

AWS Academy Cloud Architecting

# Module 9: Implementing Elasticity, High Availability, and Monitoring



## Sections

1. Architectural need
2. Scaling your compute resources
3. Scaling your databases
4. Designing an environment that's highly available
5. Monitoring

## Demonstrations

- Creating Scaling Policies for Amazon EC2 Auto Scaling
- Creating a Highly Available Web Application
- Amazon Route 53

## Labs

- Guided Lab: Creating a Highly Available Environment
- **Challenge Lab: title**



**Knowledge check**

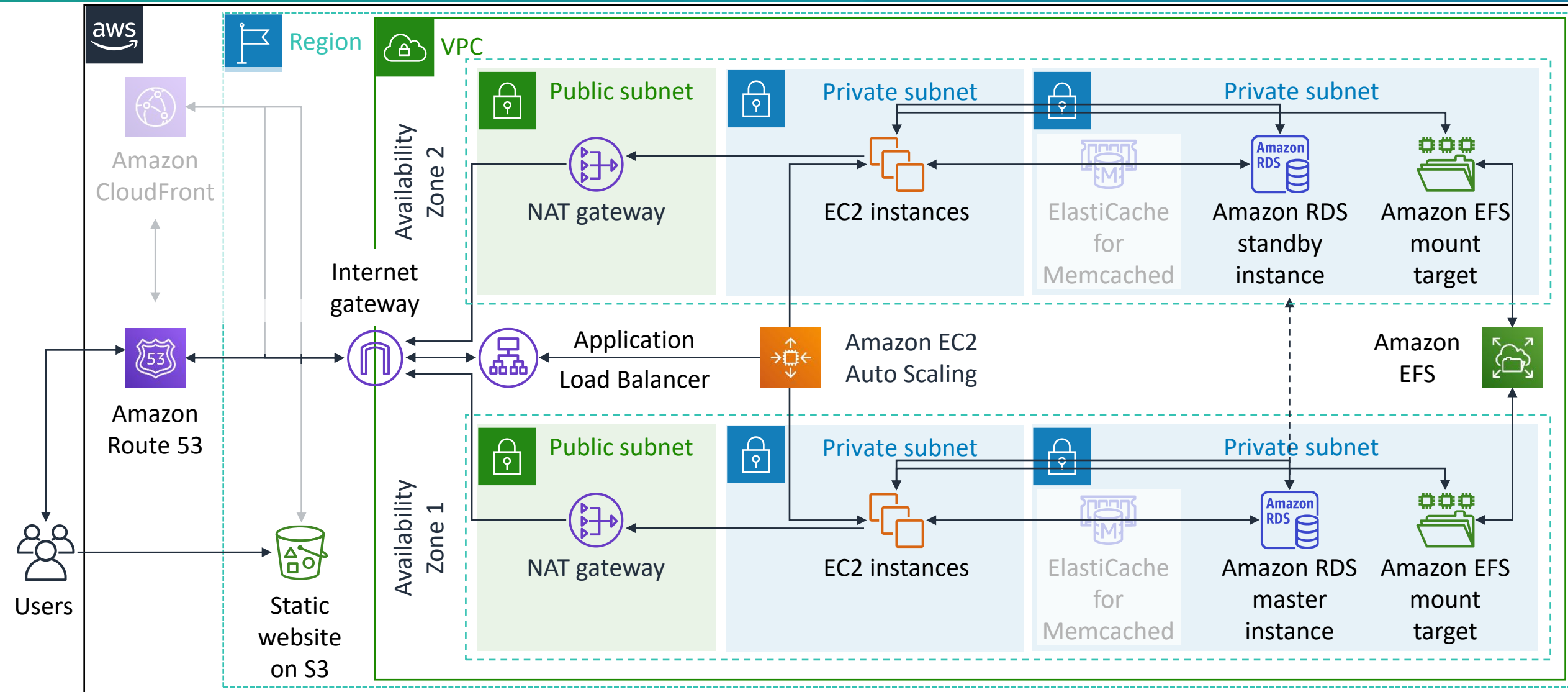
At the end of this module, you should be able to:

- Use Amazon EC2 Auto Scaling within an architecture to promote elasticity
- Explain how to scale your database resources
- Deploy an Application Load Balancer to create a highly available environment
- Use Amazon Route 53 for Domain Name System (DNS) failover
- Create a highly available environment
- Design architectures that use Amazon CloudWatch to monitor resources and react accordingly

## Module 9: Implementing Elasticity, High Availability, and Monitoring

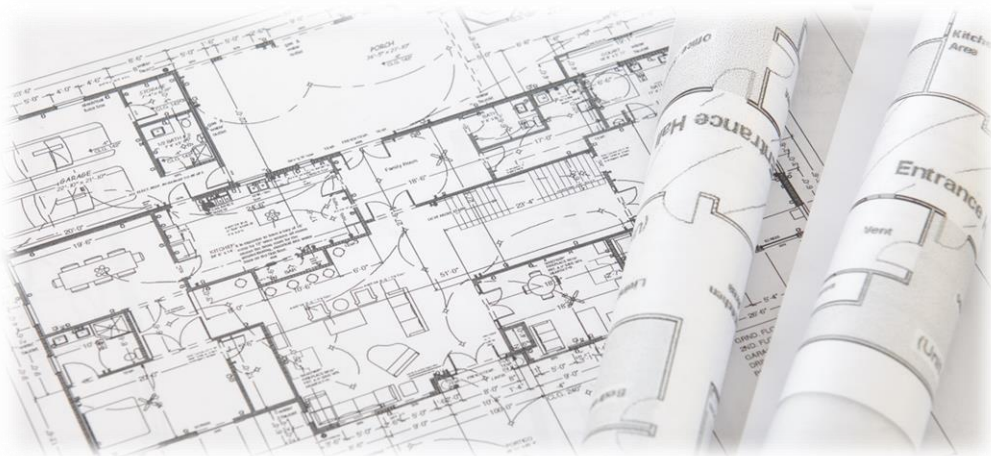
# Section 1: Architectural need

# Implementing high availability as part of a larger architecture



# Café business requirement

The café will be featured in a famous TV food show. When it airs, the architecture must handle significant increases in capacity.



# Reactive architectures



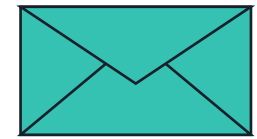
Elastic  
and scalable



Resilient



Responsive



Message-driven

Module 9: Implementing Elasticity, High Availability, and Monitoring

## Section 2: Scaling your compute resources



# What is elasticity?

An elastic infrastructure can **expand and contract** as capacity needs change.

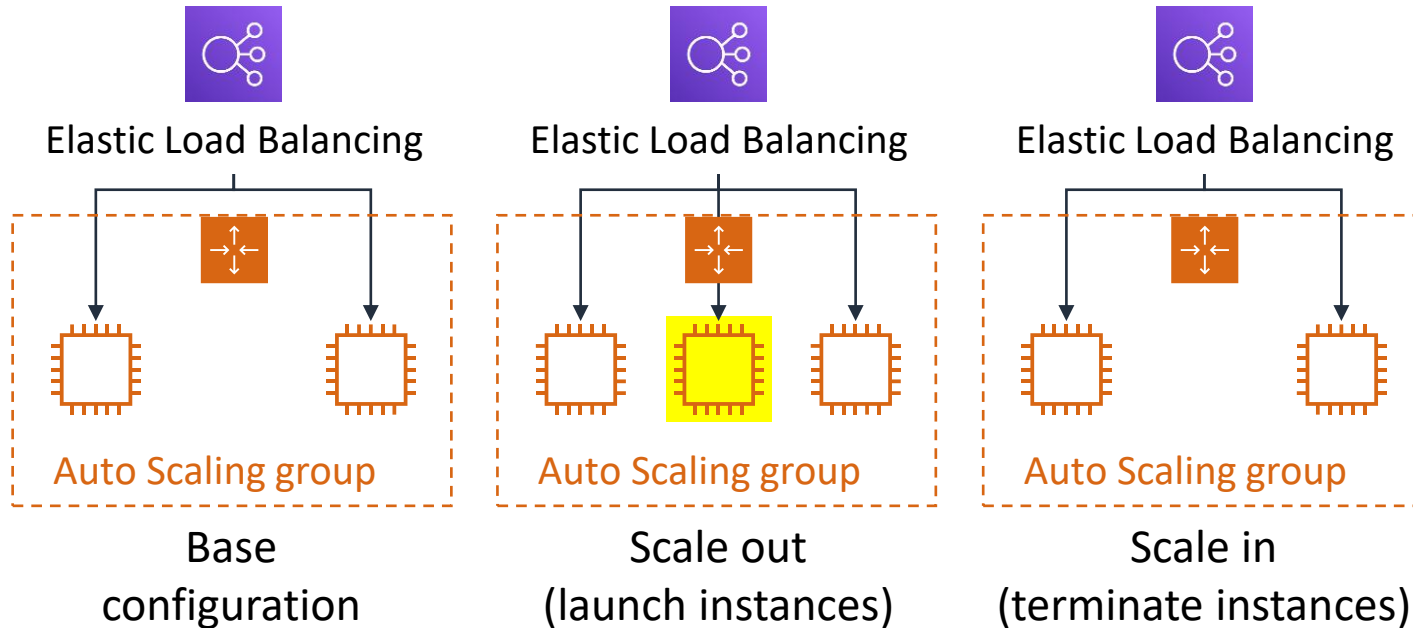
## Examples:

- Increasing the number of web servers when traffic spikes
- Lowering write capacity on your database when traffic goes down
- Handling the day-to-day fluctuation of demand throughout your architecture

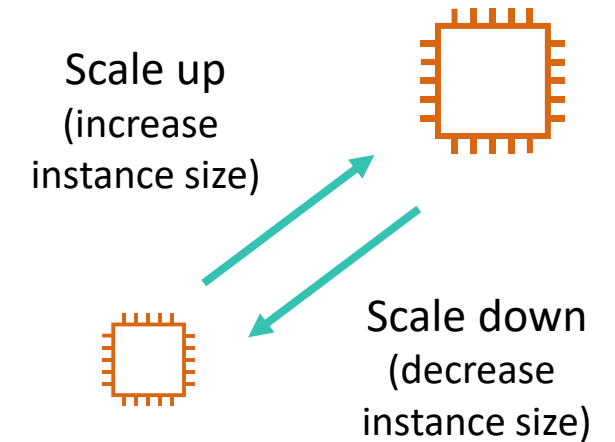
# What is scaling?

