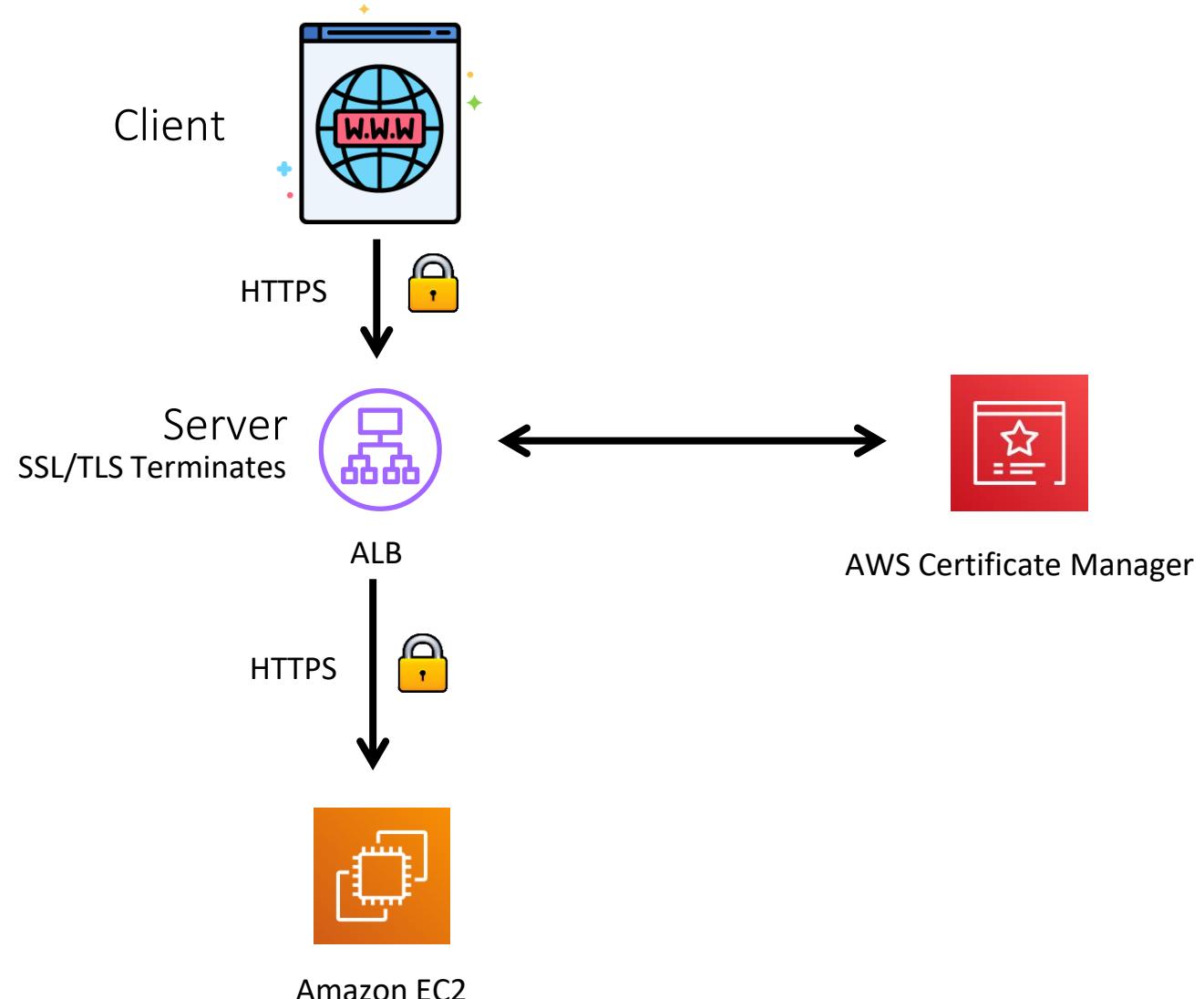
- SSL/TLS can NOT be terminated at NLB
- Backend server need to validate cert

# TLS with ALB



- ALB can accept either HTTP or HTTPS traffic from client
- SSL/TLS will terminate at ALB
  - Backend traffic within AWS network

# TLS with ALB

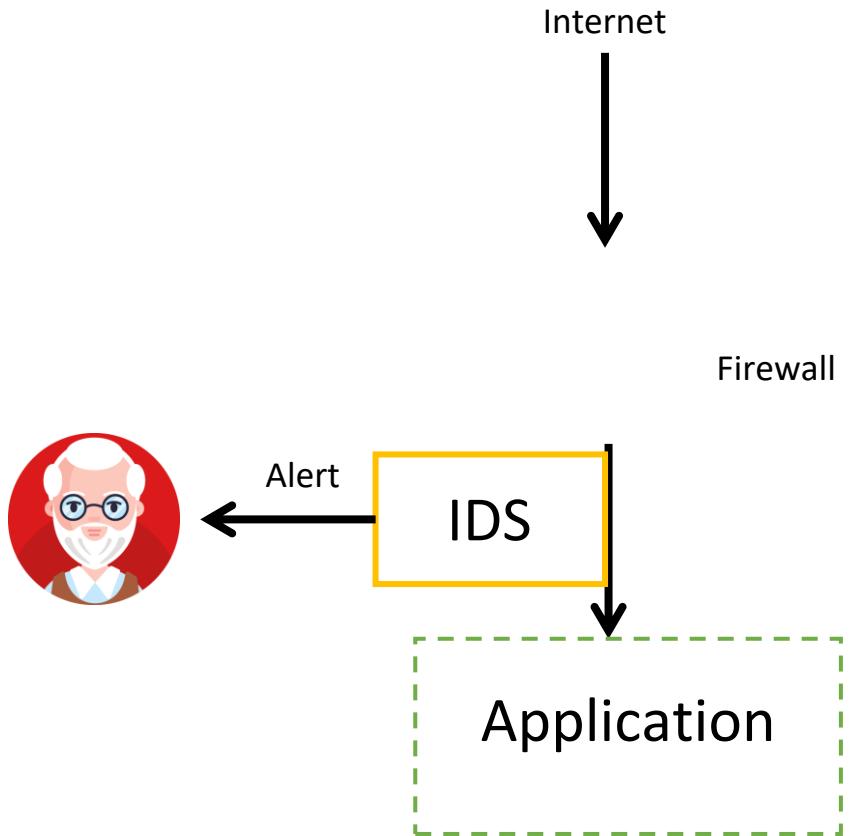


- ALB can have HTTPS to server if required

# IDS/IPS

# Intrusion Detection/Prevention System

# IDS

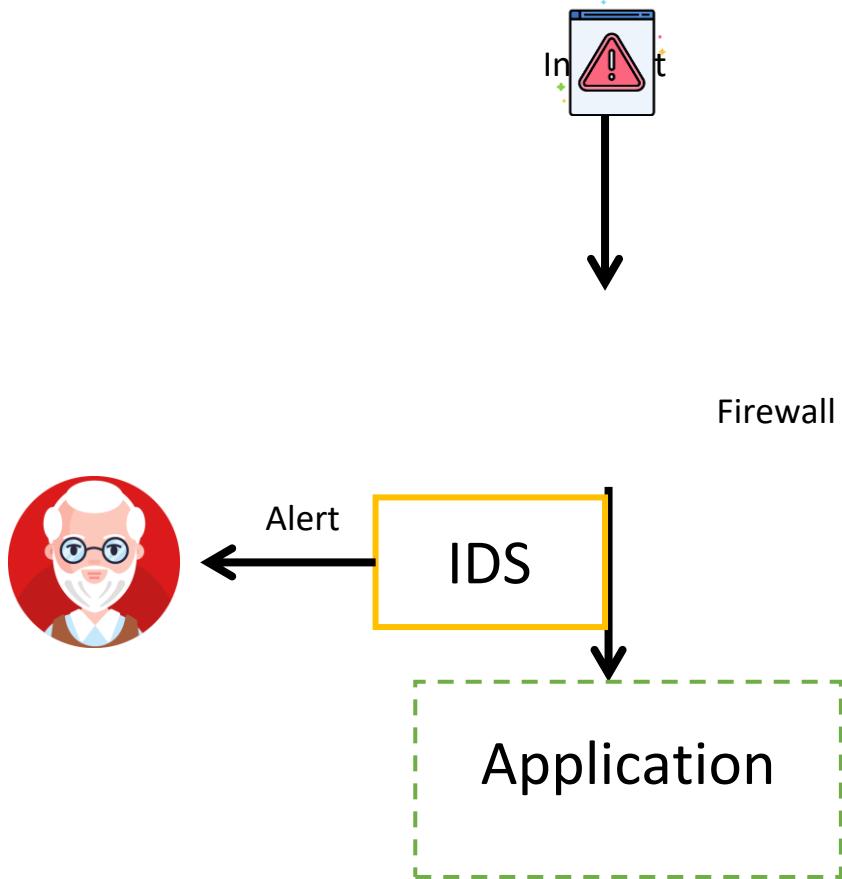


# IPS

- Scans L3-L7 traffic
- Detects and sends alerts

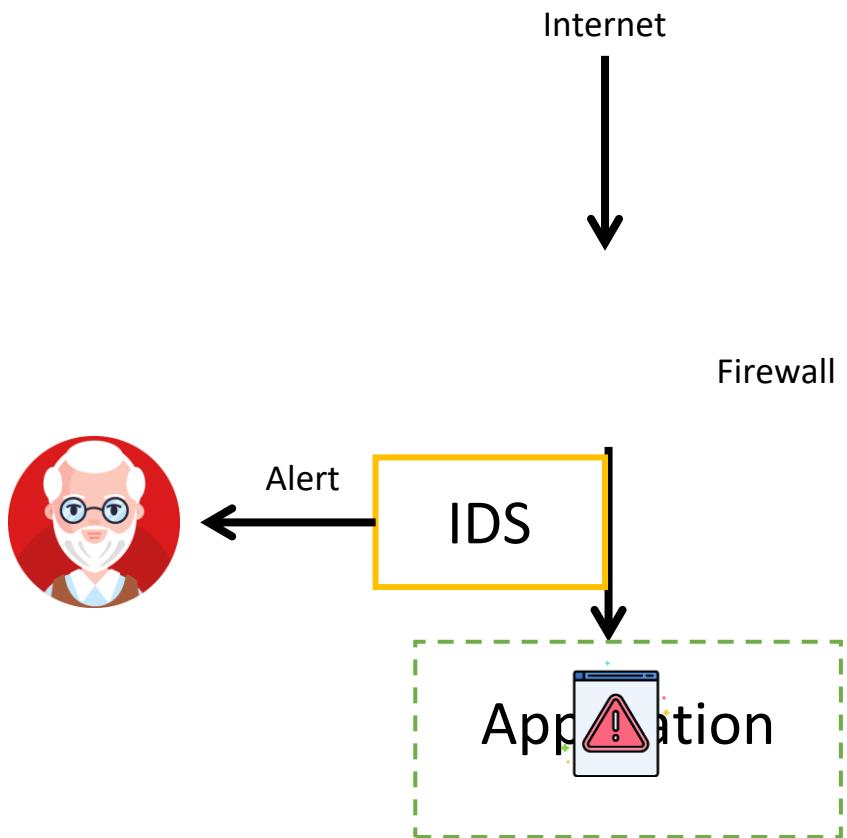
# IDS

# IPS



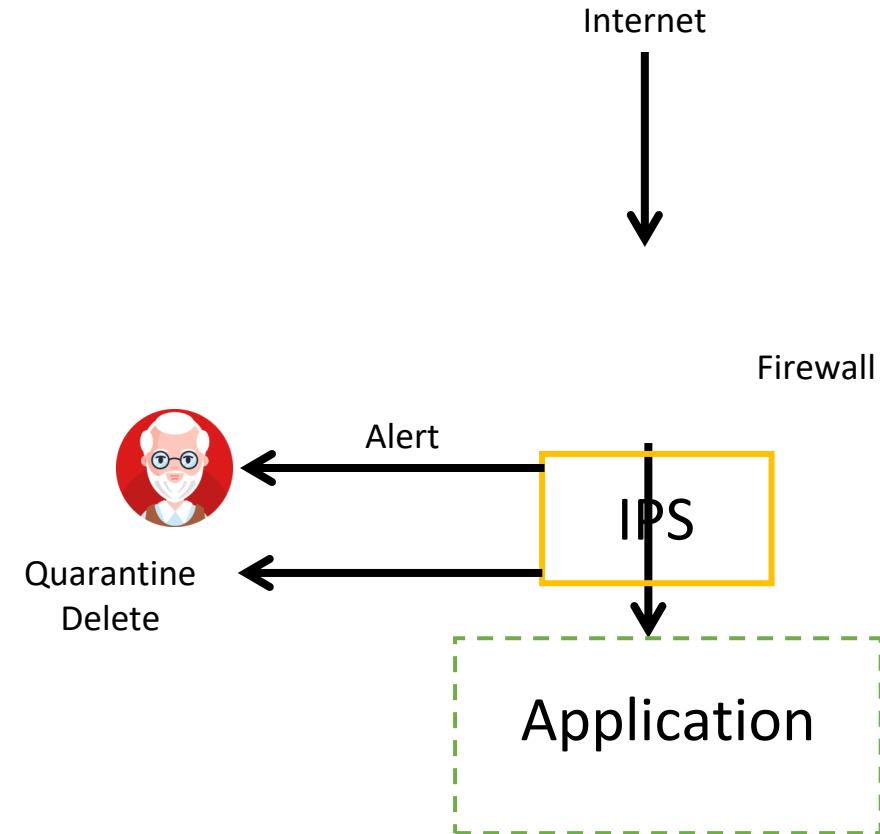
- Scans L3-L7 traffic
- Detects and sends alerts
- Does NOT prevent the traffic

# IDS



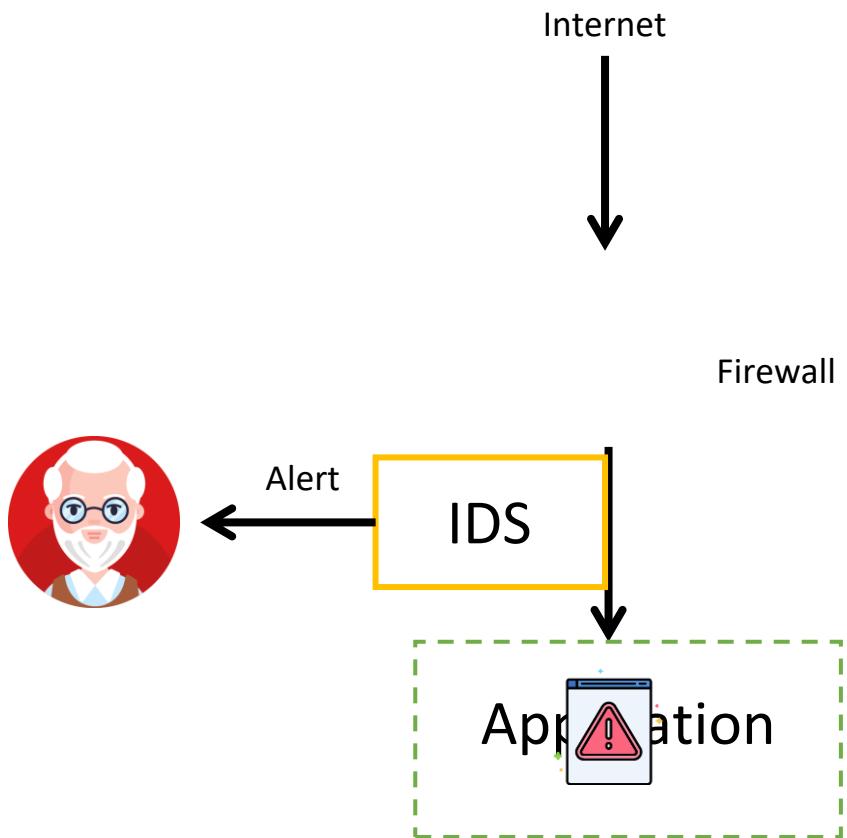
- Scans L3-L7 traffic
- Detects and sends alerts
- Does NOT prevent the traffic