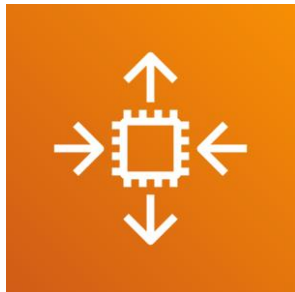
A technique that is used to achieve elasticity

## Horizontal scaling



## Vertical scaling





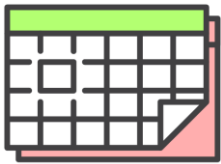
Amazon EC2  
Auto Scaling

- Launches or terminates instances based on specified conditions
- Automatically registers new instances with load balancers when specified
- Can launch across Availability Zones

# Scaling options

## Scheduled

Good for predictable workloads



Scale based on date and time

**Use case:** Turning off your development and test instances at night

## Dynamic

Good for changing conditions



Supports target tracking

**Use case:** Scaling based on CPU utilization

## Predictive

Good for predicted demand



Scale based on machine learning

**Use case:** Handling an increase in workload for ecommerce website during a major sales event

# Dynamic scaling policy types

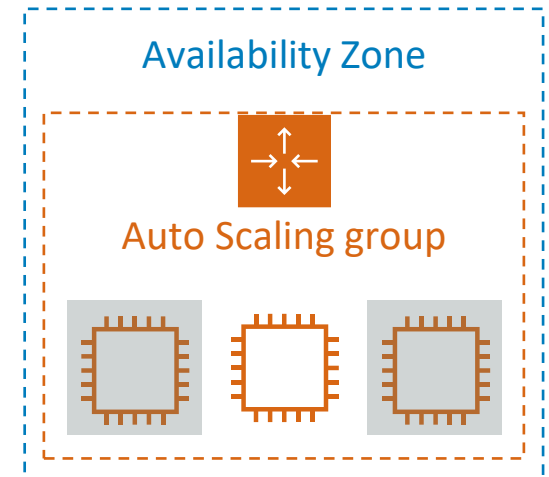
- **Simple scaling** – Single scaling adjustment
  - Example use cases: New workloads, spiky workloads
- **Step scaling** – Adjustment depends on size of alarm breach
  - Example use case: Predictable workloads
- **Target tracking scaling** – Target value for specific metric
  - Example use case: Horizontally scalable applications, such as load-balanced applications and batch data-processing applications

# Auto Scaling groups

An Auto Scaling group defines:

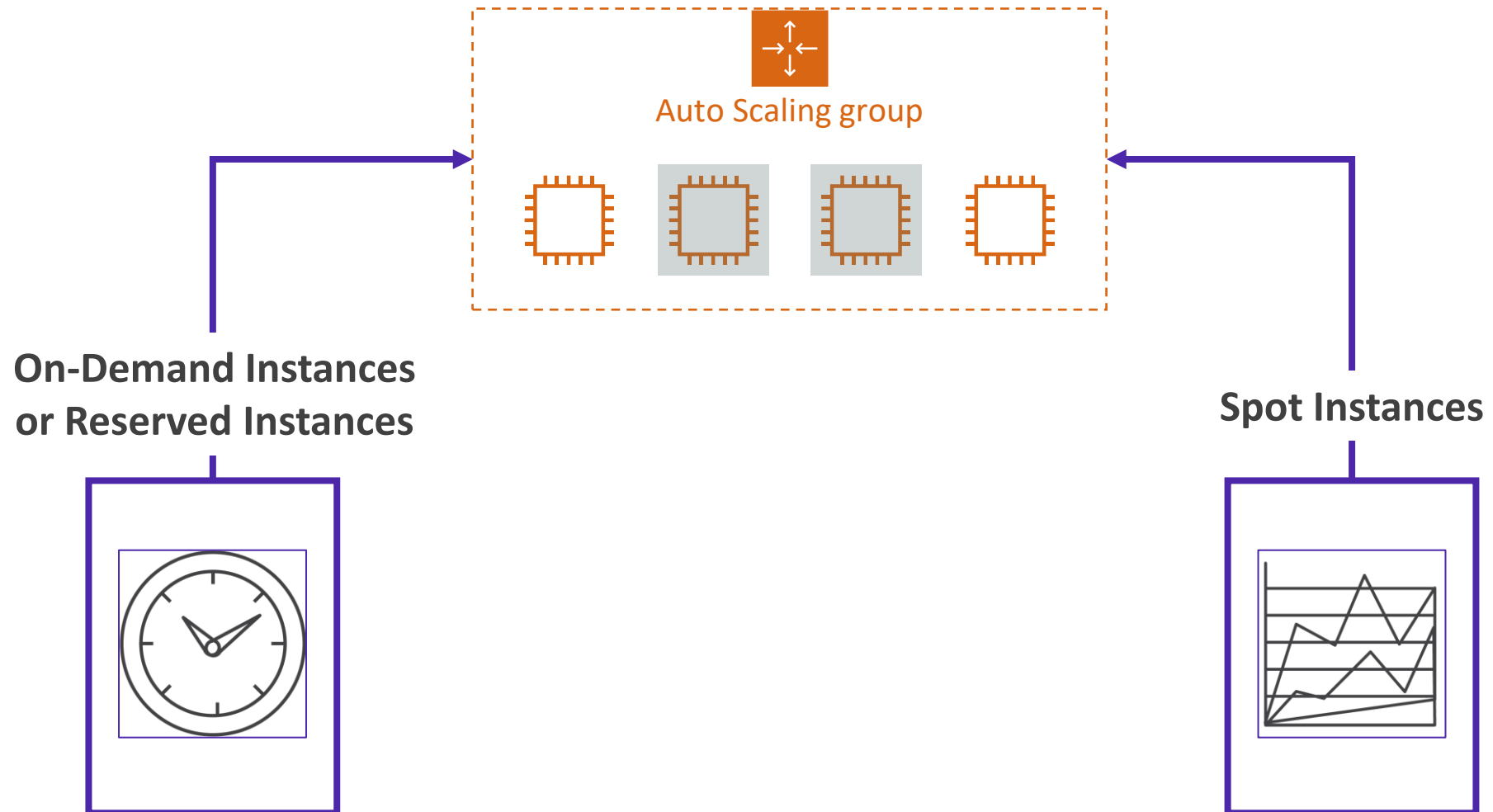
- Minimum capacity
- Maximum capacity
- Desired capacity\*

Capacity?



\*The **desired capacity** reflects the number of instances that are running and can fluctuate in response to events.

# Amazon EC2 Auto Scaling: Purchasing options



# Automatic scaling considerations

- Multiple types of automatic scaling
- Simple, step, or target tracking scaling
- Multiple metrics (not just CPU)
- When to scale out and scale in
- Use of lifecycle hooks



# Demonstration: Creating Scaling Policies for Amazon EC2 Auto Scaling



# Section 2 key takeaways



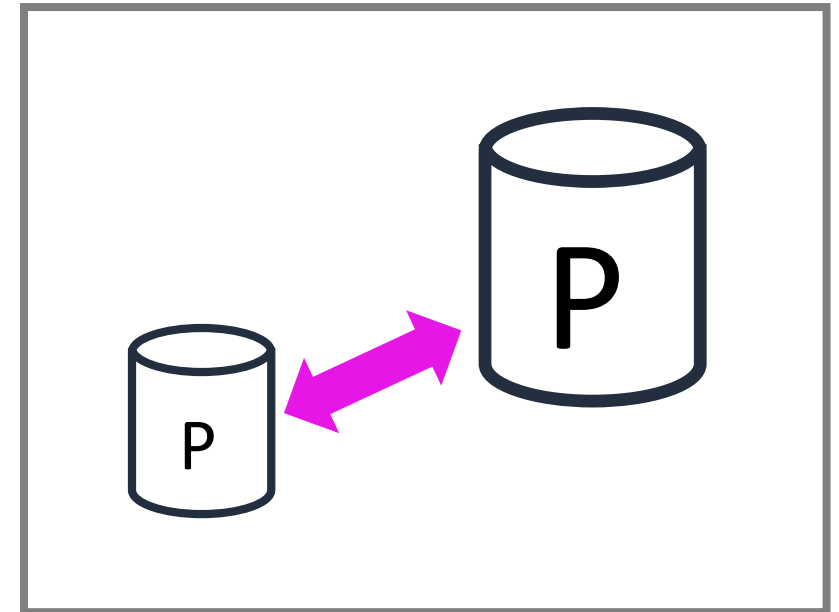
- An **elastic infrastructure** can expand and contract as capacity needs change
- **Amazon EC2 Auto Scaling** automatically adds or removes EC2 instances according to policies that you define, schedules, and health checks
- Amazon EC2 Auto Scaling provides **several scaling options** to best meet the needs of your applications
- When you configure an Auto Scaling group, you can specify the **EC2 instance types** and the combination of **pricing models** that it uses

Module 9: Implementing Elasticity, High Availability, and Monitoring

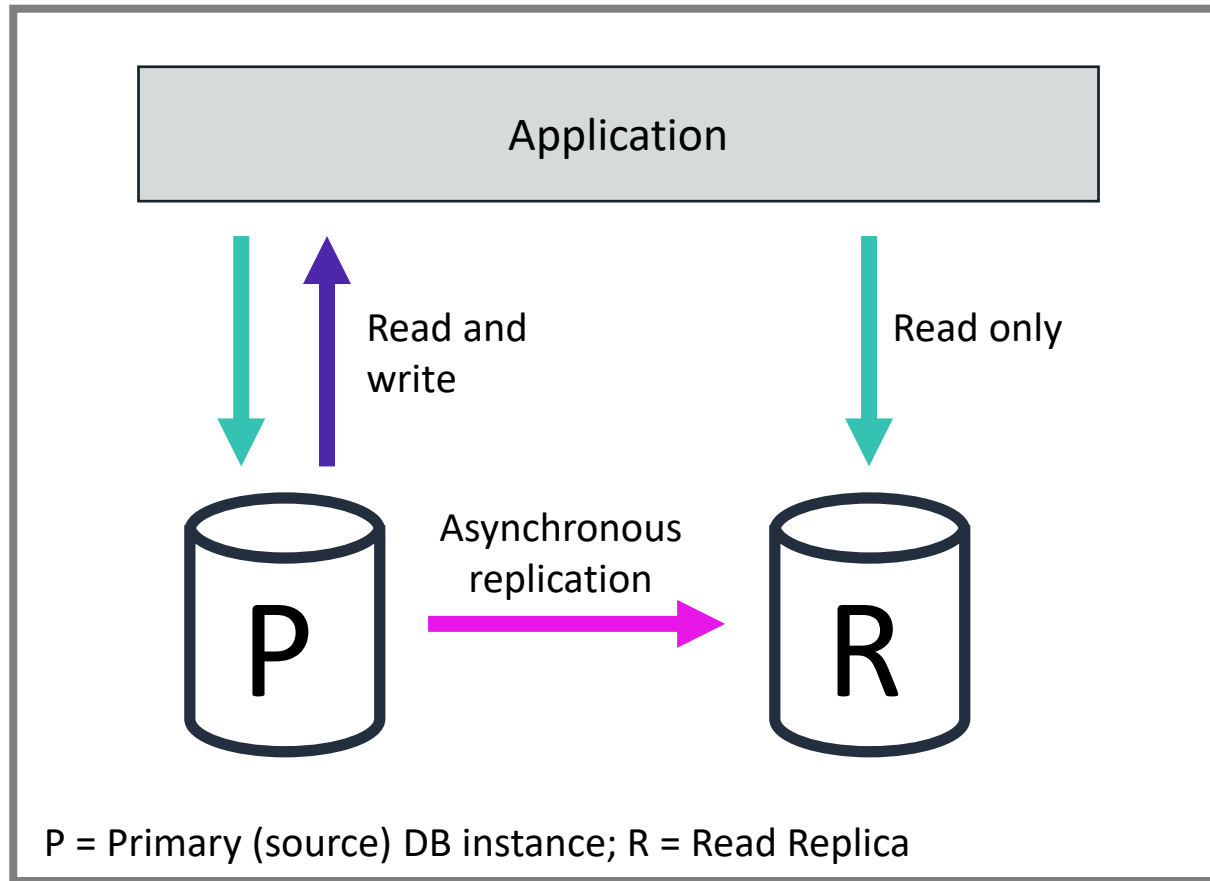
## Section 3: Scaling your databases

# Vertical scaling with Amazon RDS: Push-button scaling

- Scale DB instances **vertically** up or down
- From **micro** to **24xlarge** and everything in between
- Scale vertically with **minimal downtime**



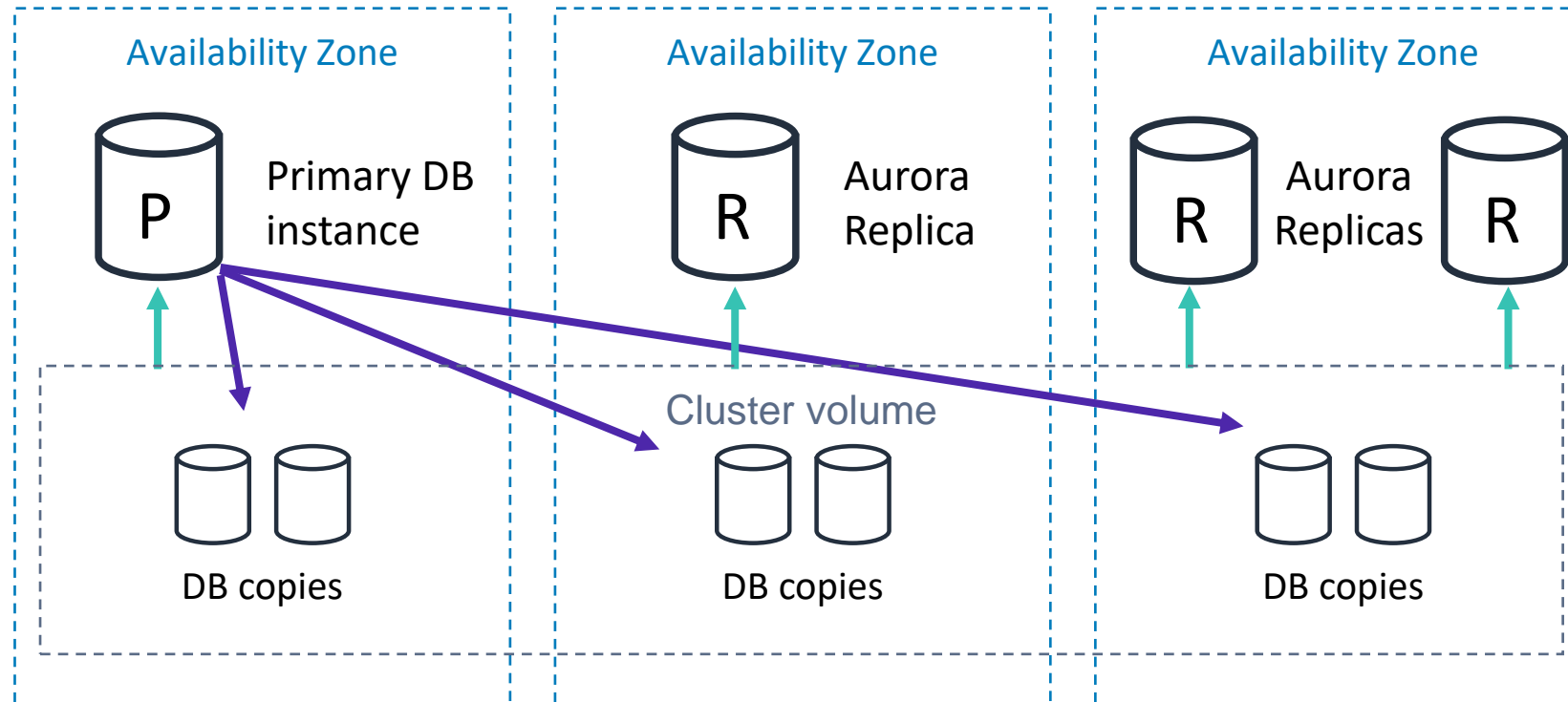
# Horizontal scaling with Amazon RDS: Read replicas



- Horizontally scale for **read-heavy** workloads
- Up to **five read replicas** and up to **15 Aurora replicas**
- Replication is **asynchronous**
- Available for Amazon RDS for MySQL, MariaDB, PostgreSQL, and Oracle

# Scaling with Amazon Aurora

**Each Aurora DB cluster can have up to 15 Aurora replicas**



# Amazon Aurora Serverless

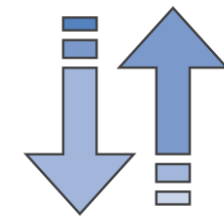


Responds to your application automatically:

- Scales capacity
- Starts up
- Shuts down



Pay for the number of Aurora capacity units (ACUs) that are used



Good for intermittent and unpredictable workloads

# Horizontal scaling: Database sharding

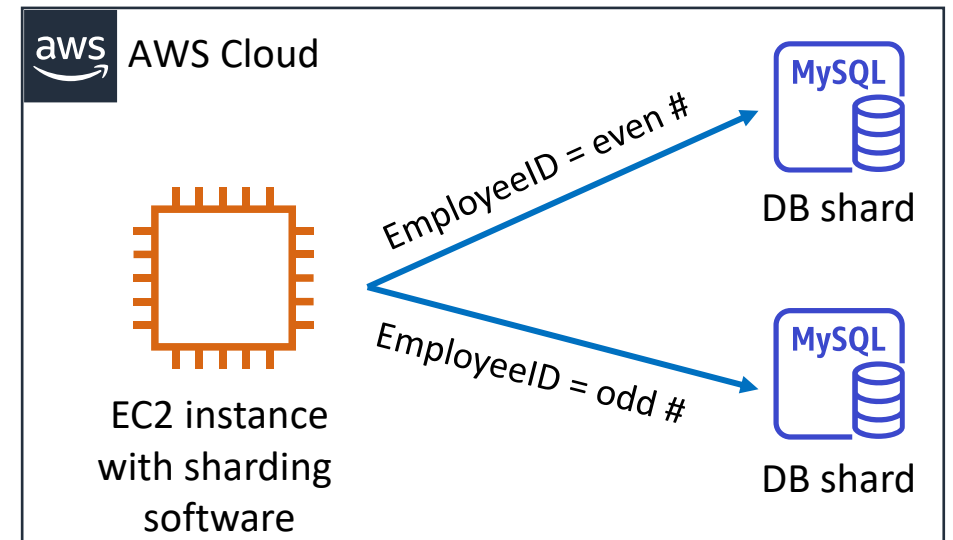
Without shards, all data resides in **one partition**.

- Example: Employee IDs in one database

With **sharding**, data is split into **large chunks** (shards).

- Example: Even-numbered employee IDs in one database, and odd-numbered employee IDs in another database

In many circumstances, sharding **improves write performance**.





# Scaling with Amazon DynamoDB: On-Demand

## On-Demand

---

Pay per request



No more provisioning

**Use case:** Spiky, unpredictable workloads.  
Rapidly accommodates to need.

---

# Scaling with Amazon DynamoDB: Auto scaling

## On-Demand

---

Pay per request



No more provisioning

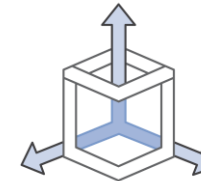
**Use case:** Spiky, unpredictable workloads.  
Rapidly accommodates to need.

---

## Auto scaling

---

Default for all new tables

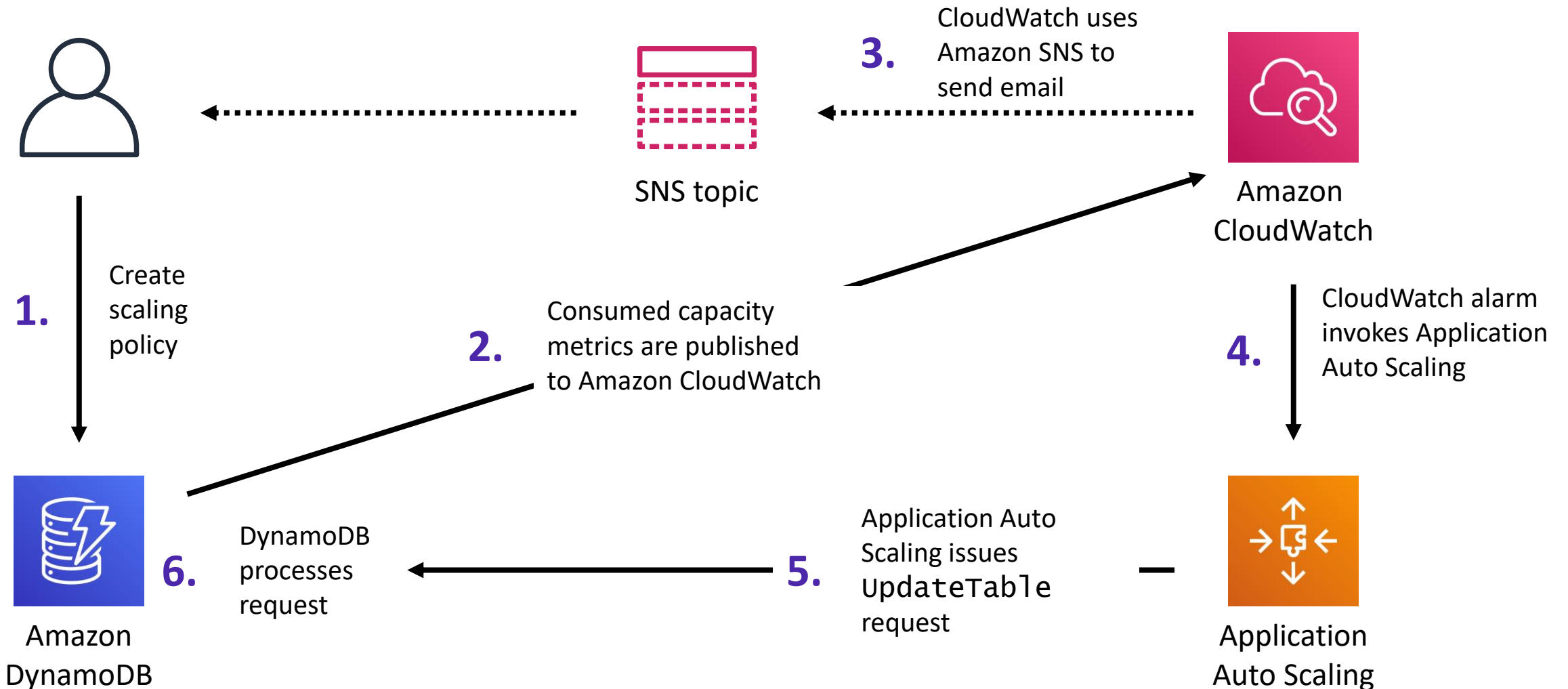


Specify upper and  
lower bounds

**Use case:** General scaling, good  
solution for most applications.

---

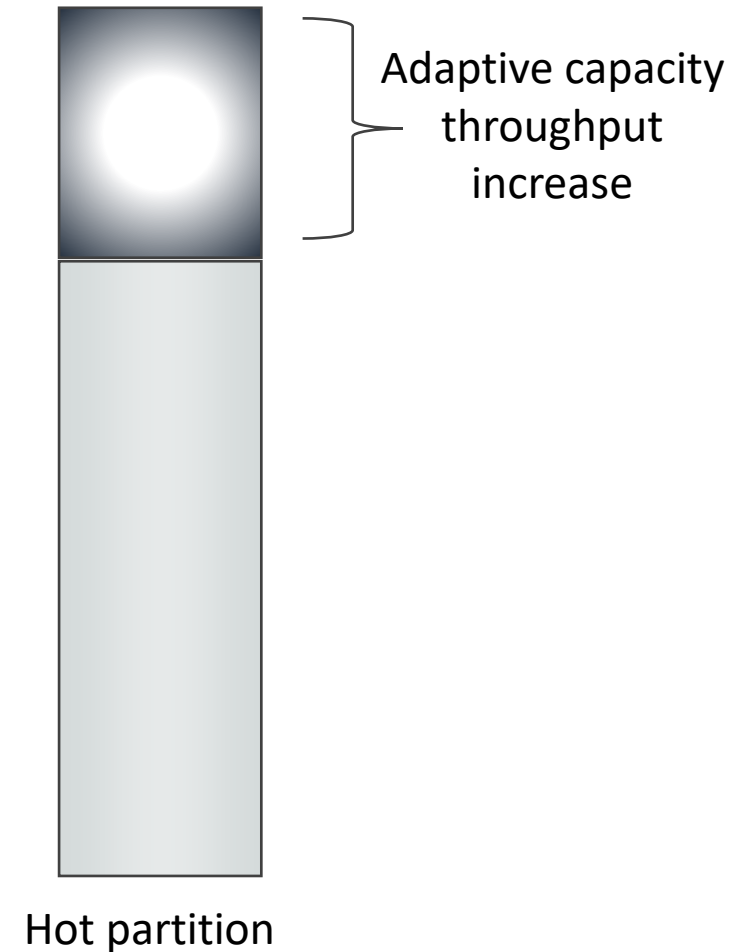
# How to implement DynamoDB auto scaling