# IPS



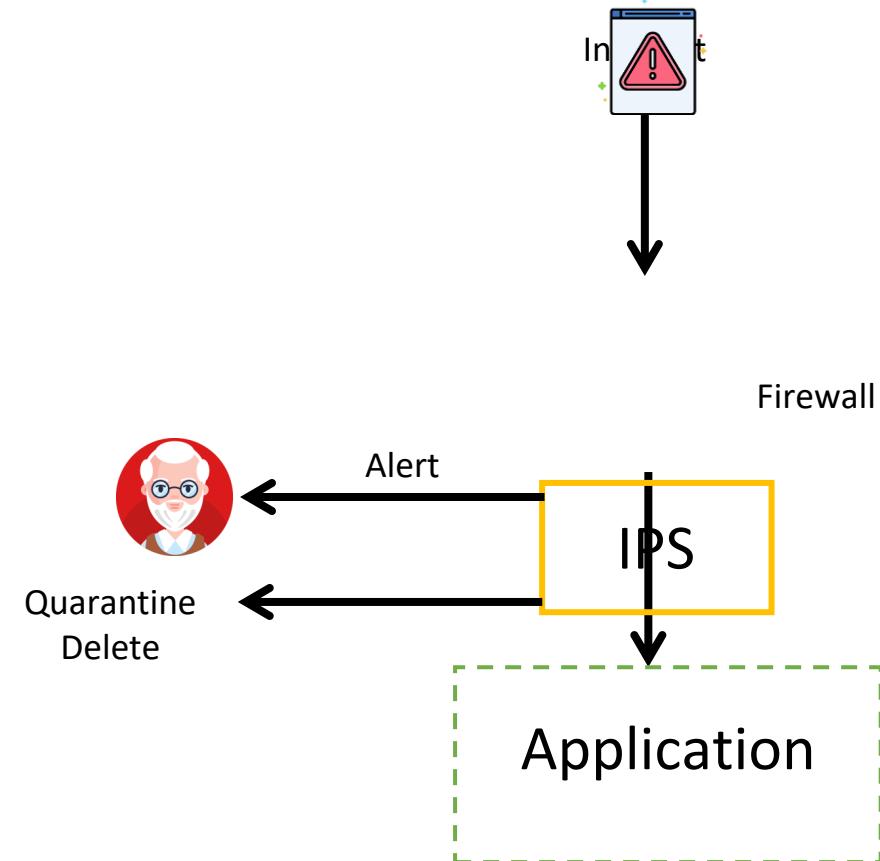
- Scans L3-L7 traffic
- Detects and sends alerts
- Prevents malicious traffic from reaching application

# IDS



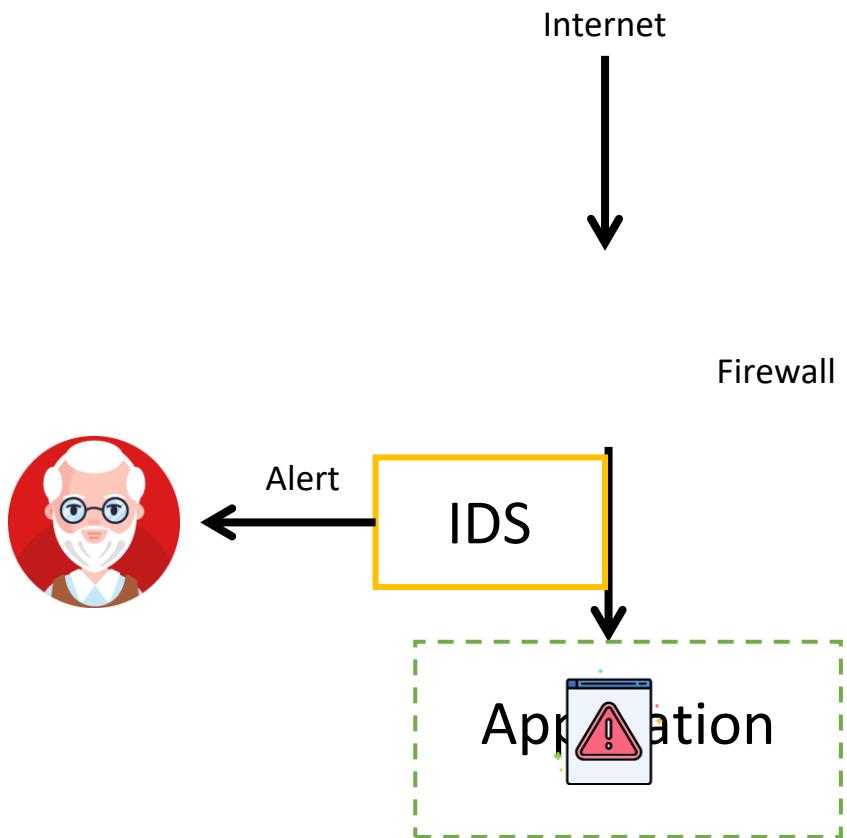
- Scans L3-L7 traffic
- Detects and sends alerts
- Does NOT prevent the traffic

# IPS



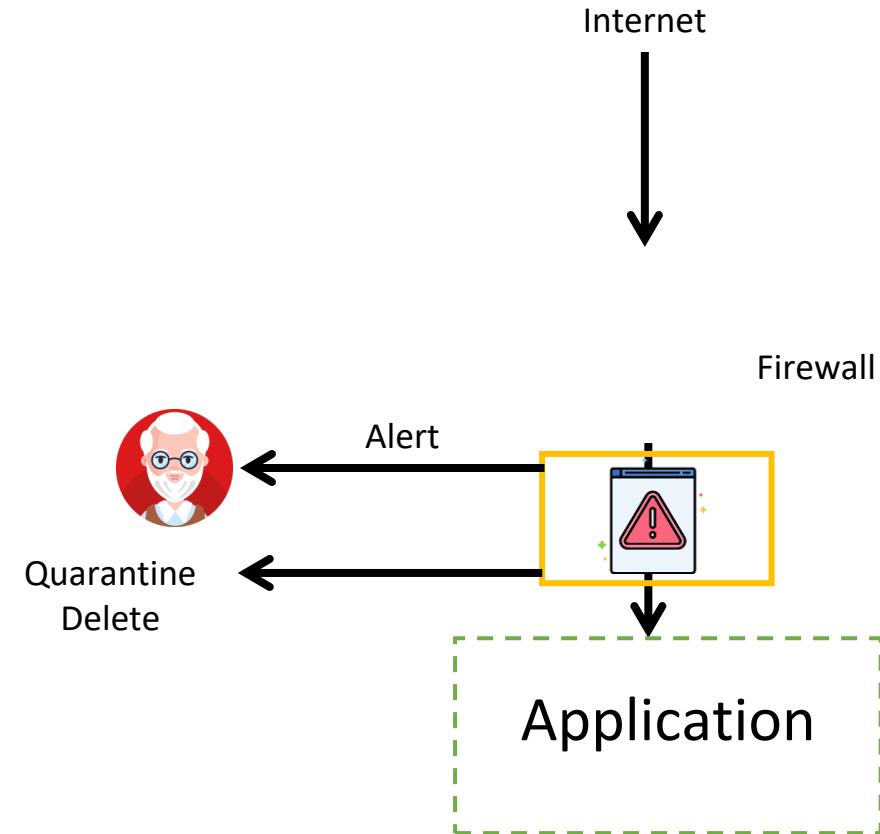
- Scans L3-L7 traffic
- Detects and sends alerts
- Prevents malicious traffic from reaching application

# IDS



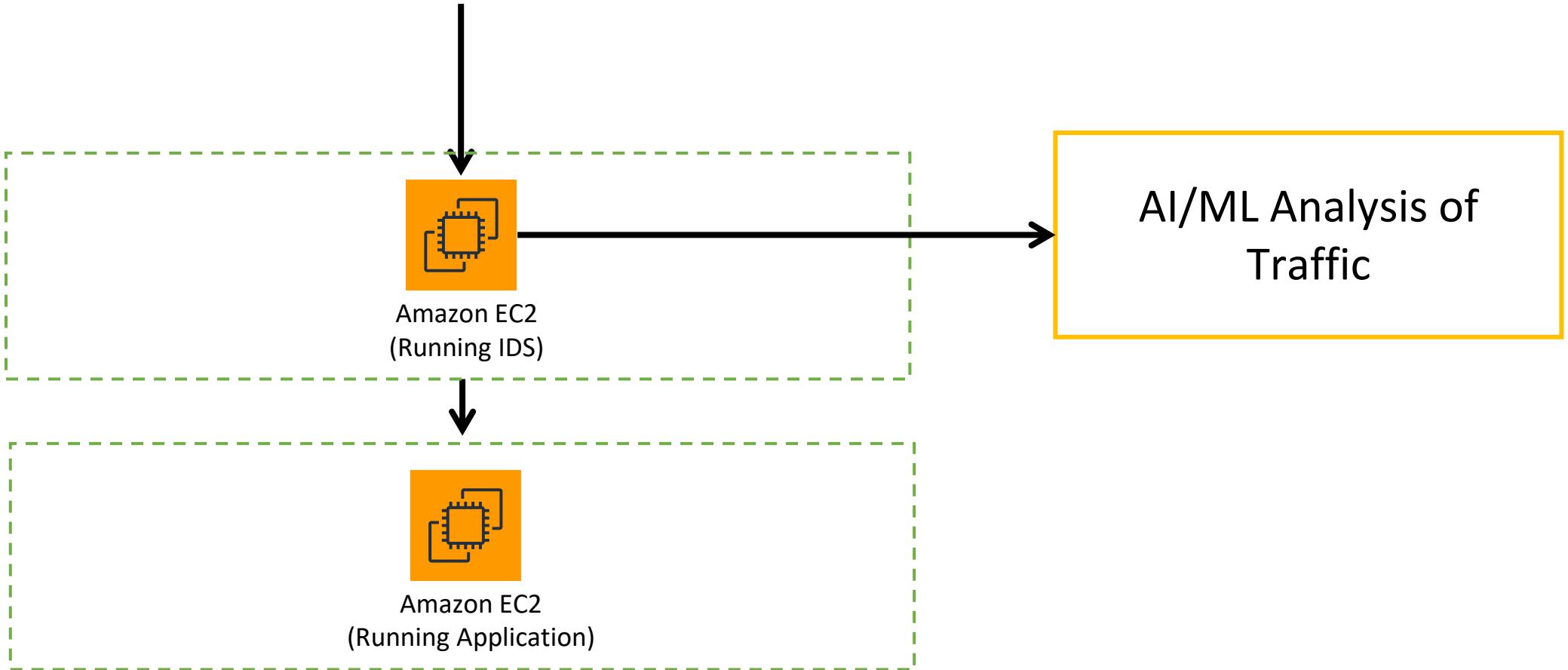
- Scans L3-L7 traffic
- Detects and sends alerts
- Does NOT prevent the traffic

# IPS

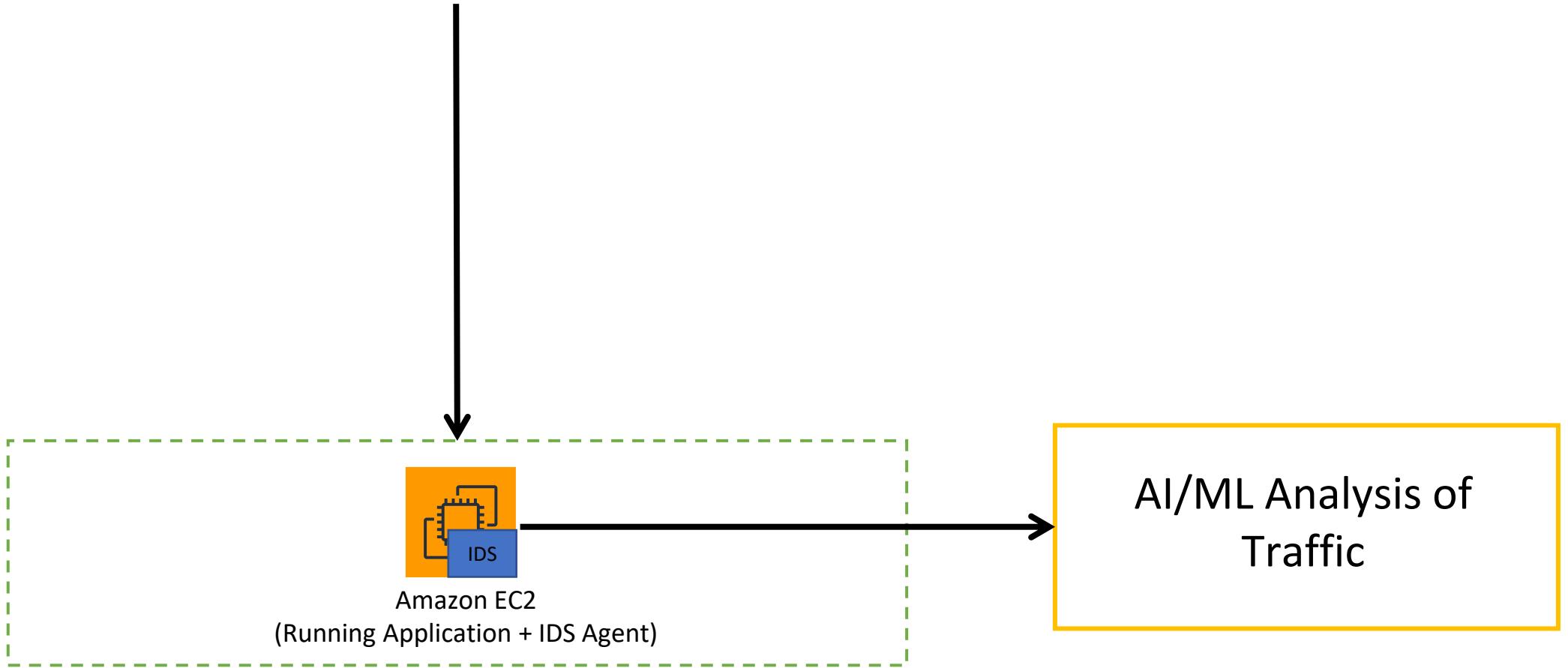


- Scans L3-L7 traffic
- Detects and sends alerts
- Prevents malicious traffic from reaching application