# Scaling throughput capacity: DynamoDB adaptive capacity

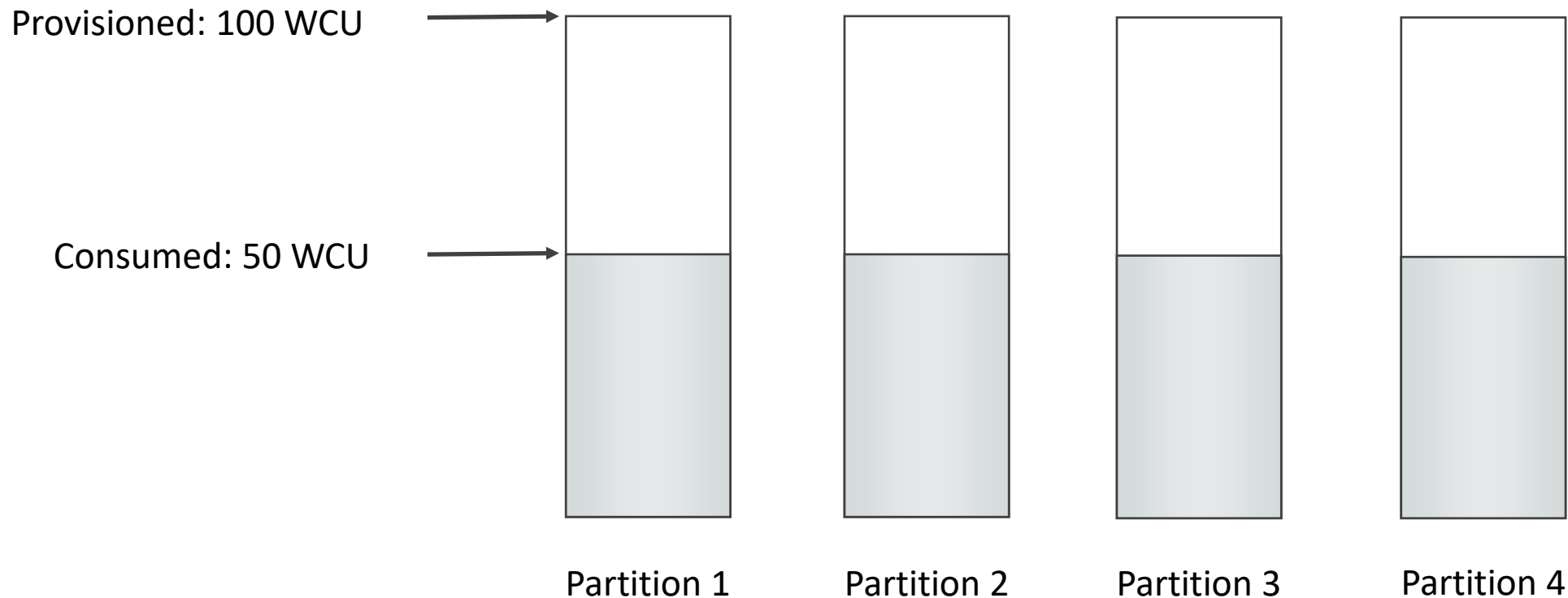
- Enables reading and writing to hot partitions **without throttling**
- **Automatically increases throughput capacity** for partitions that receive more traffic\*
- Is **enabled automatically** for every DynamoDB table

\*Traffic cannot exceed the table's total provisioned capacity or the partition's maximum capacity.



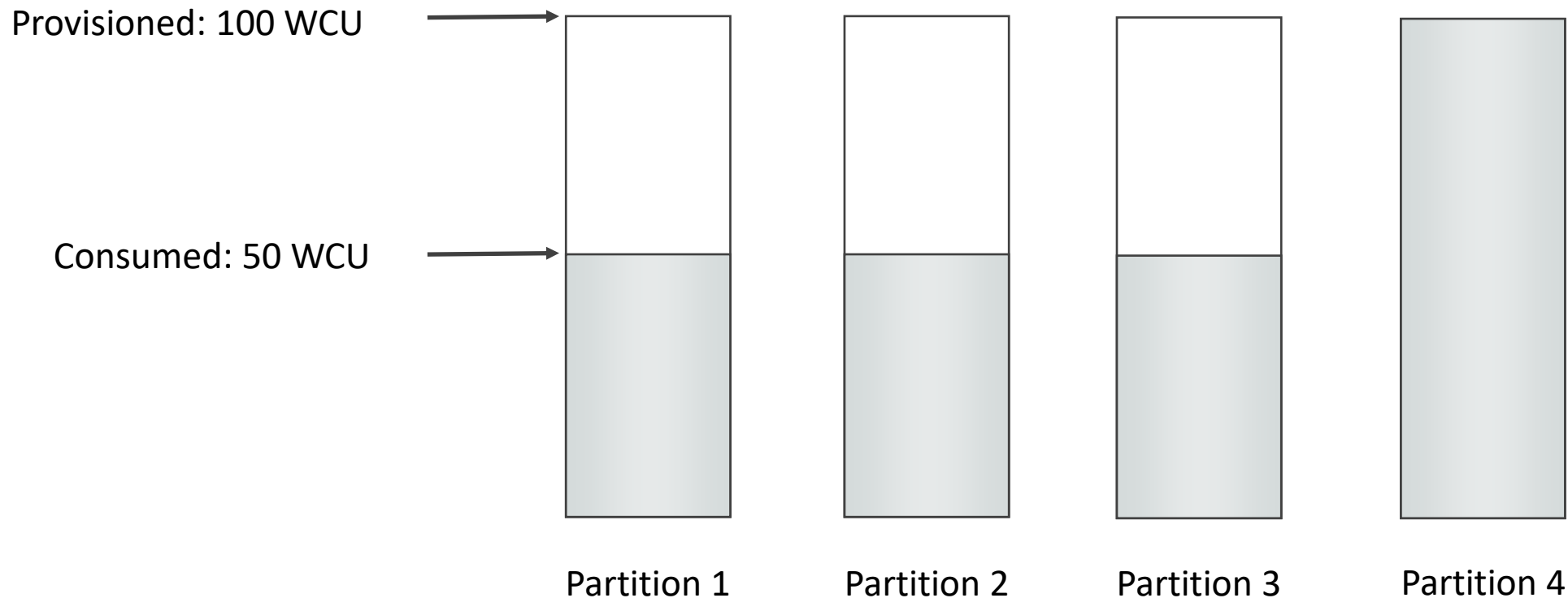
# Adaptive capacity example (1 of 3)

Example table with **adaptive** capacity  
Total provisioned capacity = 400 WCUs  
Total consumed capacity = 200 WCUs



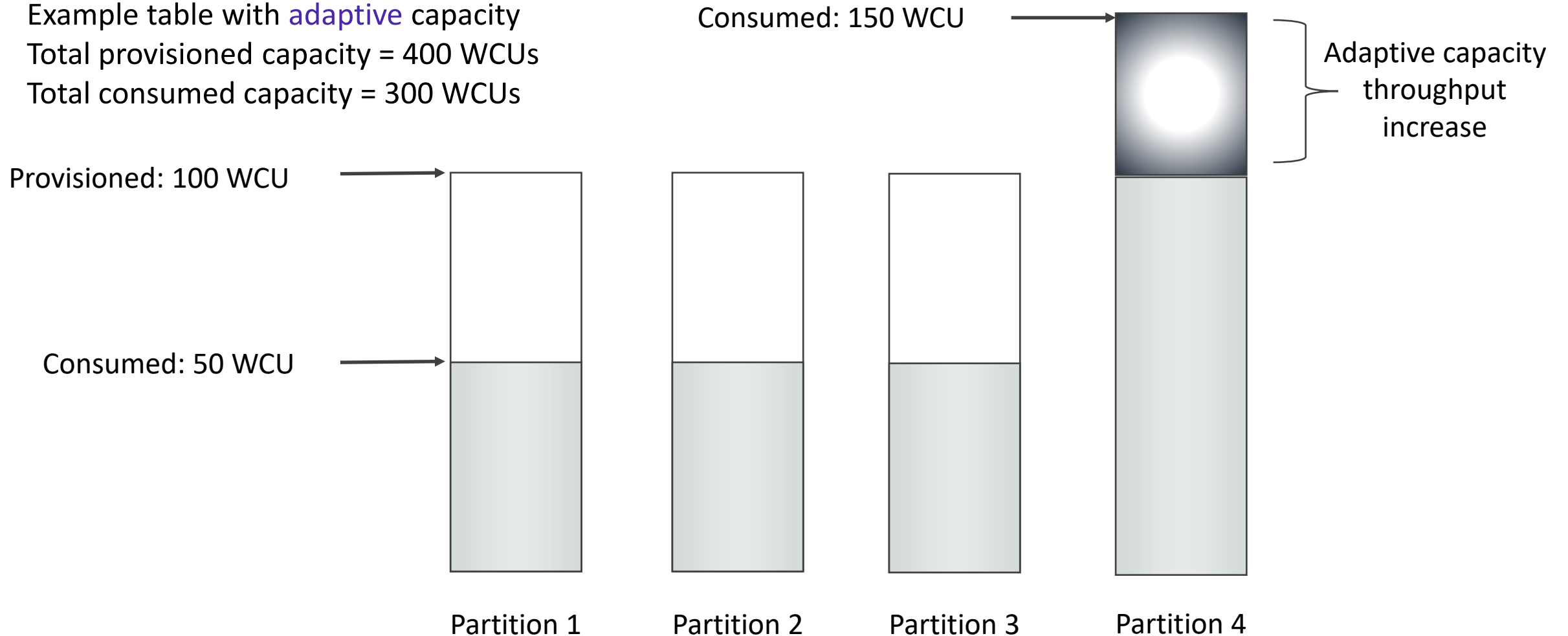
# Adaptive capacity example (2 of 3)

Example table with **adaptive** capacity  
Total provisioned capacity = 400 WCUs  
Total consumed capacity = 250 WCUs



# Adaptive capacity example (3 of 3)

Example table with **adaptive** capacity  
Total provisioned capacity = 400 WCUs  
Total consumed capacity = 300 WCUs



# Adaptive capacity does not fix hot keys and hot partitions

Partition key value	Uniformity
User ID, where the application has many users	Good
Status code, where there are only a few possible status codes	Bad
Item creation date, rounded to the nearest time period (for example, day, hour, or minute)	Bad
Device ID, where each device accesses data at relatively similar intervals	Good
Device ID, where even if many devices are tracked, one is much more popular than all the others	Bad



# Section 3 key takeaways



- You can use [push-button scaling](#) to vertically scale compute capacity for your RDS DB instance
- You can use [read replicas](#) or [shards](#) to horizontally scale your RDS DB instance
- With [Amazon Aurora](#), you can choose the DB instance class size and number of Aurora replicas (up to 15)
- [Aurora Serverless](#) scales resources automatically based on the minimum and maximum capacity specifications
- Amazon DynamoDB [On-Demand](#) offers a pay-per-request pricing model
- DynamoDB [auto scaling](#) uses Amazon Application Auto Scaling to dynamically adjust provisioned throughput capacity
- DynamoDB [adaptive capacity](#) works by automatically increasing throughput capacity for partitions that receive more traffic

## Module 9: Implementing Elasticity, High Availability, and Monitoring

# Section 4: Designing an environment that's highly available

# Highly available systems

- Can withstand some measure of degradation while remaining available
- Have minimized downtime
- Require minimal human intervention
- Recover from failure or roll over to secondary source in an acceptable amount of degraded performance time

Percentage of Uptime	Maximum Downtime Per Year	Equivalent Downtime Per Day
90%	36.5 days	2.4 hours
99%	3.65 days	14 minutes
99.9%	8.76 hours	86 seconds
99.99%	52.6 minutes	8.6 seconds
99.999%	5.25 minutes	0.86 seconds



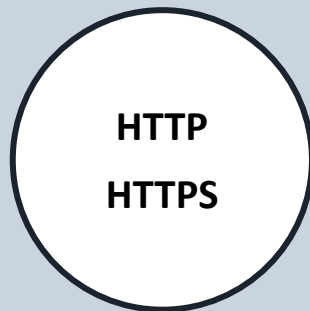
Elastic Load  
Balancing

A **managed load balancing service** that distributes incoming application traffic across multiple EC2 instances, containers, IP addresses, and Lambda functions.

- Can be **external-facing** or **internal-facing**
- Each load balancer receives a **DNS name**
- Recognizes and responds to **unhealthy instances**

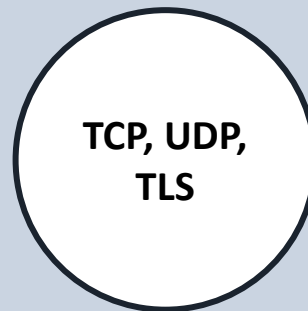
# Types of load balancers

## Application Load Balancer



- Flexible application management
- Advanced load balancing of HTTP and HTTPS traffic
- Operates at the request level (Layer 7)

## Network Load Balancer



- Ultra-high performance and static IP address for your application
- Load balancing of TCP, UDP, and TLS traffic
- Operates at the connection level (Layer 4)

## Classic Load Balancer

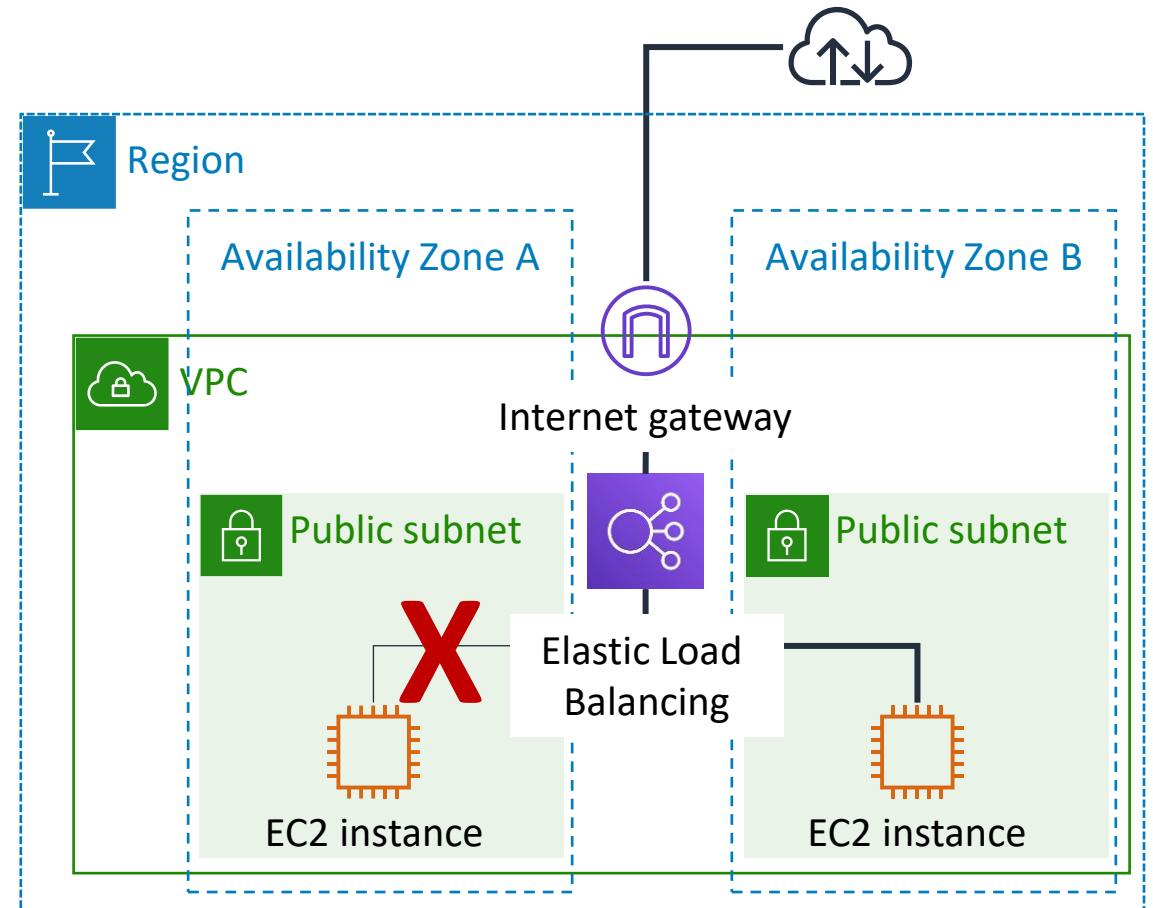
PREVIOUS GENERATION  
for HTTP, HTTPS, TCP, and SSL

- Load balancing across multiple EC2 instances
- Operates at both the request level and connection level

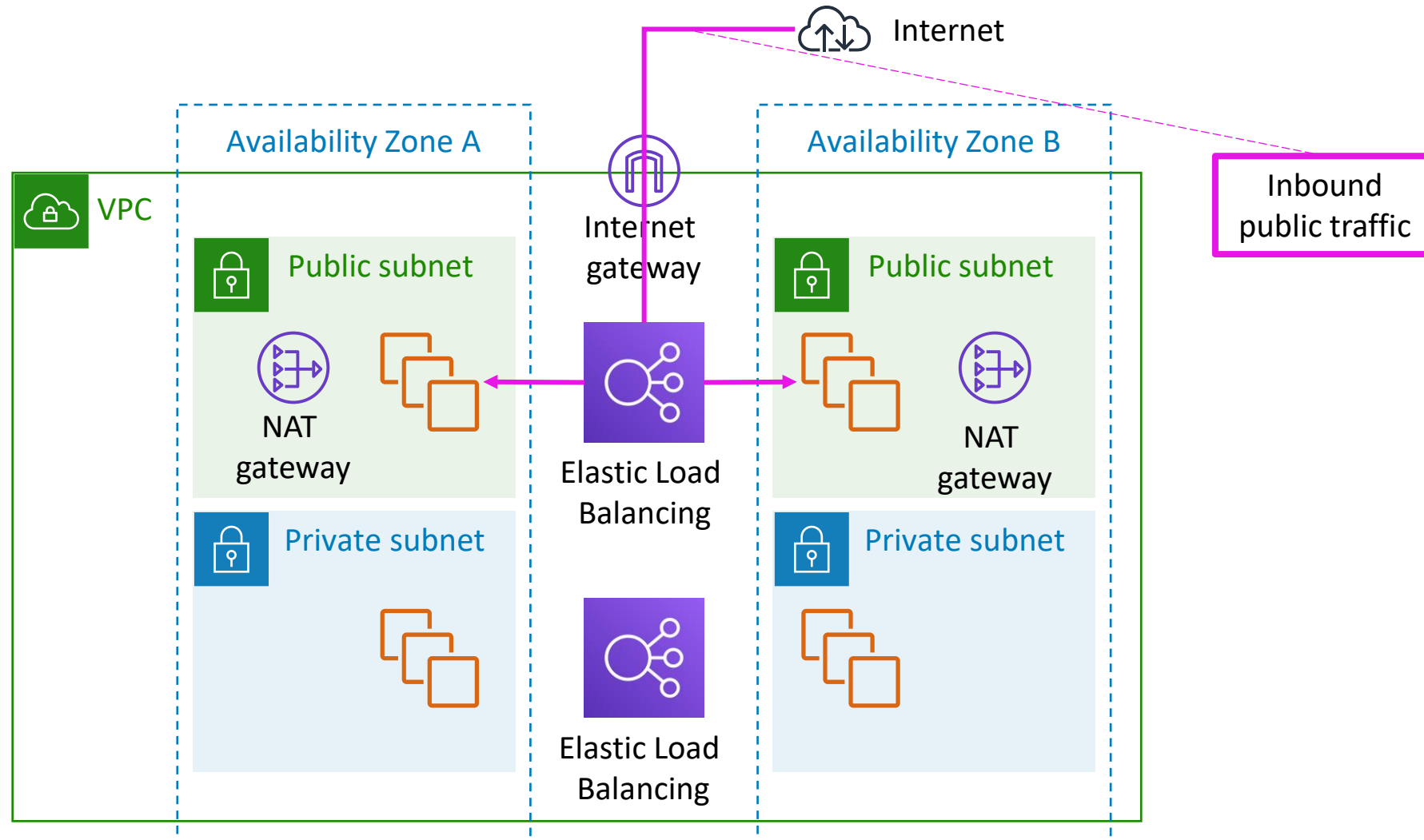
# Implementing high availability

Start with two Availability Zones per AWS Region.