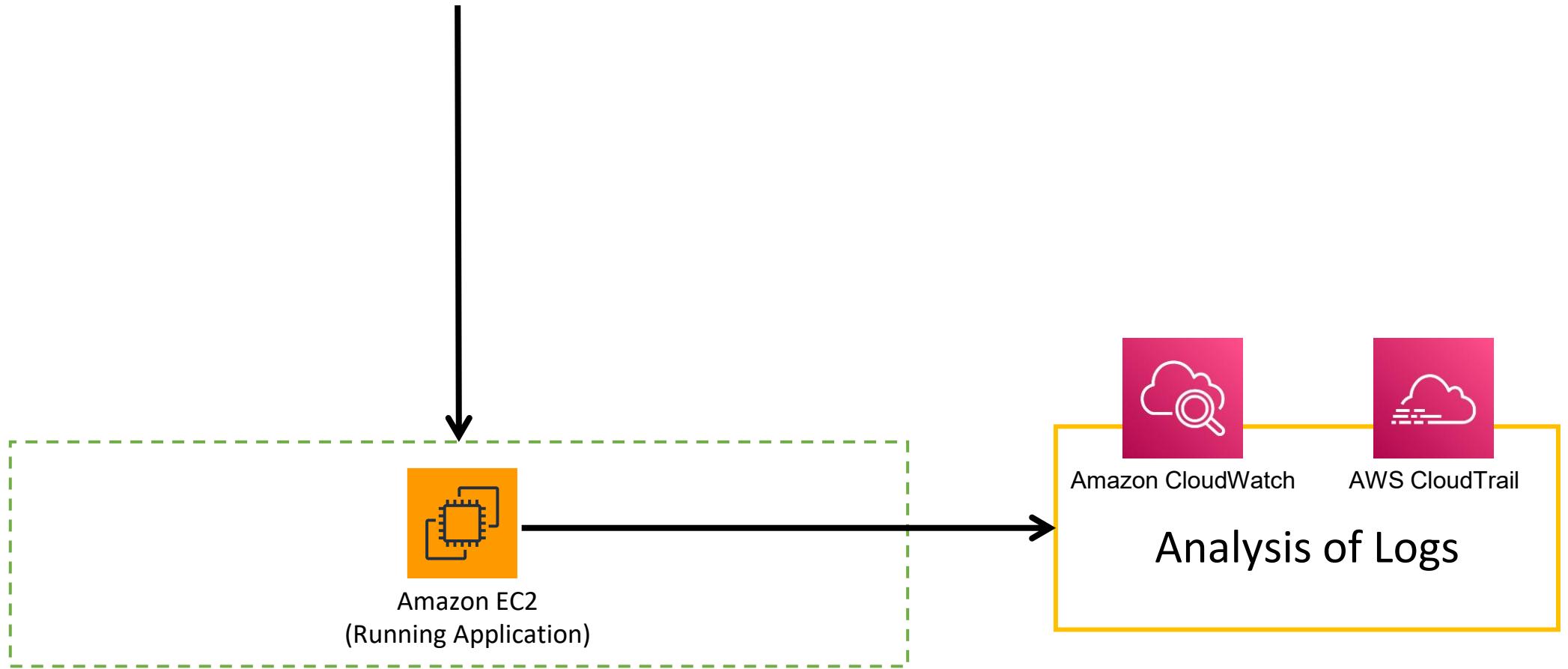
# IDS



# IDS



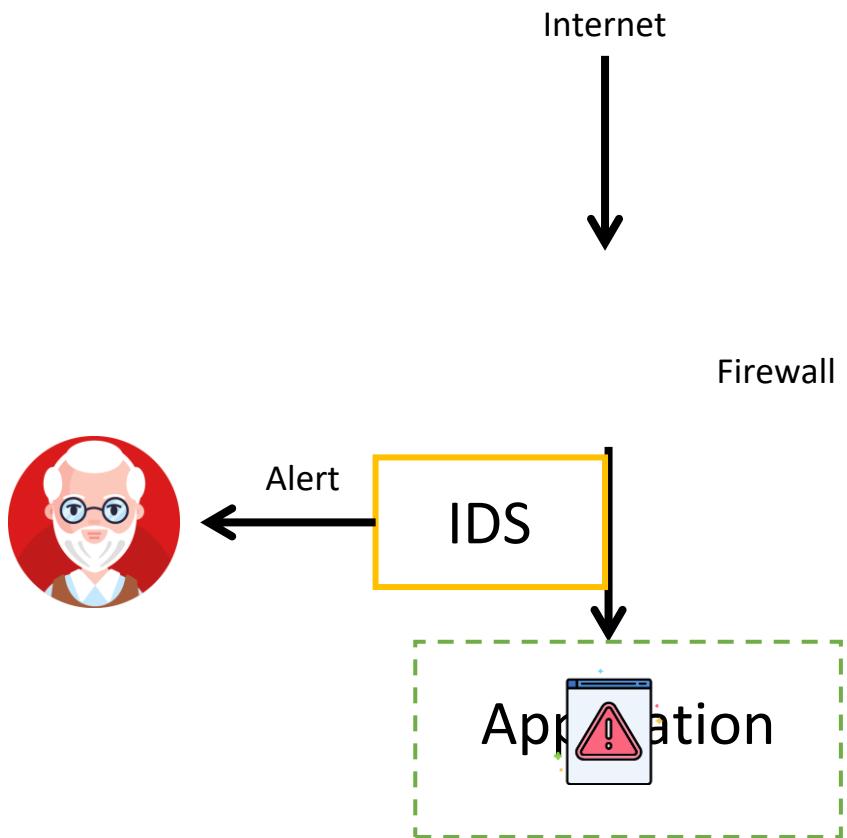
# IDS



# IDS

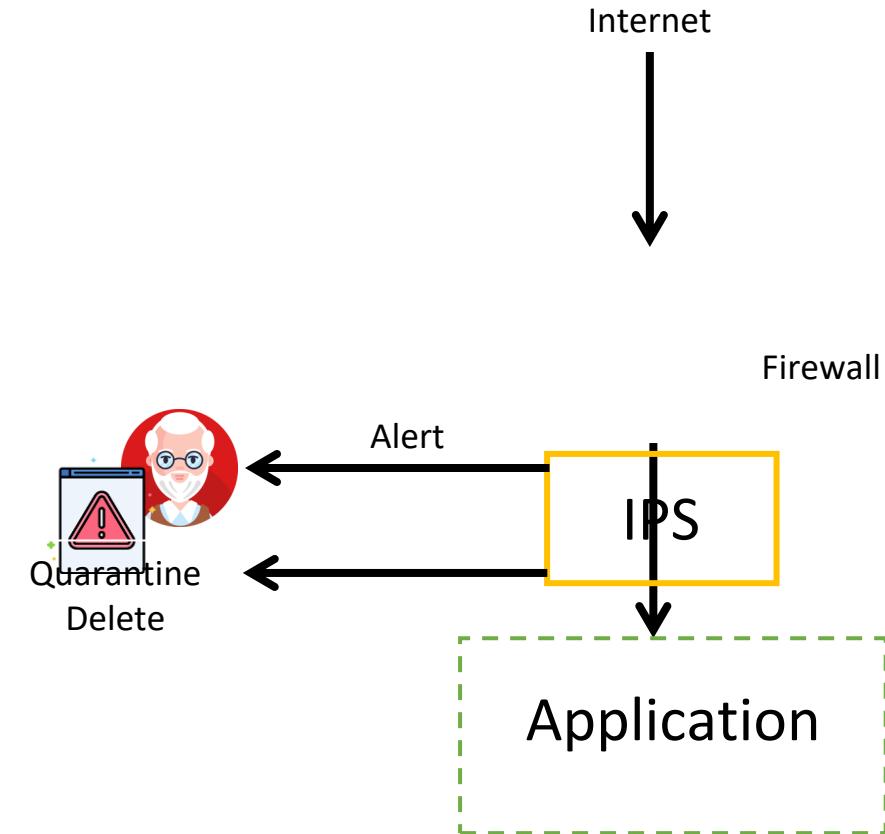


# IDS



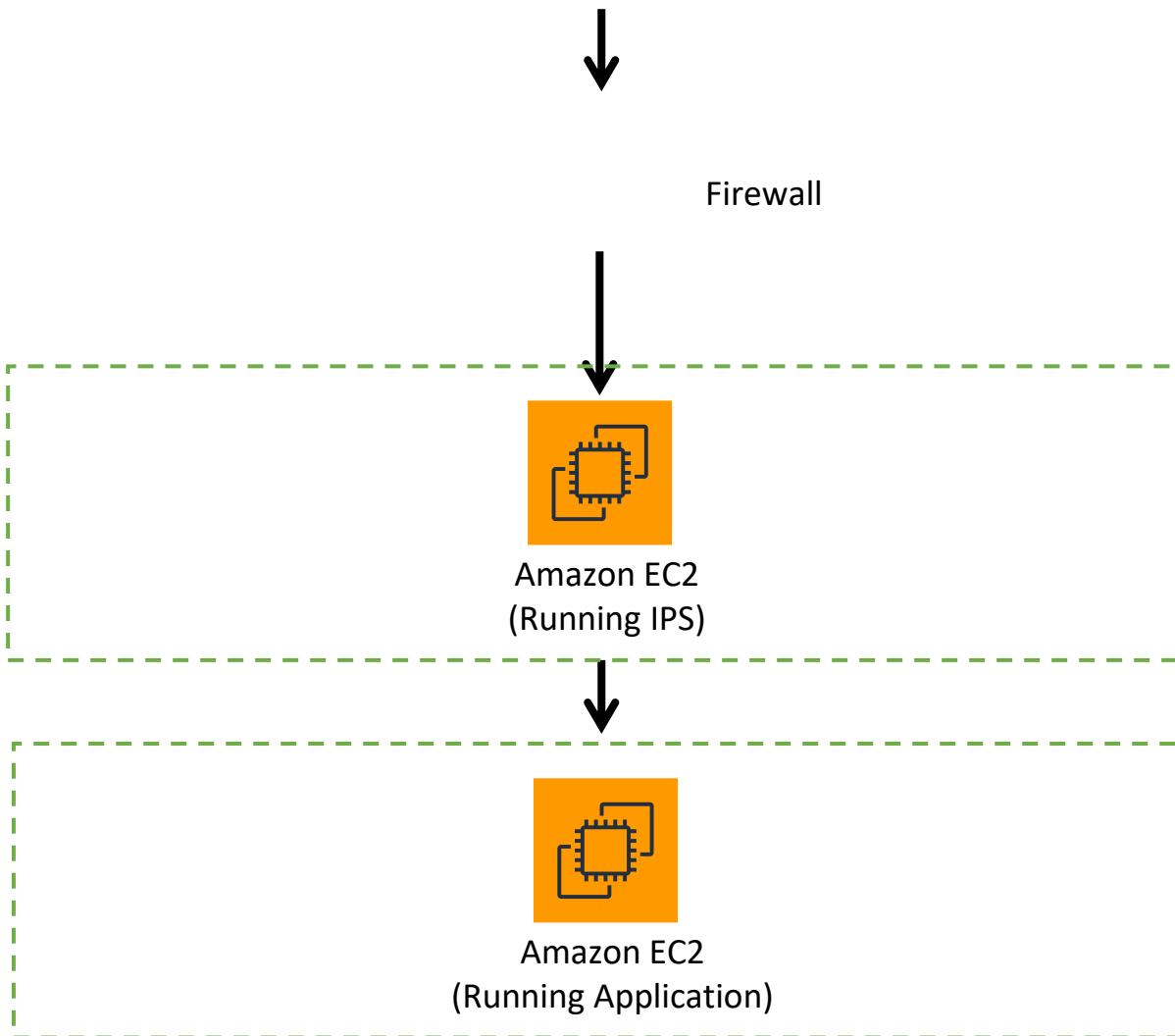
- Scans L3-L7 traffic
- Detects and sends alerts
- Does NOT prevent the traffic

# IPS

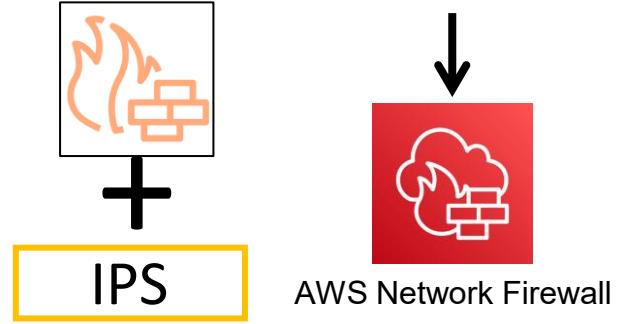


- Scans L3-L7 traffic
- Detects and sends alerts
- Prevents malicious traffic from reaching application

# IPS



# IPS



# IDS/IPS Vs NACL/Security Group

- NACL/Security Group works on just L3/4 layer
  - IDS/IPS works on L3-L7
- Security group does NOT have deny rules
  - IPS have deny rules
- NACL/Security Group doesn't have "intelligence"
  - IDS/IPS has sophisticated rules that gets updated
- IDS/IPS can introduce latency to the app

# SECTION 3 – SYSTEM DESIGN OF MODERN APPLICATIONS

# Must Knows for System Design Interviews

- Microservices
  - Using Load Balancer Vs. API Gateway
  - Sync Vs. Async patterns
- Database Selection
  - SQL Vs. NoSQL
- Caching
  - Caching of Database and CDN
- Security
  - AuthN/Z, Encryption at Rest and Transit
- Make it Scalable and Highly Available

# URL Shortener (TinyUrl/Bit.ly)

Bitly | Link Management

app.bitly.com/B18u37VkmAW/bitlinks/2UWIXp4

Apps New Tab 3080 Amazon ElastiCache SAM local testing Database Migration... Whitepapers – Ama... Best practices for m... Map AWS services t... The Kubernetes API... Google Kubernetes... AWS Well-Architect... Lambda functions a... Map of the AWS W... How do I prepare f... AWS reInvent 2018... Other bookmarks Reading list

All Links

Search Links

CREATE Agent of Change Free Account

Date Created Top Performing

Filters Tag Show Hidden Links Only

Upgrade for custom links SELECT DATE

11 TOTAL CLICKS

11 Email, SMS, Direct TOP REFERRER

8 United States TOP LOCATION

Total Links Created

AUG 10 AUG 17 AUG 24 AUG 31

5 Results Clicks all time

CREATED AUG 30, 3:13 AM | Agent of Change

Rocking Kubernetes with Amazon EKS, Fargate, And DevOps | Udemy

<https://www.udemy.com/course/rocking-kubernetes-with-amazon-eks-fargate-and-devops/?couponCode=AUG21BP1>

bit.ly/2UWIXp4 COPY SHARE EDIT UPGRADE QR CODE

4 TOTAL CLICKS

AUG 29 SEP 1

MONDAY, AUG 30 Total Clicks 2

DATA IN UTC

REFERRERS LOCATIONS

?