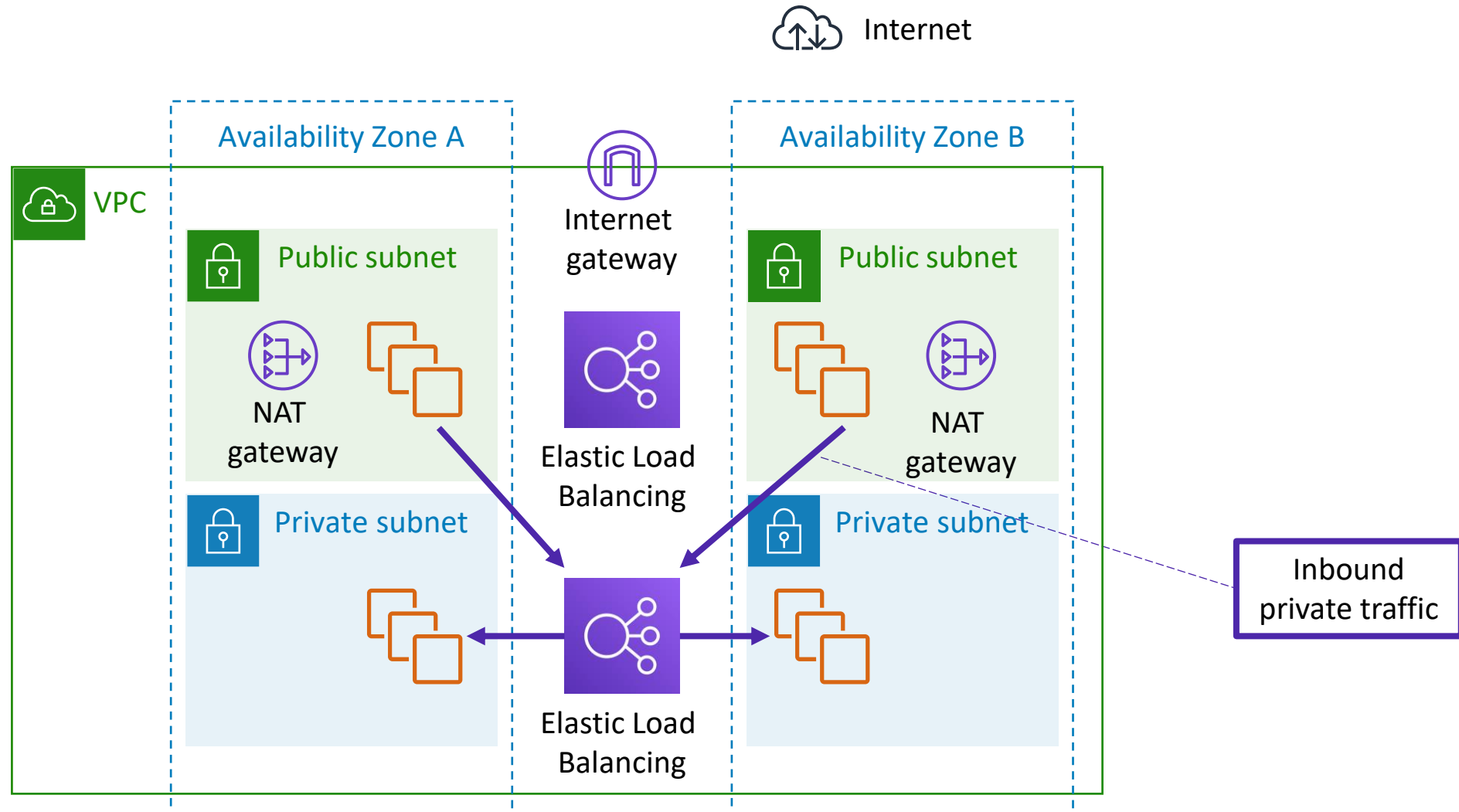
If resources in one Availability Zone are unreachable, your application shouldn't fail.



# Example of a highly available architecture (1 of 3)

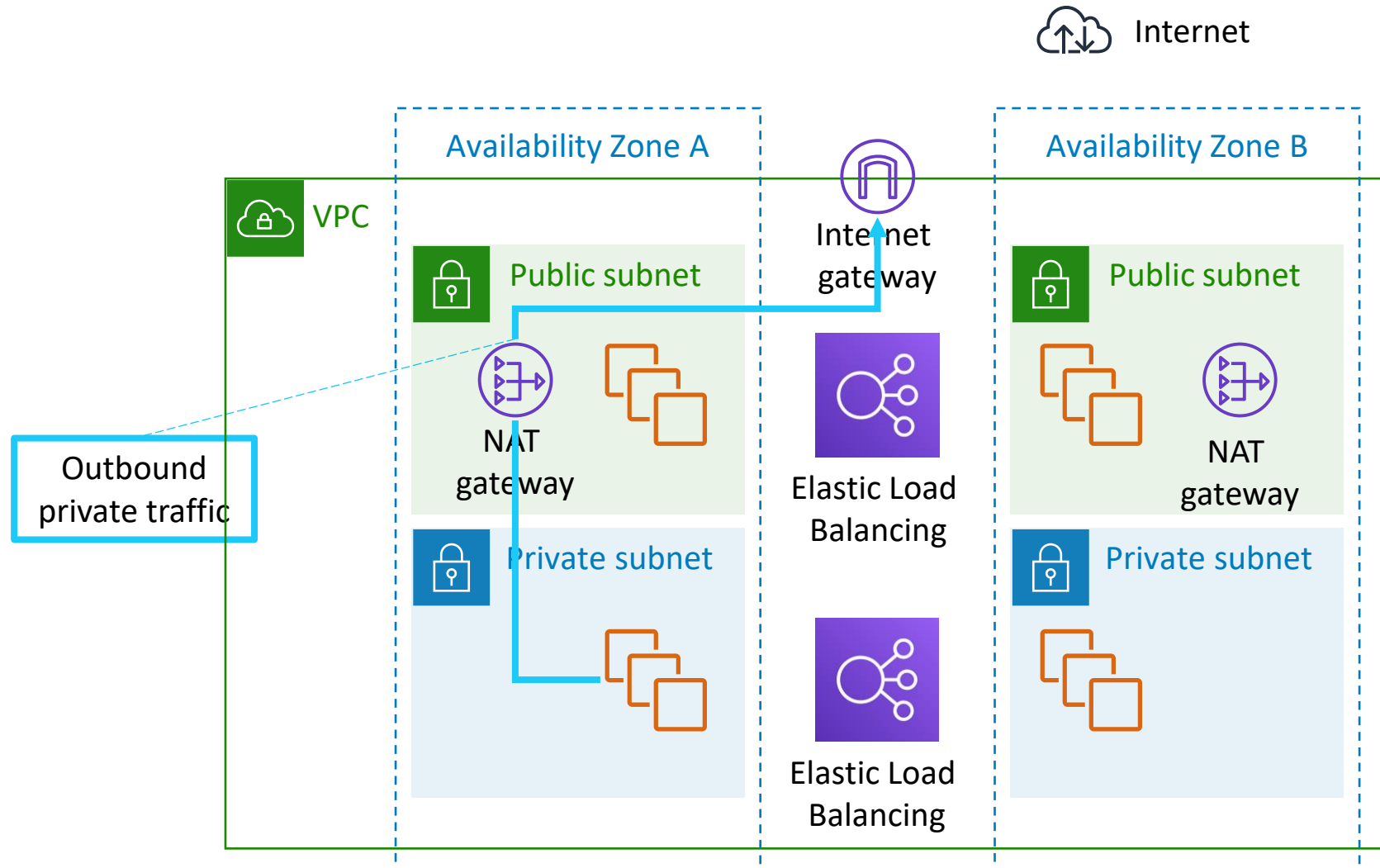


# Example of a highly available architecture (2 of 3)





# Example of a highly available architecture (3 of 3)



# Demonstration: Creating a Highly Available Web Application





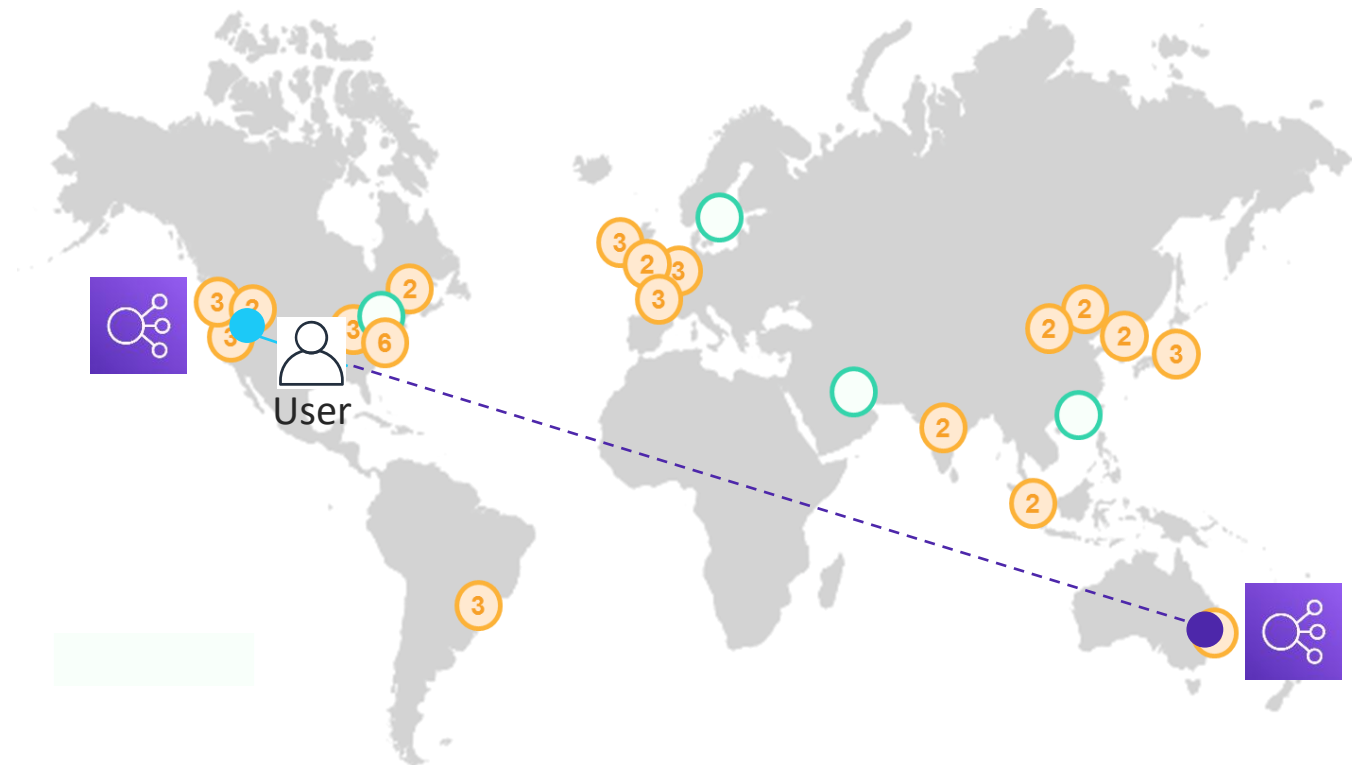
Amazon Route  
53

Amazon Route 53 is a highly available and scalable cloud DNS service.