2

CREATED AUG 30, 3:13 AM | Agent of Change

# Rocking Kubernetes with Amazon EKS, Fargate, And DevOps | Udemy

<https://www.udemy.com/course/rocking-kubernetes-with-amazon-eks-fargate-and-devops/?couponCode=AUG21BP1>

bit.ly/2UWIXp4

COPY

SHARE

EDIT

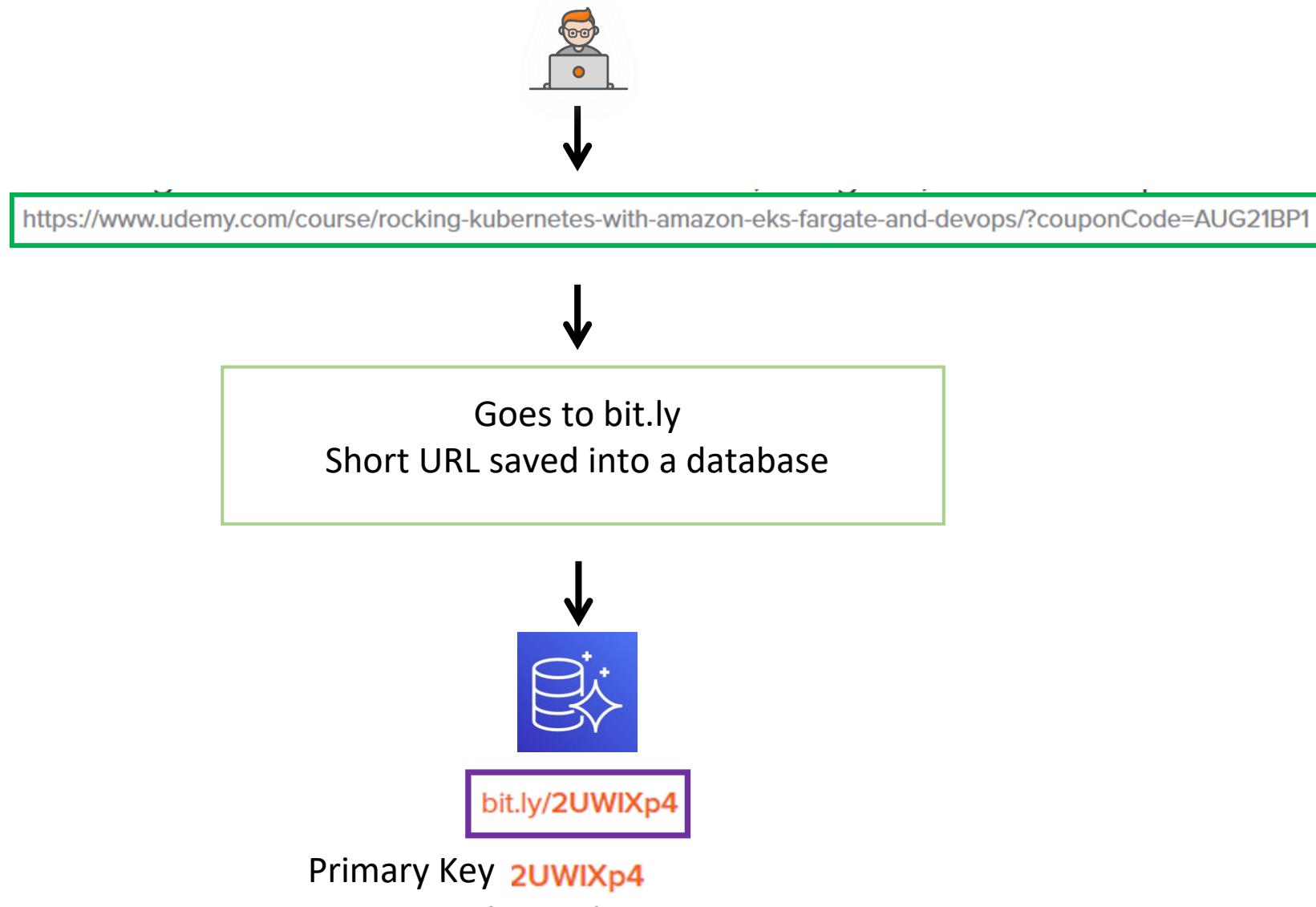
UPGRADE

QR CODE

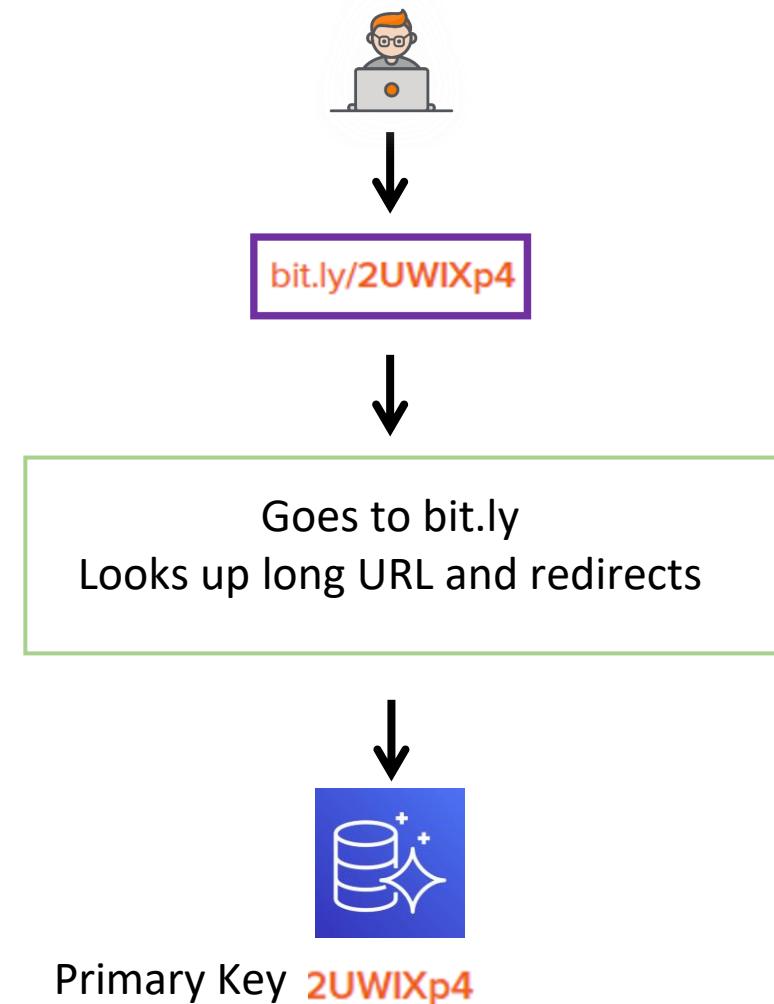
4 

TOTAL CLICKS

# Basic Functionality - Saving

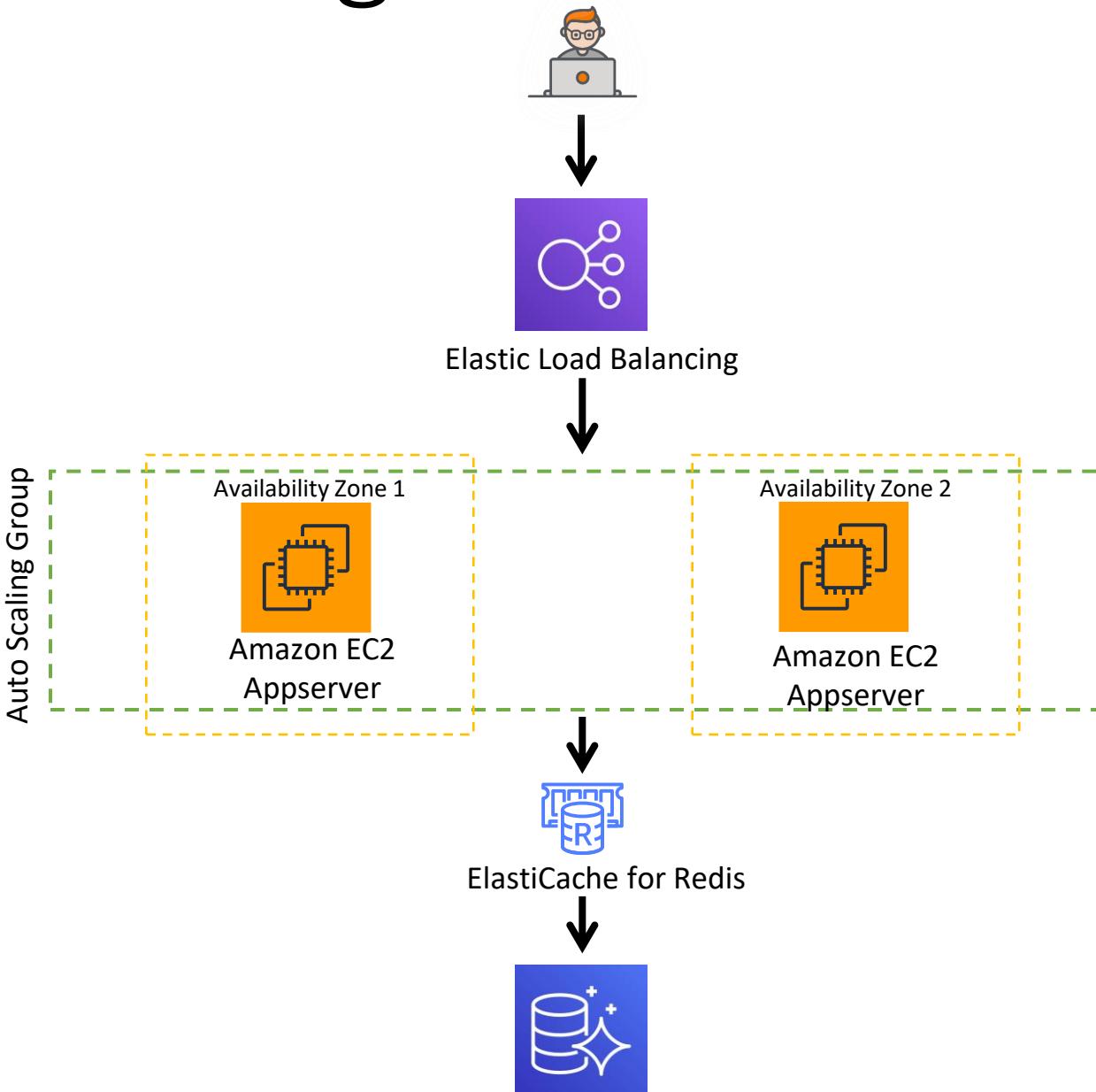


# Basic Functionality - Retrieving

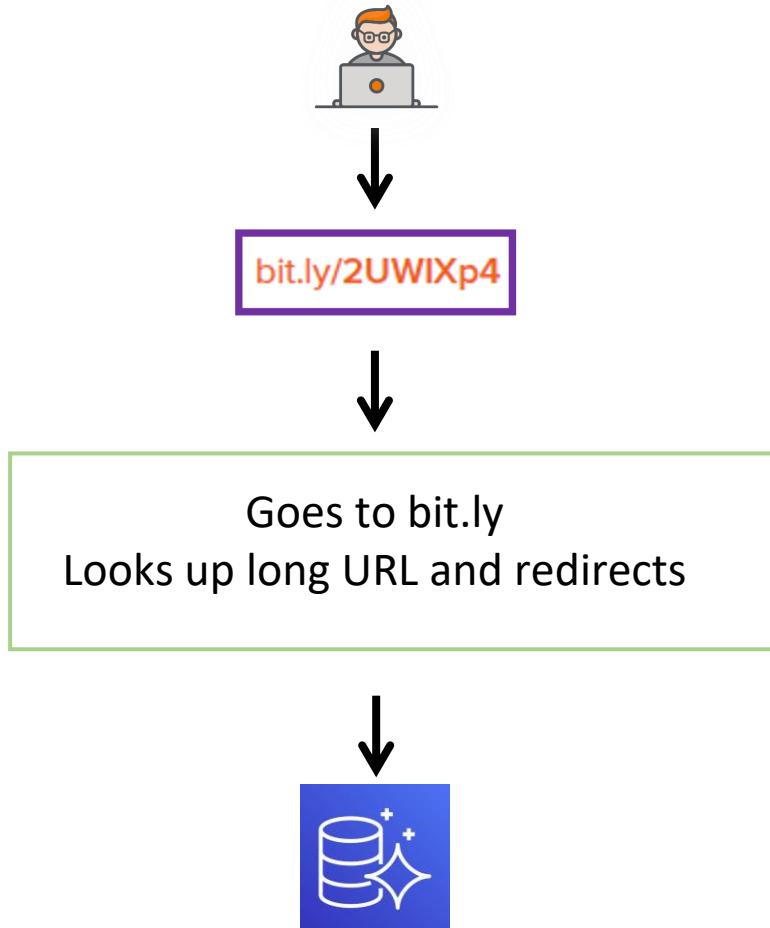


<https://www.udemy.com/course/rocking-kubernetes-with-amazon-eks-fargate-and-devops/?couponCode=AUG21BP1>

# High Level Diagram



# What is Interviewer Looking for?



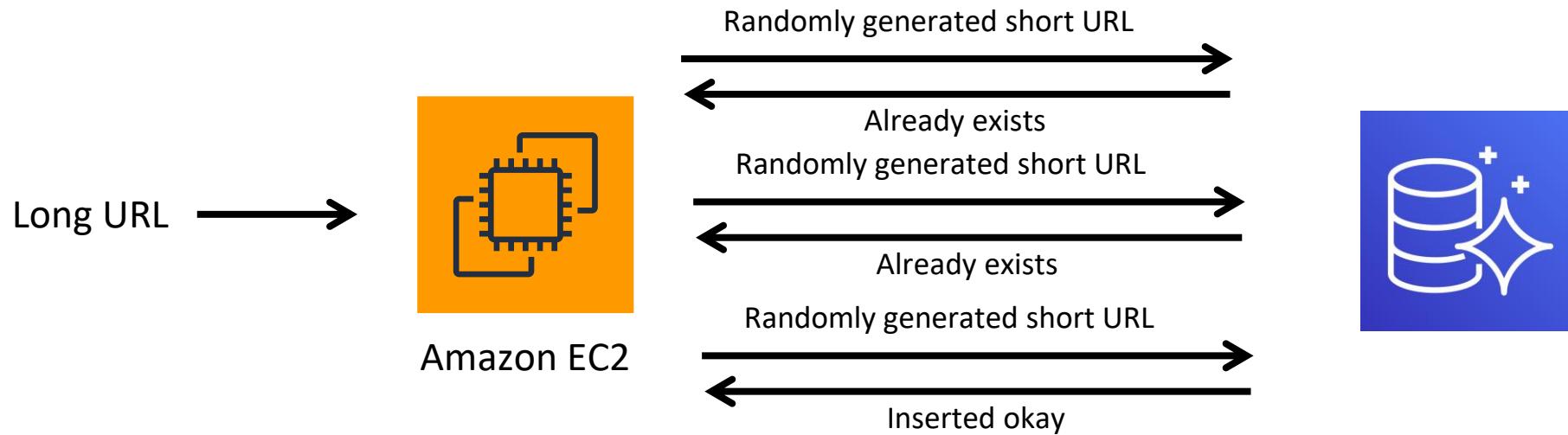
- How is the 7-byte shortened URL generated
- How can the URL generator scale?

# Some Math!

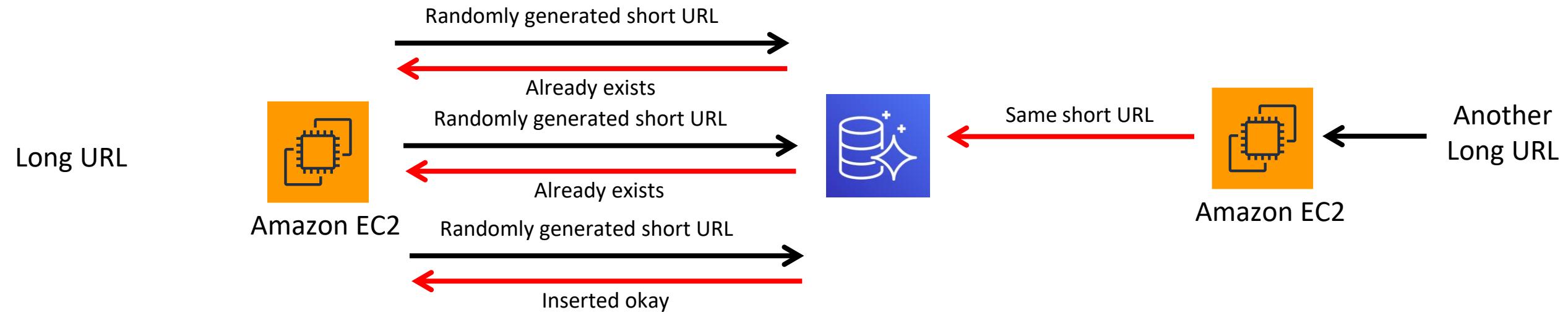
bit.ly/2UWIXp4

- Shortened URL can contain:
  - a-z = 26 characters
  - A – Z = 26 characters
  - 0-9 = 10 characters
- Total of  $(26 + 26 + 10) = 62$  characters
- 7 characters URL out of 62 characters =  $62^7$ 
  - = 3.5 Trillion unique combinations = 42 bits ( $2^{42}$ )
- Depending on rate of consumption, increase the shortened URL size

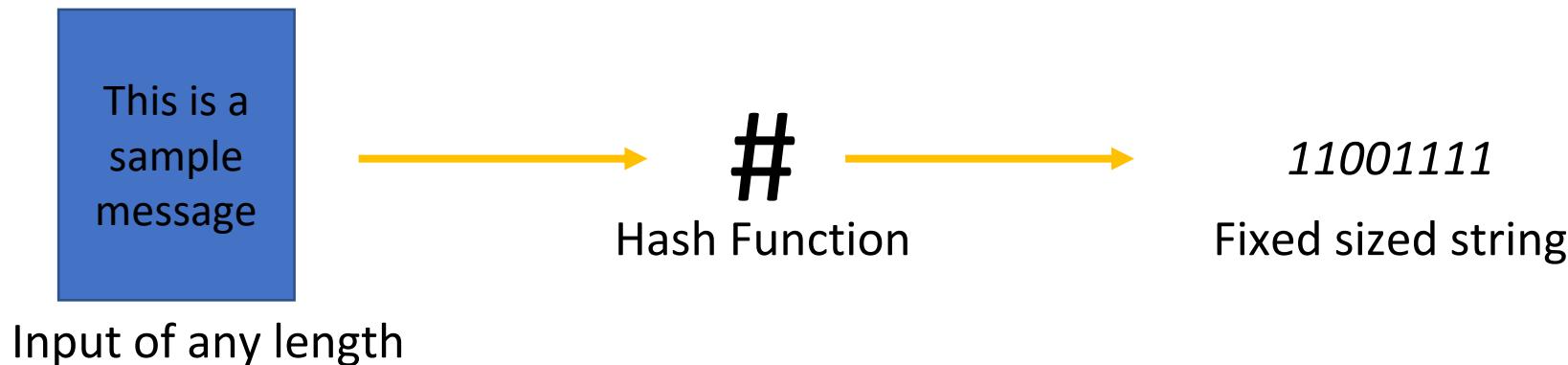
# Bad Randomizer Approach



# Bad Randomizer Approach



# Taking Randomness Out – MD5



- Message Digest Algorithm 5
- Produces 128-bit value from an arbitrary length string
- Hash algorithm
  - Same input strings will always generate same output string
  - Two different strings can NOT produce the same output

# MD5 Continued

- Take first 42 bits of the 128 bit MD5 output
- 42 bits to 7 characters  
1101.... (Total 42 bits)  
 $=2^1+2^1+2^0+2^1 + \dots$   
 $=12345$

## Base62

---

From Wikipedia, the free encyclopedia

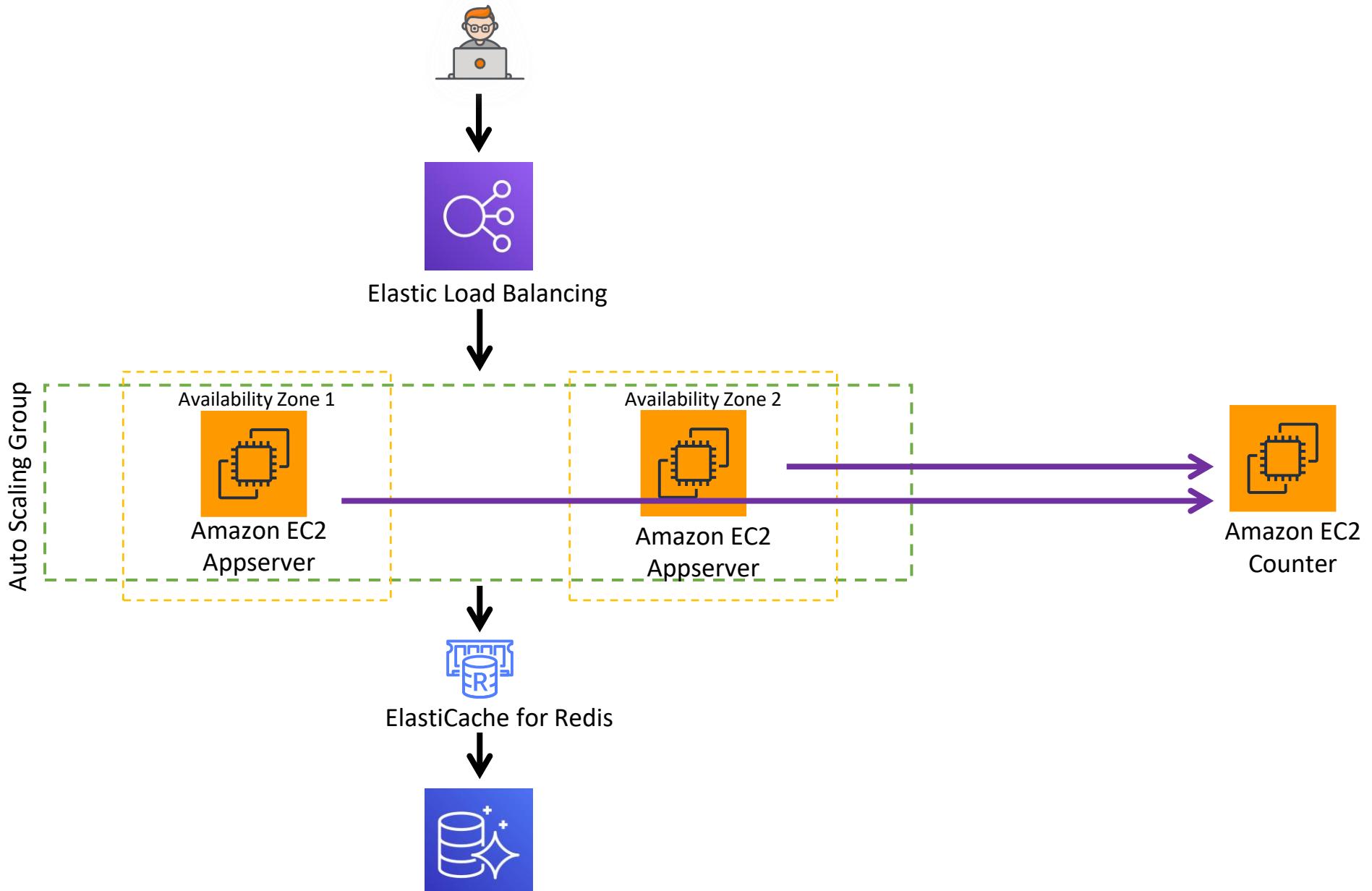
The **base62** encoding scheme uses 62 characters. The characters consist of the capital letters A-Z, the lower case letters a-z and the numbers 0–9. It is a [binary-to-text encoding](#) schemes that represent [binary data](#) in an [ASCII](#) string format.<sup>[1][2]</sup>

Convert to Base 62 (Get numbers from 0-61)  
=3idarWH

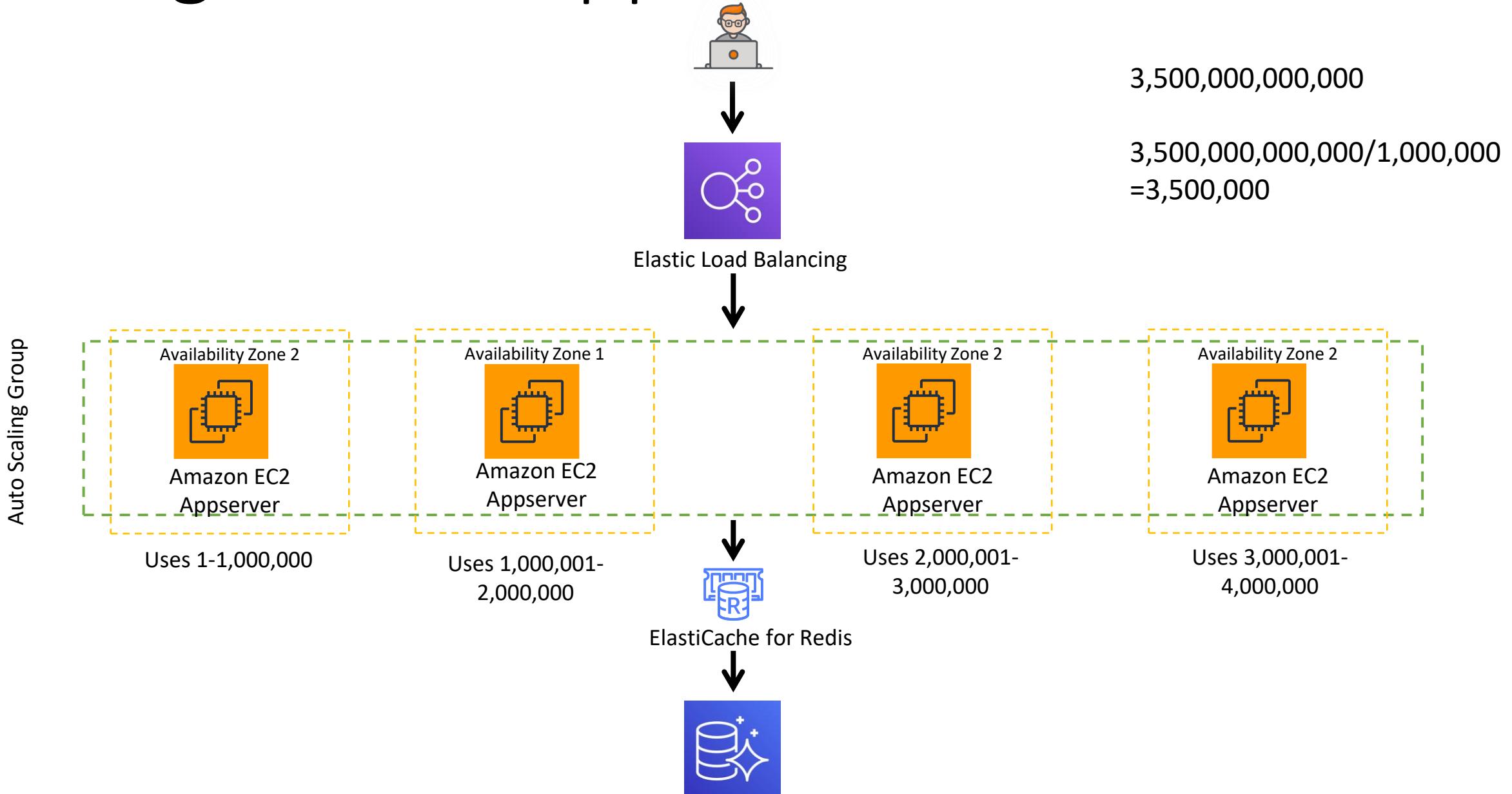
# Ideal Approach

- 42 bits to 7 characters (Remember 42 bits translate to max 3.5 trillion in actual numeric)
- For each URL shortening request if the application has a unique number between 1 - 3.5 Trillion as input, the output will be unique
- For each URL request use a number, then increment it by one for the next request
- Do this from 1 - 3.5 trillion

# Where should the Counter be?



# Range Based Approach



# Assigning Ranges

Maintains ranges from 0-3.5 trillion with each range of 1 million



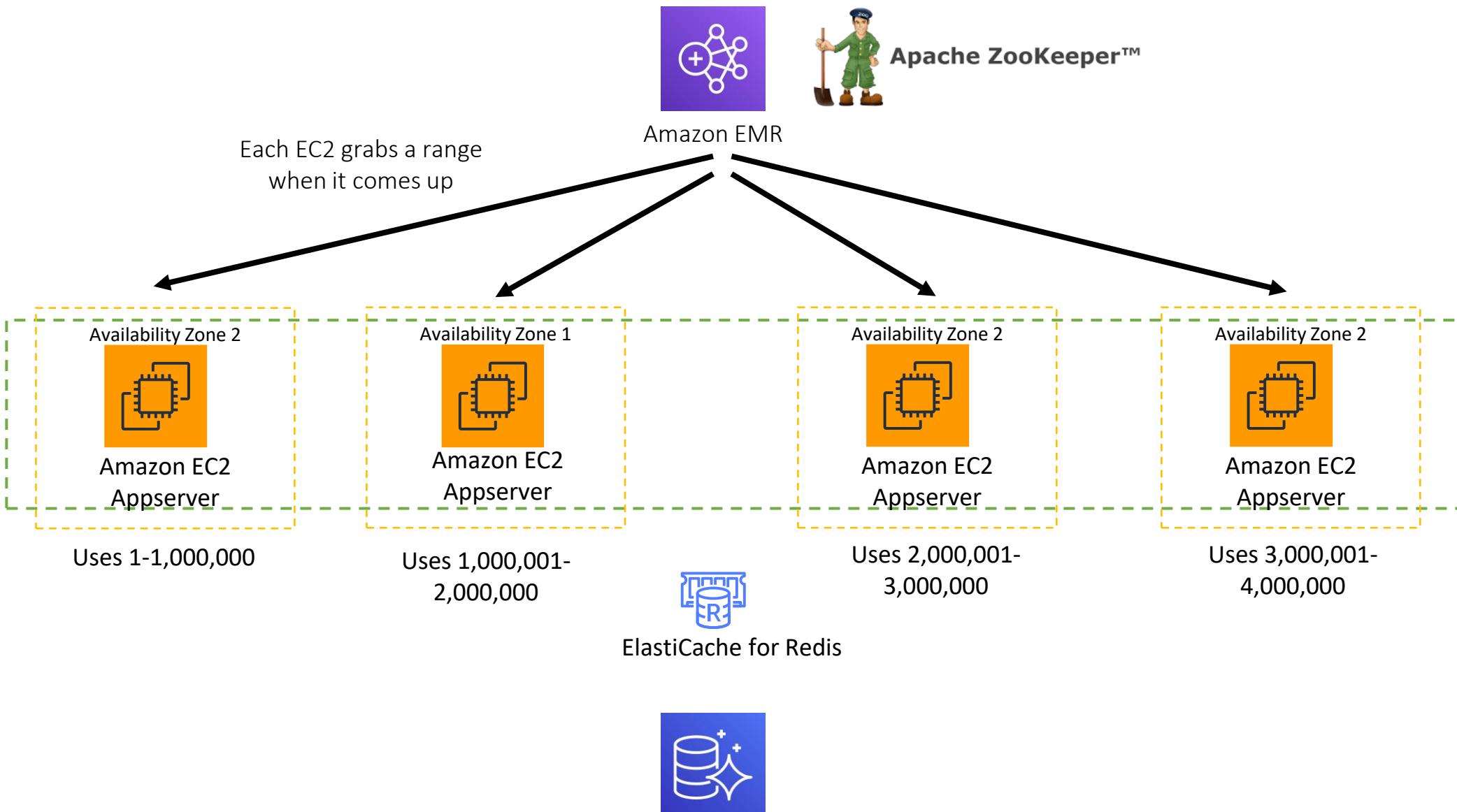
# Quick Detour to Apache ZooKeeper

Maintains ranges from  
0-3.5 trillion with each  
range of 1 million



- Centralized service for maintaining configuration information
- Highly available and provides distributed synchronization
  - Better than running a counter service on single EC2
- Can be run in Amazon EMR