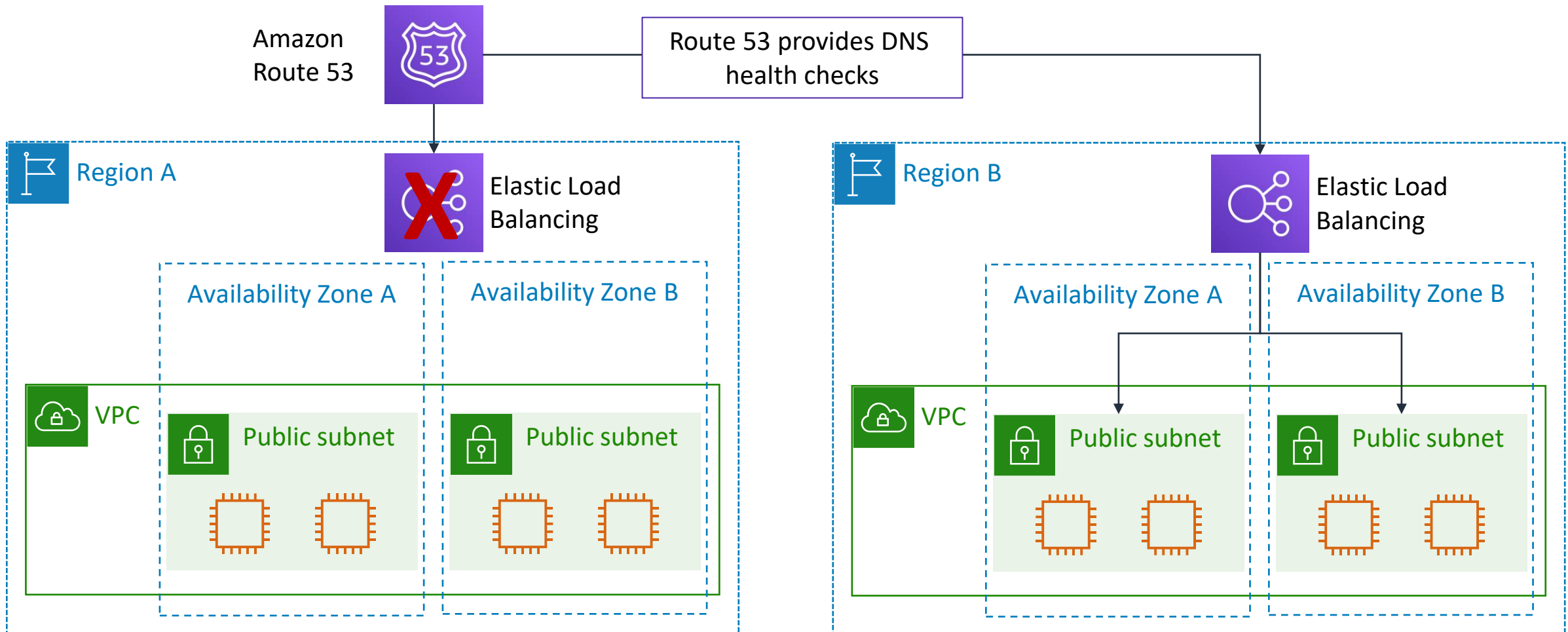
- Translates domain names into IP addresses
- Connects user requests to infrastructure that runs inside and outside of AWS
- Can be configured to route traffic to healthy endpoints, or to monitor the health of your application and its endpoints
- Offers registration for domain names
- Has multiple routing options

# Amazon Route 53 supported routing

- Simple routing
- Weighted round robin routing
- Latency-based routing
- Geolocation routing
- Geoproximity routing
- Failover routing
- Multivalue answer routing



# Multi-Region high availability and DNS



# Demonstration: Amazon Route 53



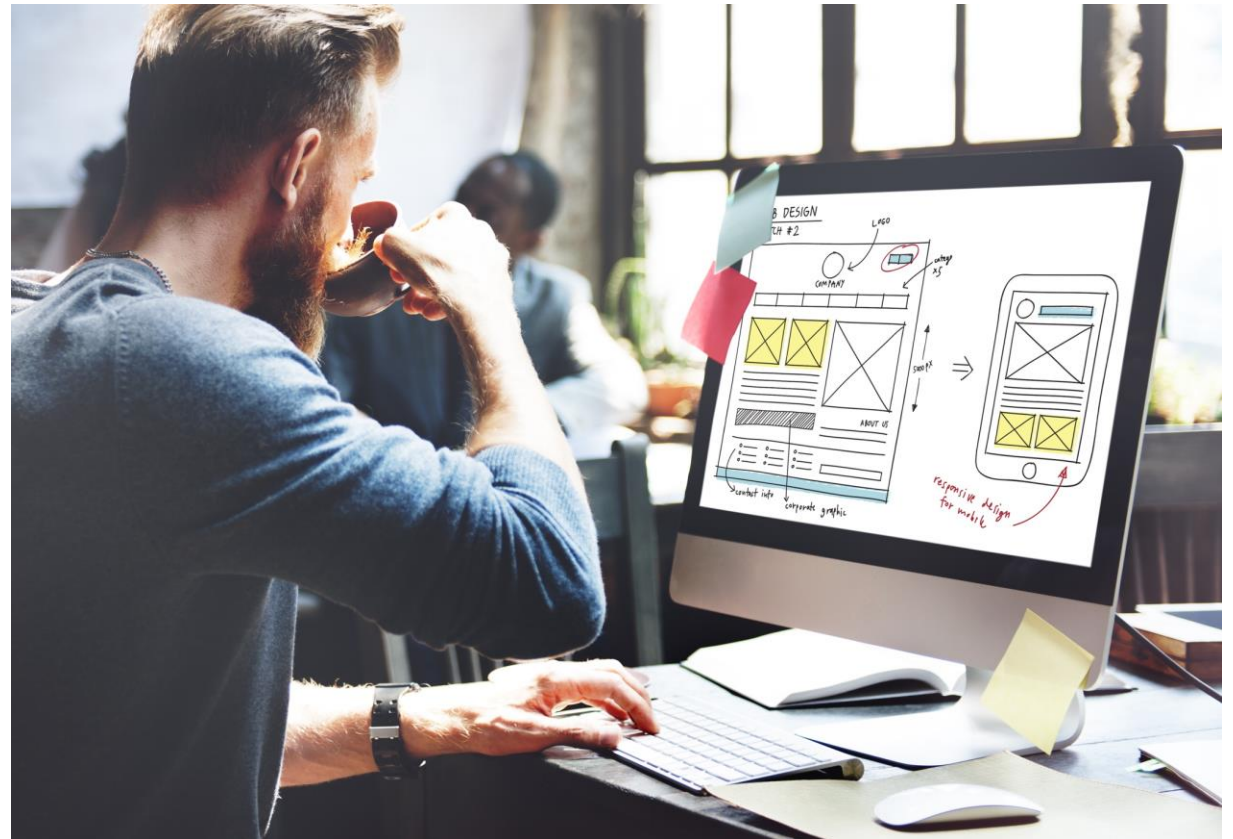


## Section 4 key takeaways



- You can design your network architectures to be highly available and avoid single points of failure
- Route 53 offers various routing options that can be combined with DNS failover to enable low-latency, fault-tolerant architectures

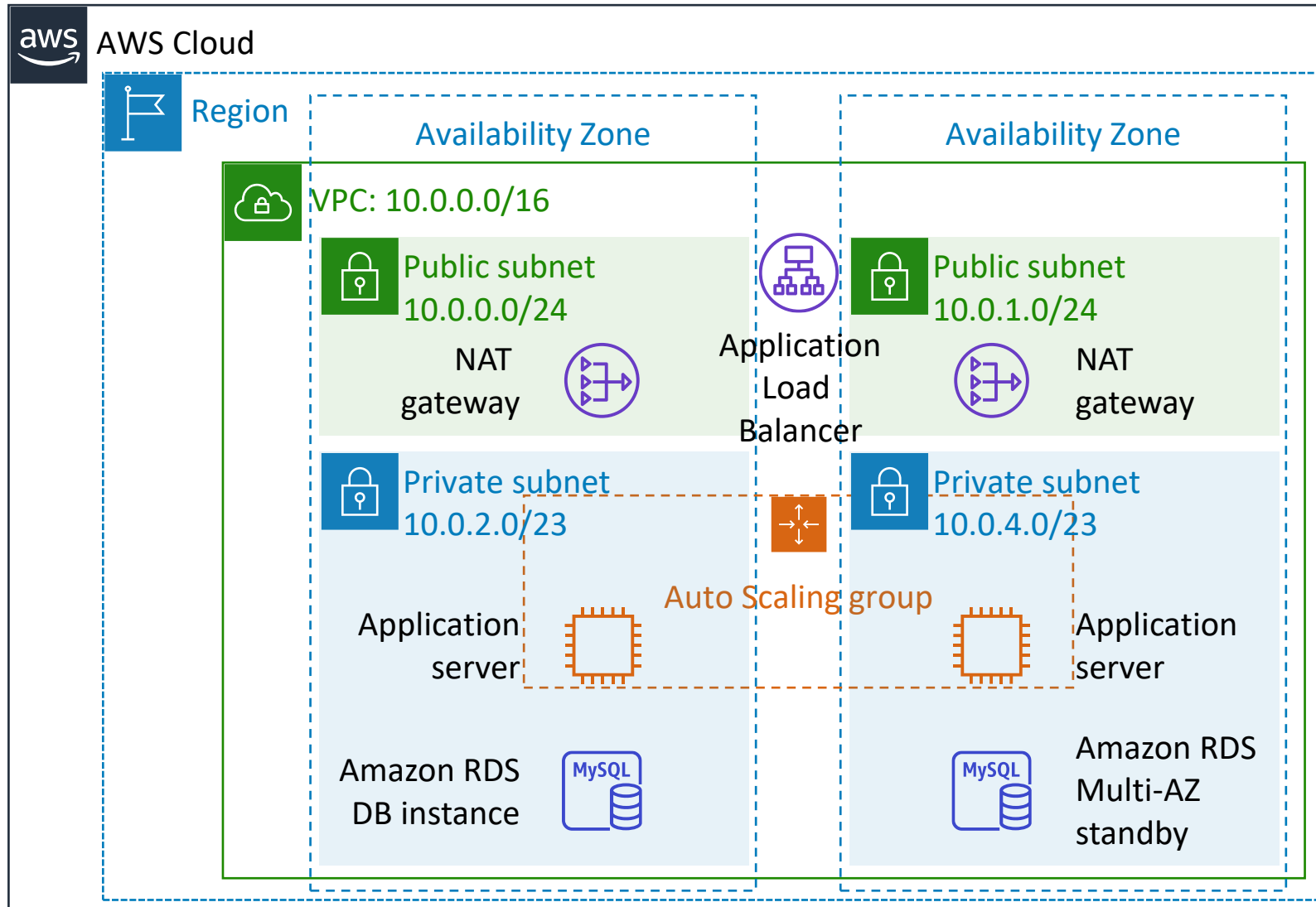
# Module 9 – Guided Lab: Creating a Highly Available Environment