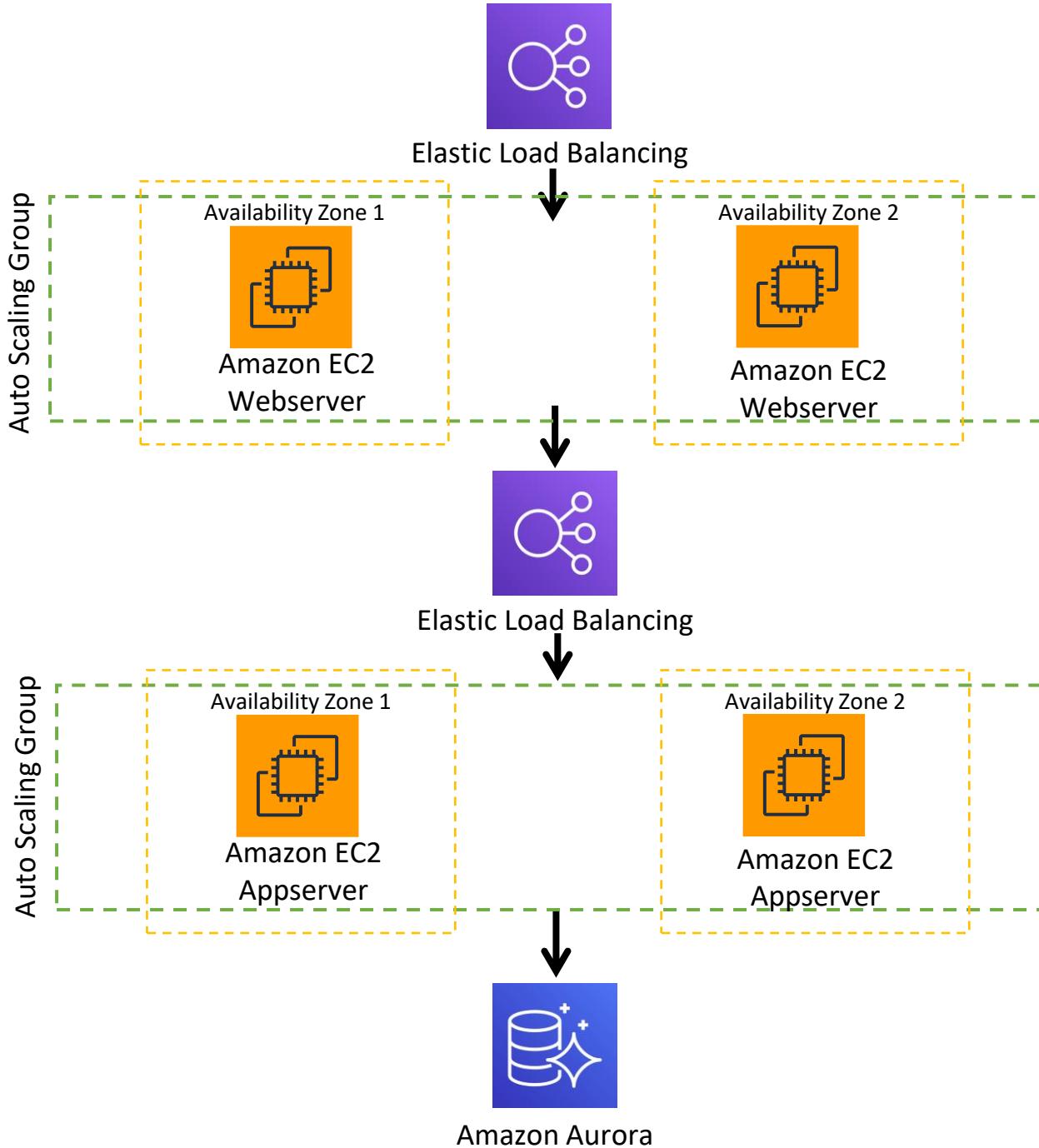
# Assigning Ranges



# Amazon/Flipcart

# Three-Tier Architecture

## Presentation Layer



## Application Layer

## Database

# Requirements/Design Spec

## REQUIREMENTS

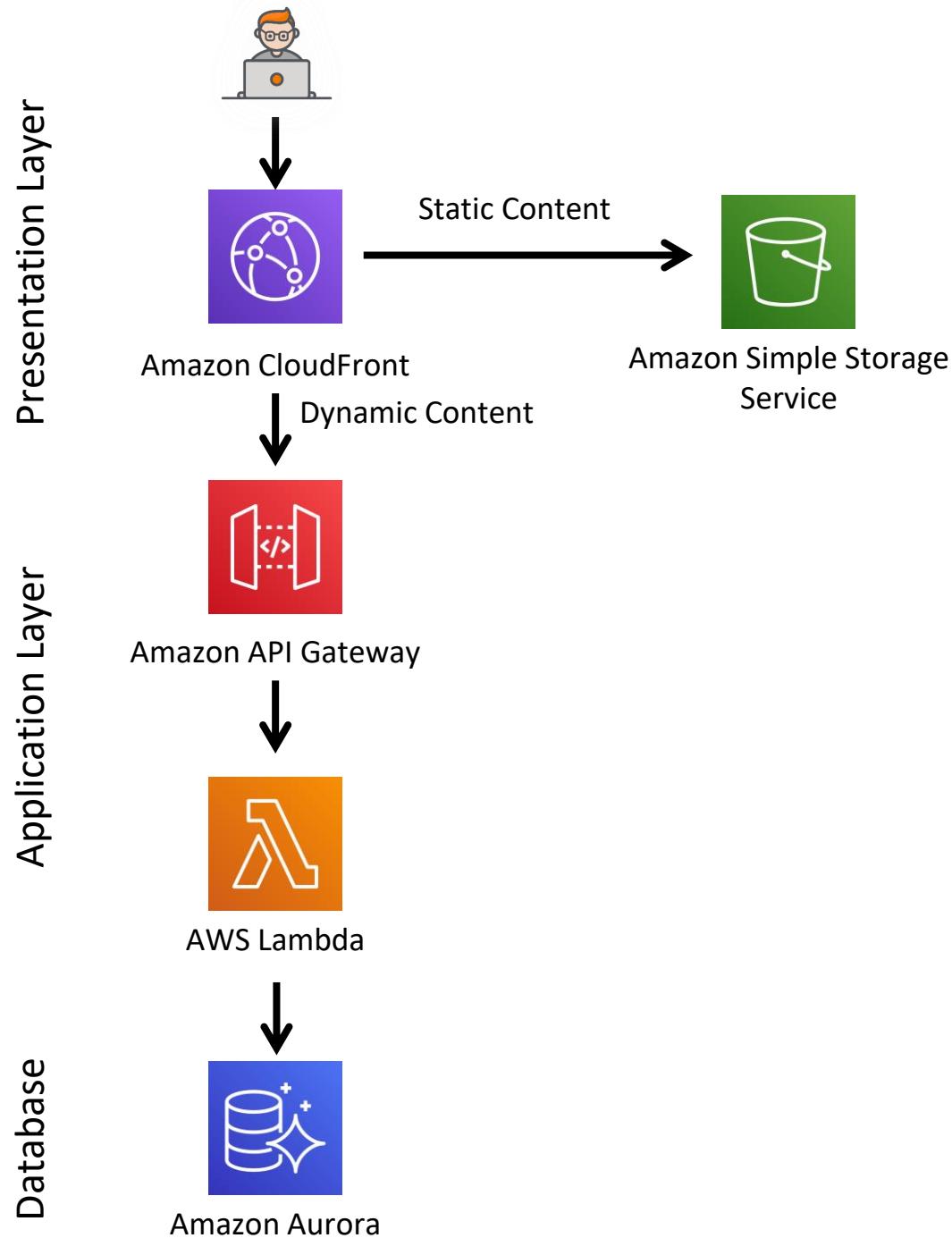
- Product catalog
- Shopping cart
- Buy product
- Product recommendation

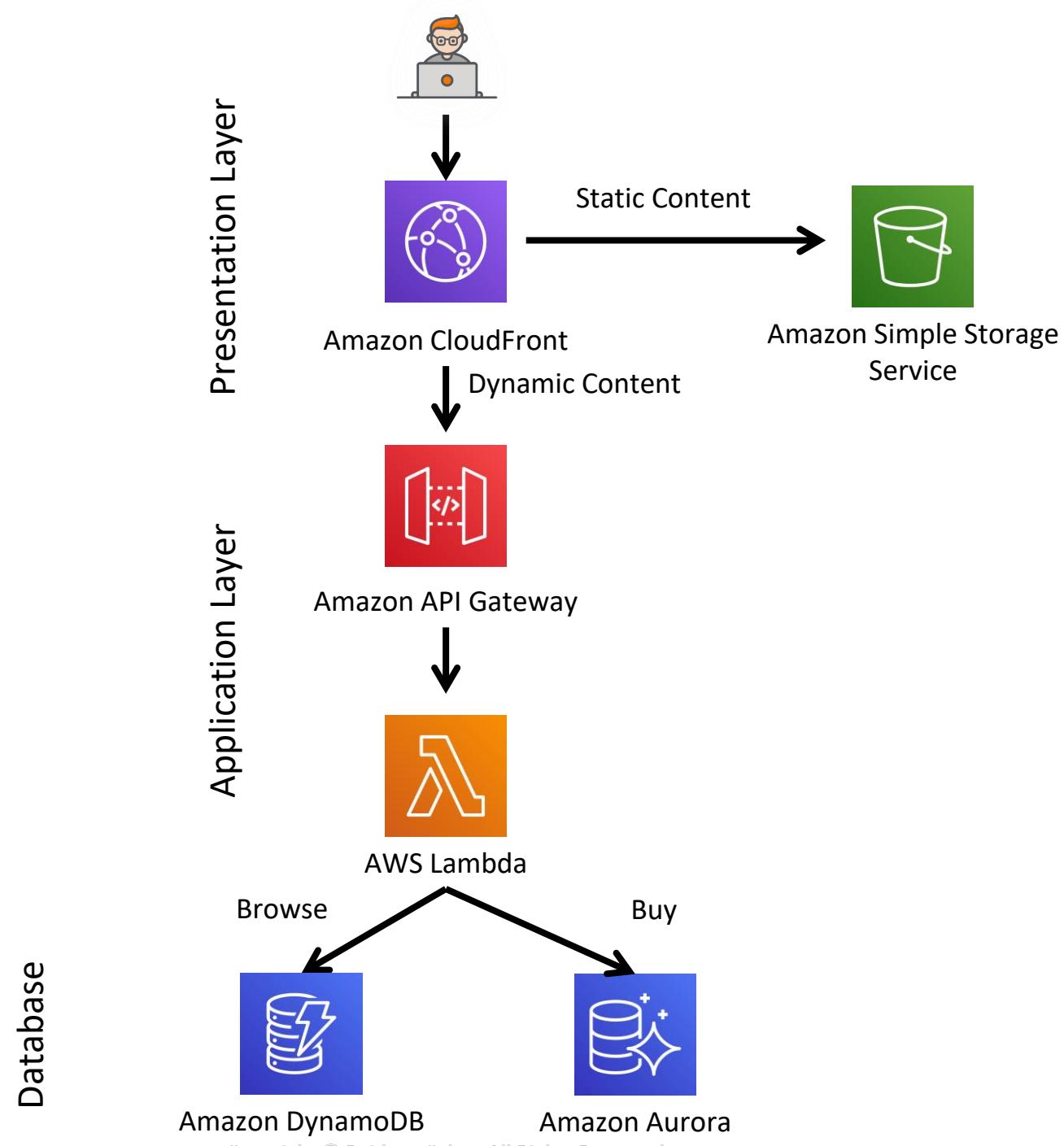
## DESIGN SPEC

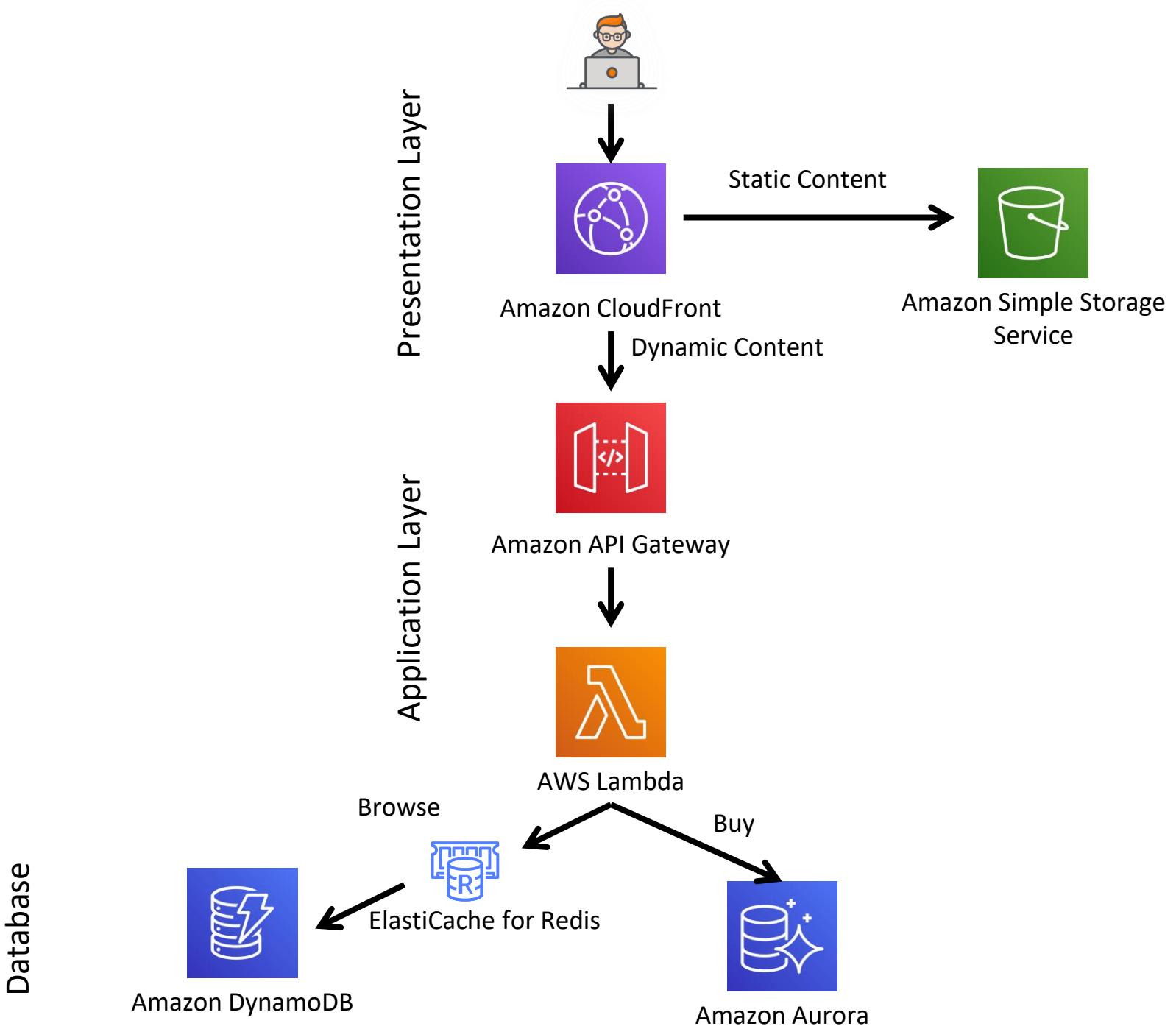
- Scalable
- Highly Available
- Cost efficient
- Secure

# Three-Tier Architecture

## EC2 Vs Kubernetes Vs Serverless







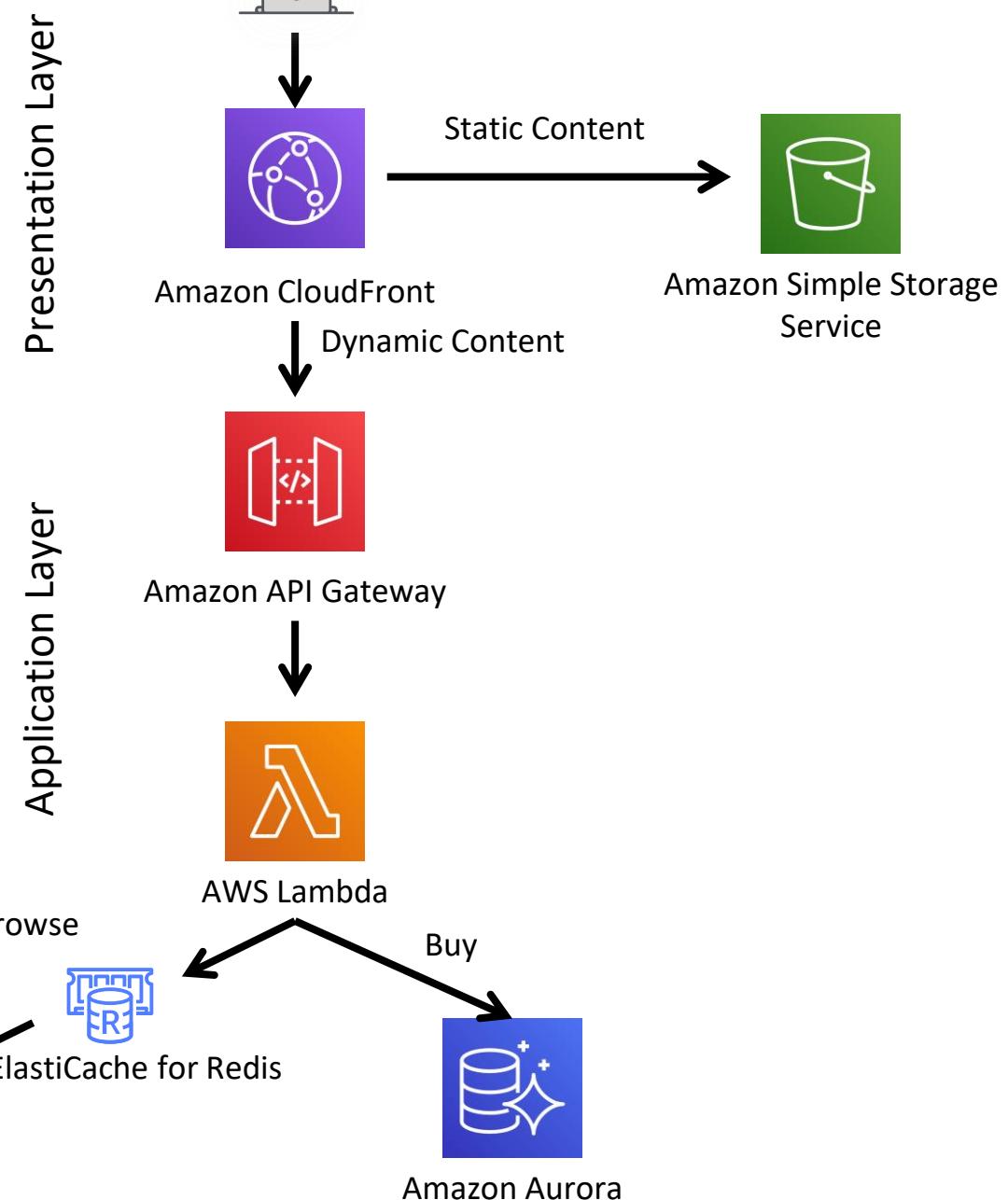
# Database Probing

- Expect lots of database questions
- Keep SQL vs NoSQL, sharding, Read replica, caching, global database in mind

Database



Amazon DynamoDB



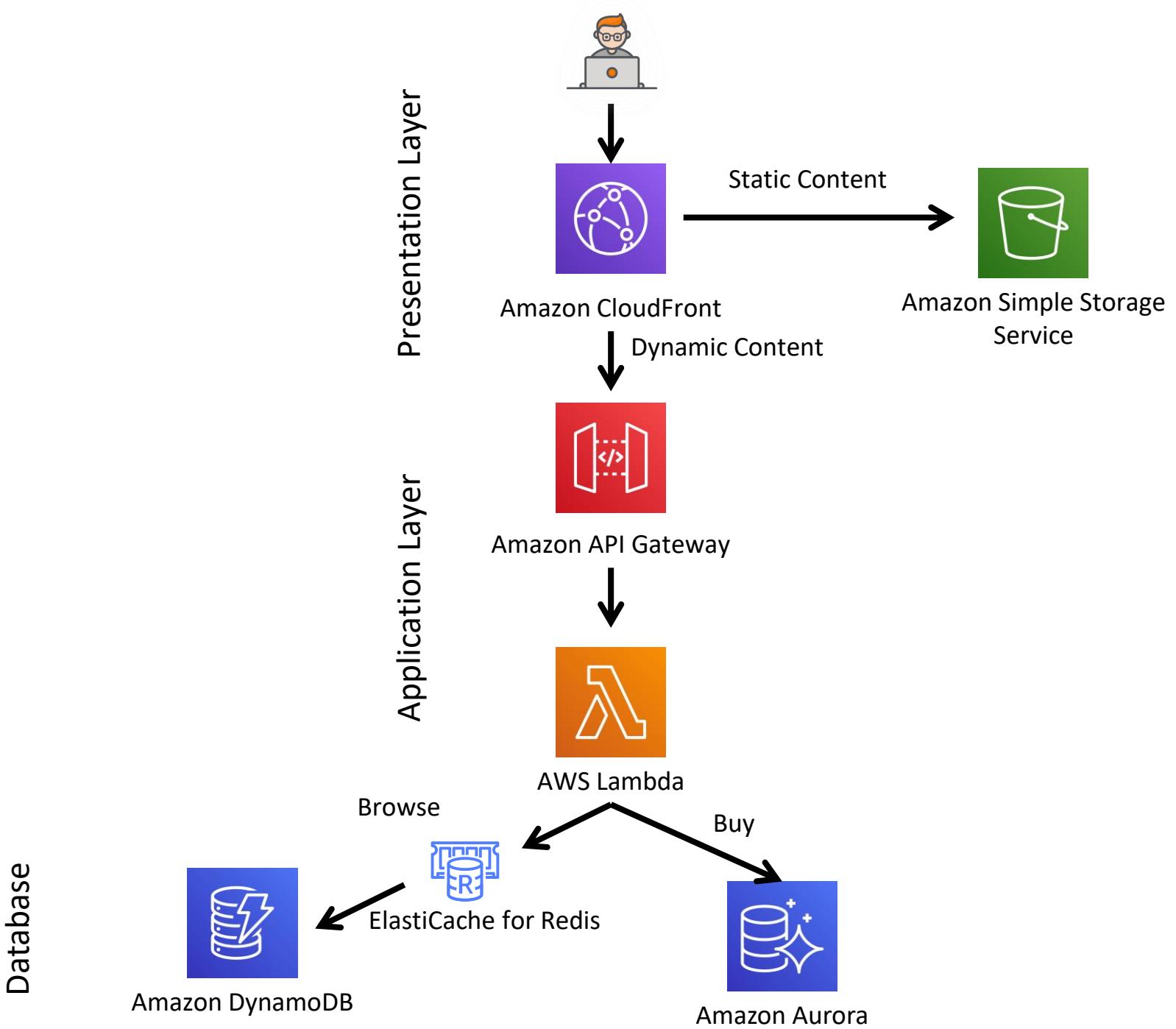
# Shopping Cart

Product

ProductID	Name	Price	AvailableCount
100	TV	\$450	5
200	Face Mask	\$5	1000
300	Hand Sanitizer	\$10	0

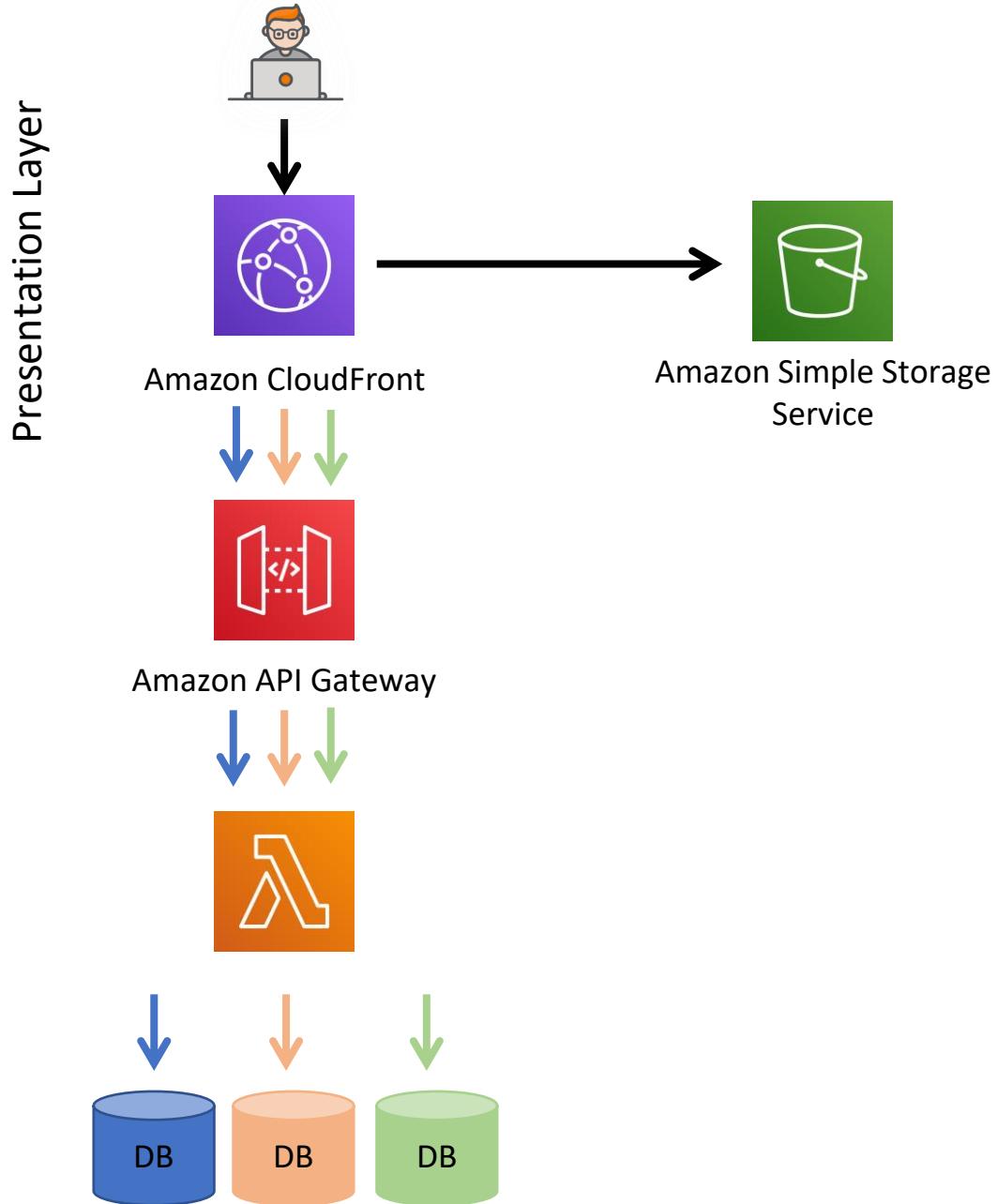
ShoppingCart

CartID	PersonLogin	ProductID
10000	John.Wilson	100
10000	John.Wilson	200
20000	Tina.Smith	300



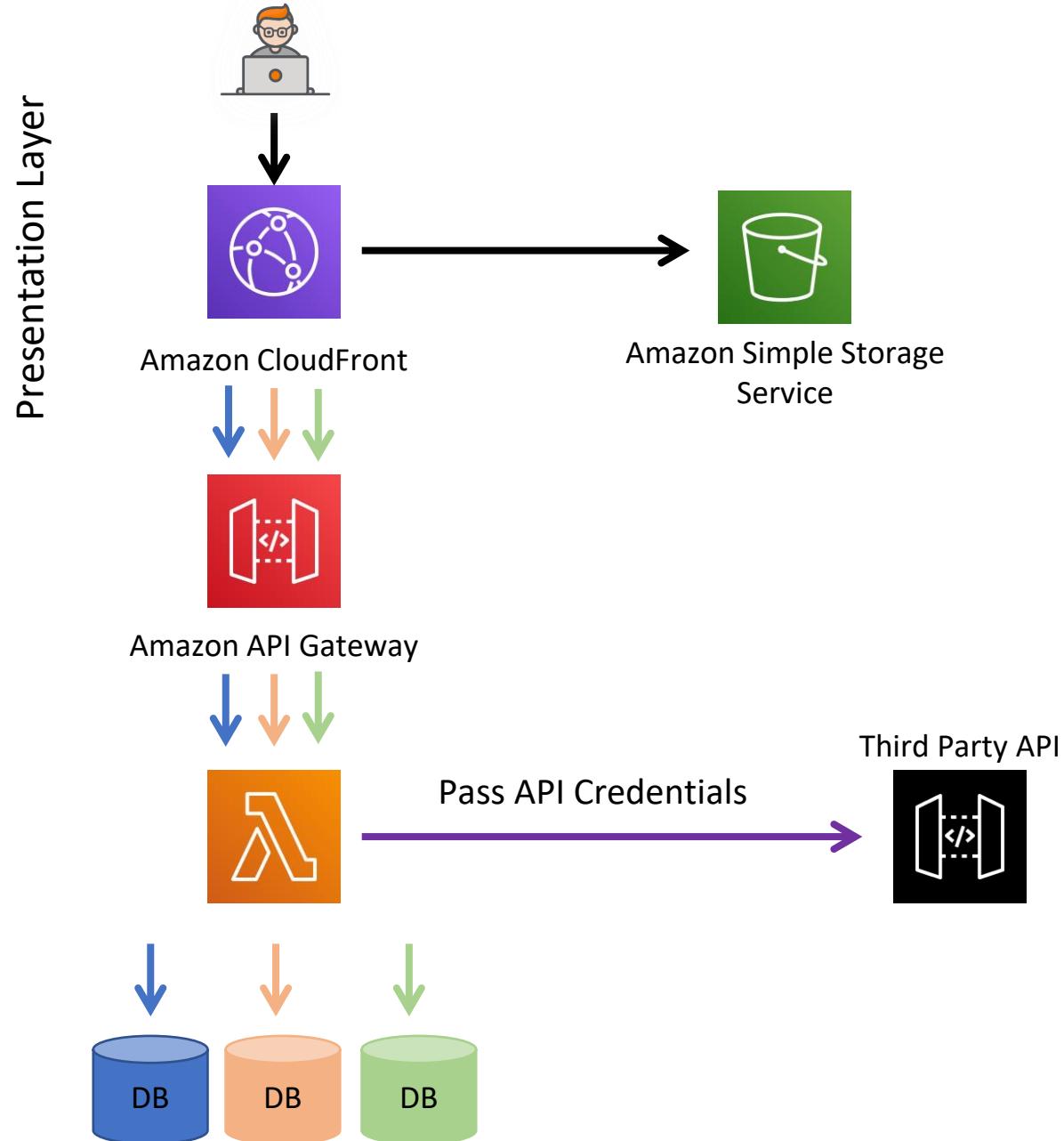
# Several Microservices

Database  
Application Layer



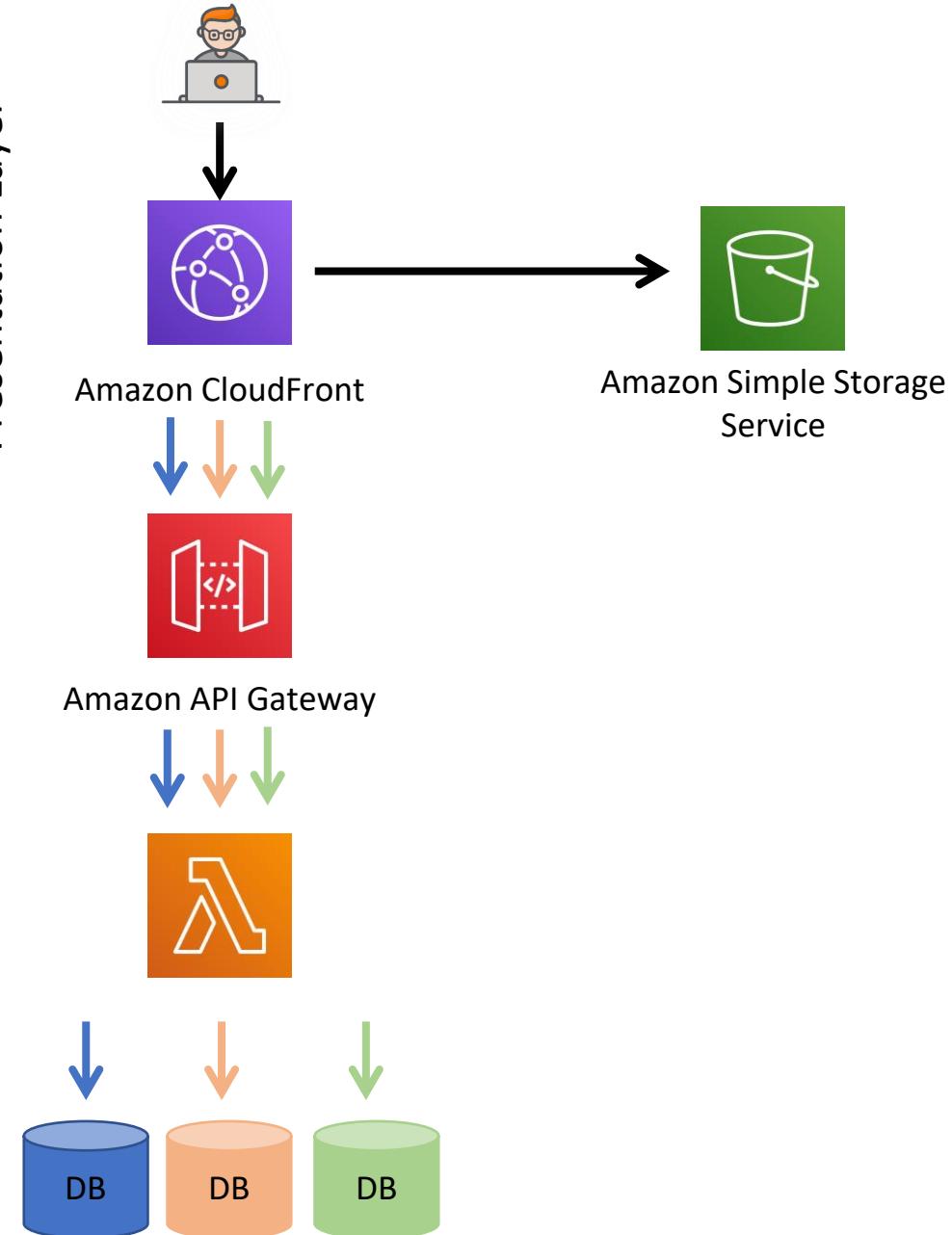
# Calling Third Party APIs

Database  
Application Layer

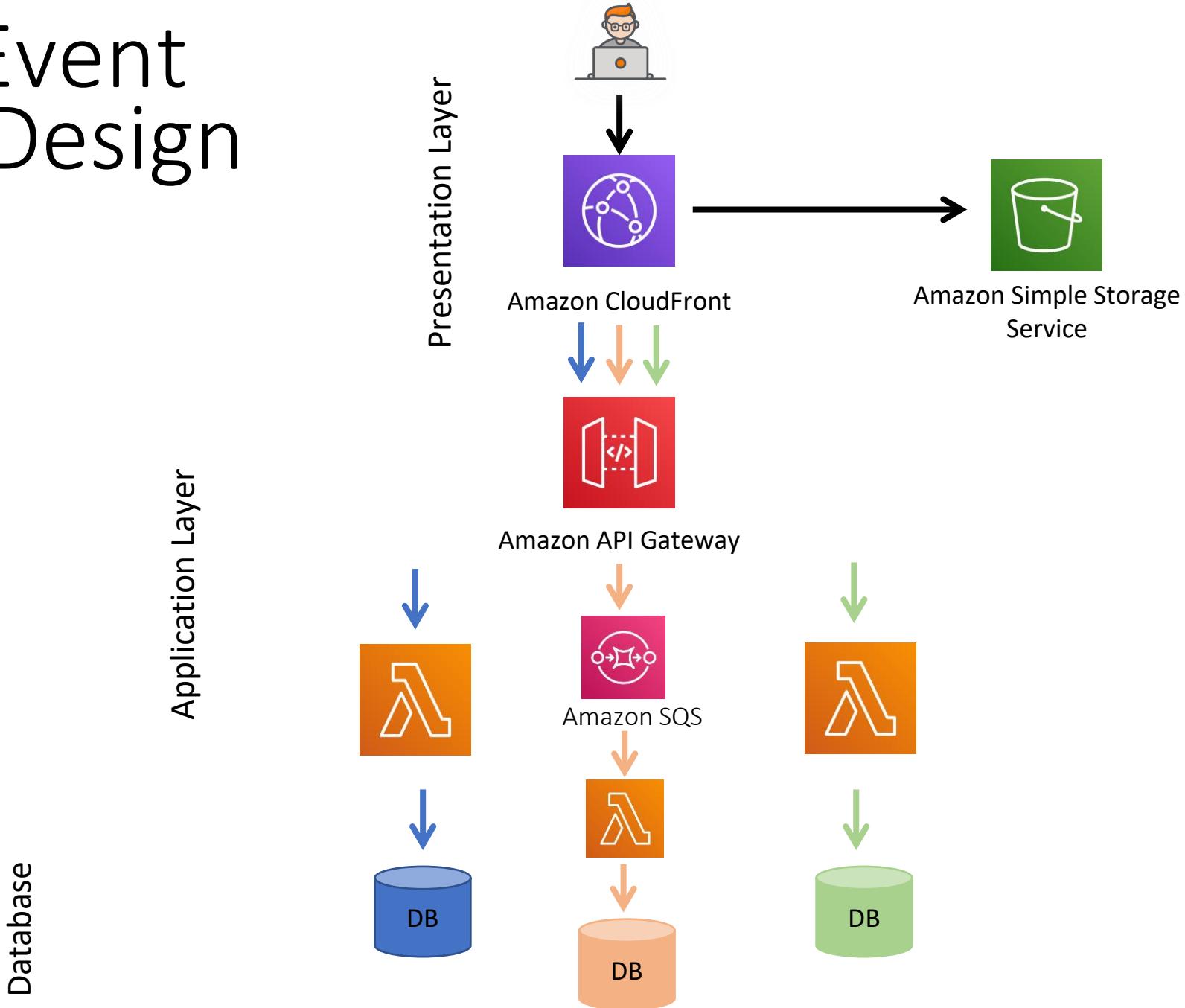


# Handling Super High Traffic

Database  
Application Layer  
Presentation Layer



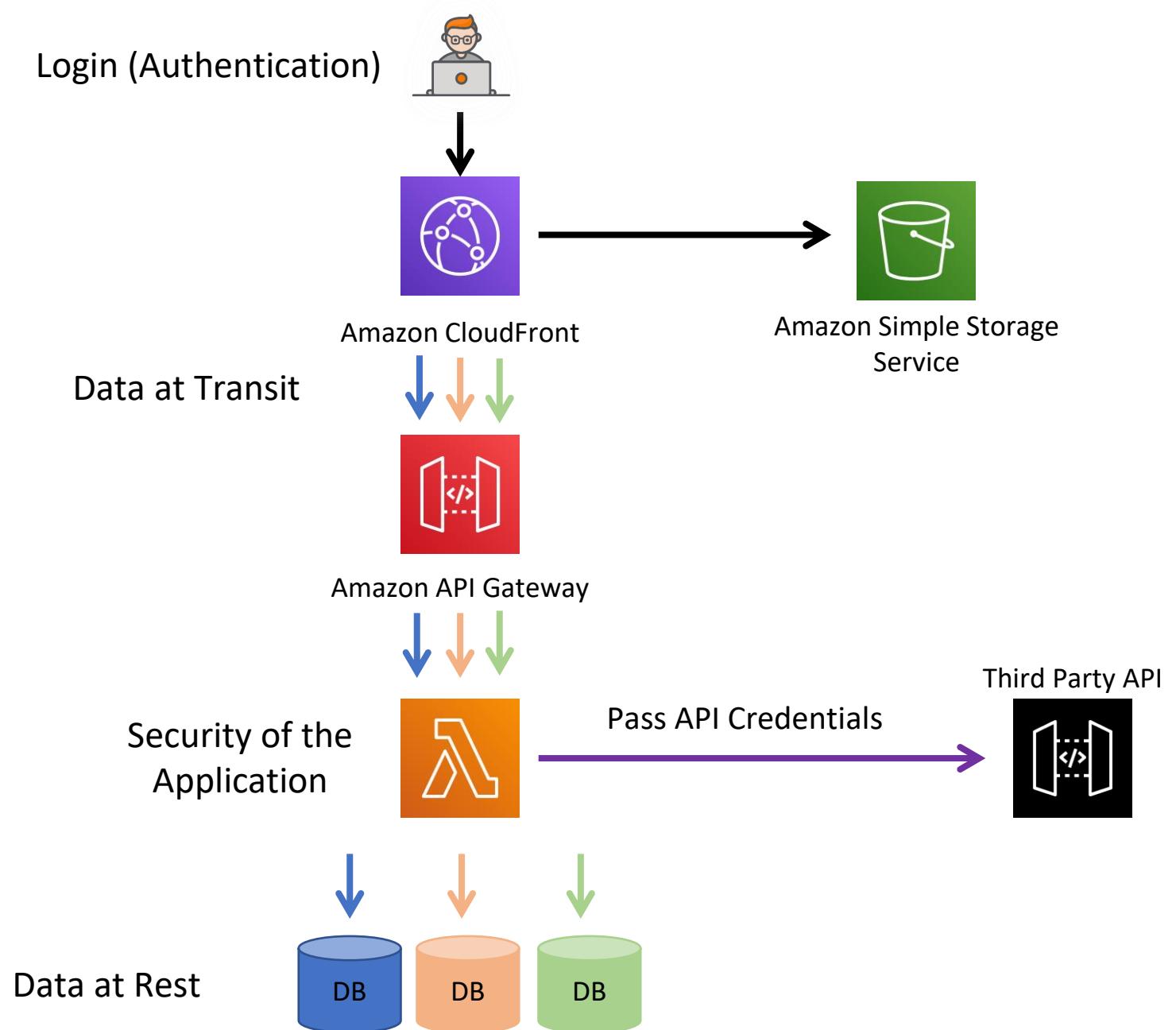
# Async/Event Driven Design



# Product Recommendation

- Collaborative Filtering
- Check Tinder system design recommendation part

# Security



[IT & Software](#) > [Other IT & Software](#) > [Amazon EKS](#)

# Rocking Kubernetes with Amazon EKS, Fargate, And DevOps

Learn Docker Container on Kubernetes on AWS EKS, Fargate along with deploying real world applications with DevOps.

Highest Rated **4.5 ★★★★☆** (48 ratings) 358 students

Created by [Rajdeep Saha | Cloud Architect @ Fortune10 Company](#)

>Last updated 8/2020    English    English [Auto]

Wishlist

Share

Gift this course

**Preview this course**

Black Friday Sale

**\$9.99** ~~\$94.99~~ 89% off

⌚ 6 days left at this price!

Add to cart