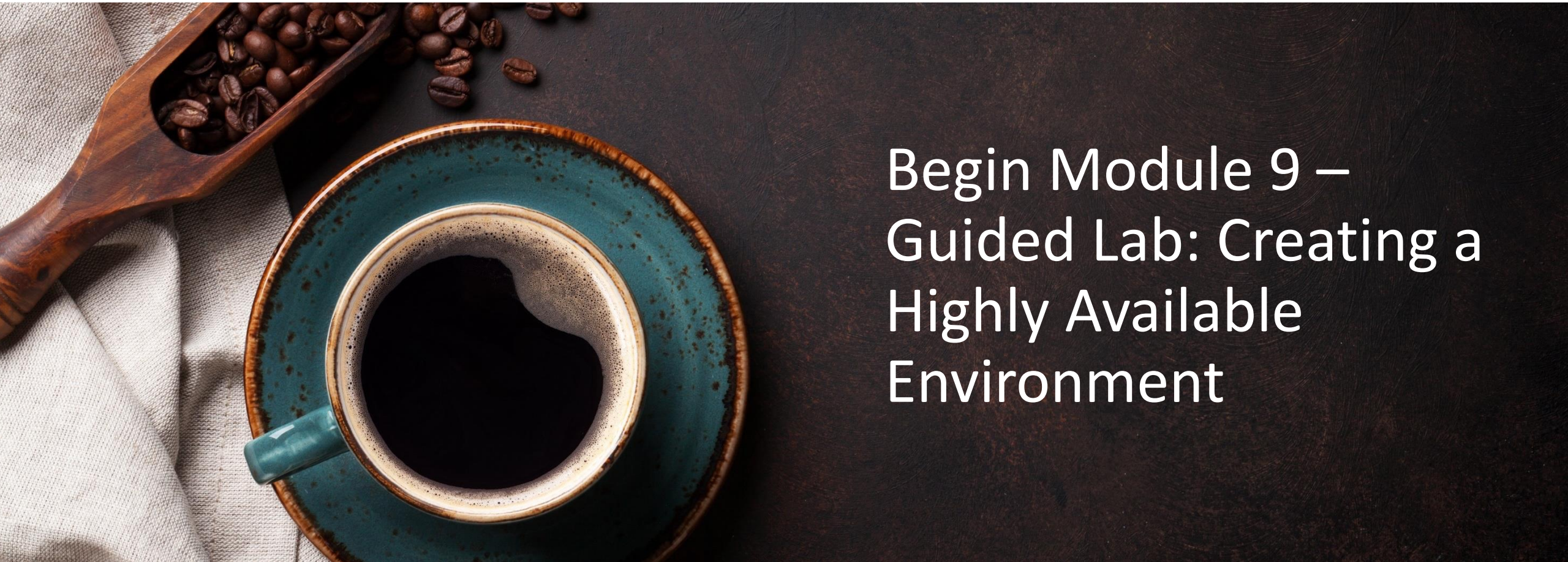




1. Inspect a provided VPC
2. Create an Application Load Balancer
3. Create an Auto Scaling group
4. Test the application for high availability

# Guided lab: Final product





# Begin Module 9 – Guided Lab: Creating a Highly Available Environment



# Guided lab debrief: Key takeaways



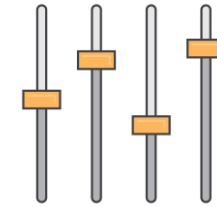
**Module 9: Implementing Elasticity, High Availability, and Monitoring**

## Section 5: Monitoring

# Monitoring usage, operations, and performance



Operational Health



Resource Utilization



Application Performance



Security Auditing

# Monitoring your costs

To create a more flexible and elastic architecture, you should **know where you are spending money**.

AWS Cost Explorer



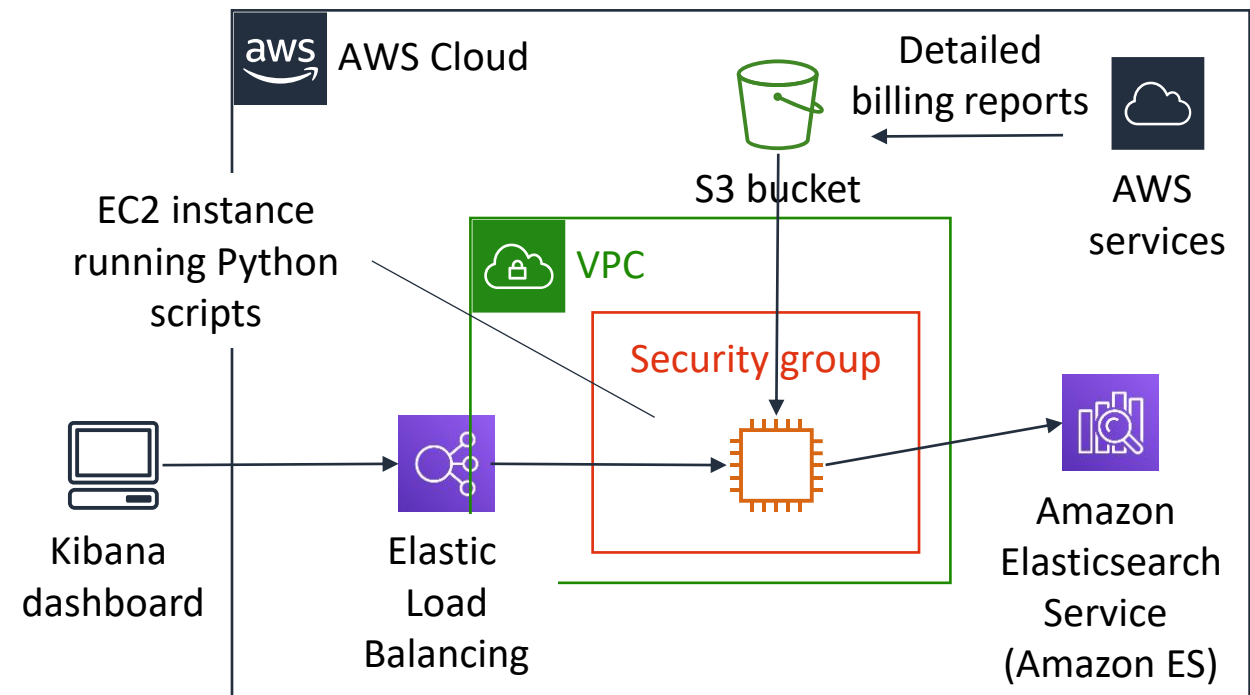
AWS Budgets

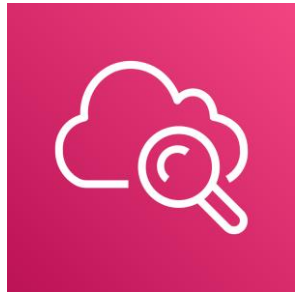


AWS Cost and Usage Report



## Cost Optimization Monitor





Amazon  
CloudWatch

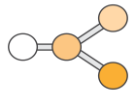
- Collects and tracks metrics for your resources and applications
- Helps you correlate, visualize, and analyze metrics and logs
- Enables you to create alarms and detect anomalous behavior
- Can send notifications or make changes to resources that you are monitoring



# How CloudWatch responds



Metrics



Logs



Alarms



Events



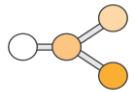
Rules



Targets



Metrics



Logs



Alarms



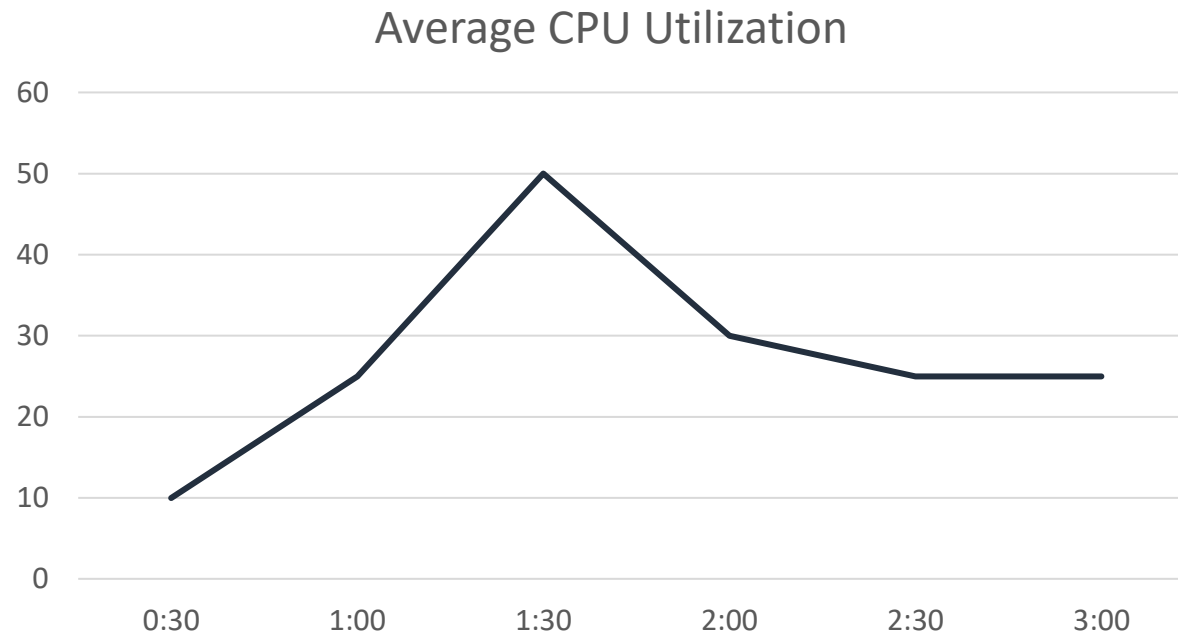
Events



Rules



Targets

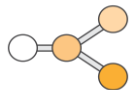


Metric data is kept for 15 months

# Amazon CloudWatch Logs



Metrics



Logs



Alarms



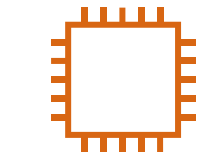
Events



Rules



Targets



Application

Log\_File.txt

Errors: 3

Warnings: 12

Connections: 20

Print out...



Amazon  
CloudWatch



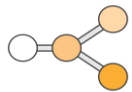
Amazon S3

Source examples

- VPC Flow Logs
- Amazon Route 53
- Elastic Load Balancing access logs



Metrics



Logs



Alarms



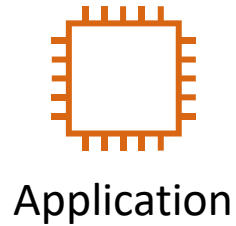
Events



Rules



Targets



CPUUtilization metric

80% 60% 45% 25% 10% 10% 10% 10% 5%

Alarm

If CPUUtilization metric is > 50% for 5 minutes

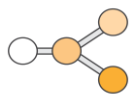
Trigger an action like:

- Send a notification to the development team
- Create another instance to handle the load

# Amazon EventBridge events



Metrics



Logs



Alarms



Events

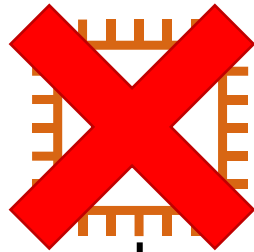


Rules



Targets

Event: EC2  
instance  
termination



Amazon  
EventBridge

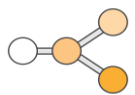
## Event examples

- Change in AWS resource, such as –
  - Console sign-in
  - EC2 instance state change
  - EC2 Auto Scaling state change
  - EBS volume creation
- AWS API call
- Events from SaaS partners
- Events from your own applications

# Amazon EventBridge rules



Metrics



Logs



Alarms



Events

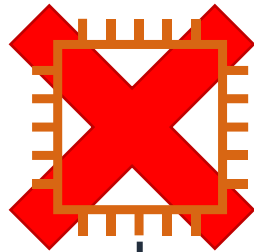


Rules



Targets

Event



Rule example

```
{
  "source": [
    "aws.ec2"
  ],
  "detail-type": [
    "EC2 Instance State-change Notification"
  ],
  "detail": {
    "state": [
      "terminated" ]
  }
}
```

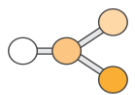


Amazon  
EventBridge

# Amazon EventBridge targets



Metrics



Logs



Alarms



Events

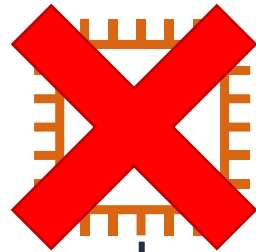


Rules



Targets

Event



Rule example

```
{
  "source": [ "aws.ec2" ],
  "detail-type": [ "EC2
Instance State-change
Notification" ],
  "detail": {
    "state": [ "terminated" ]
  }
}
```

Target examples

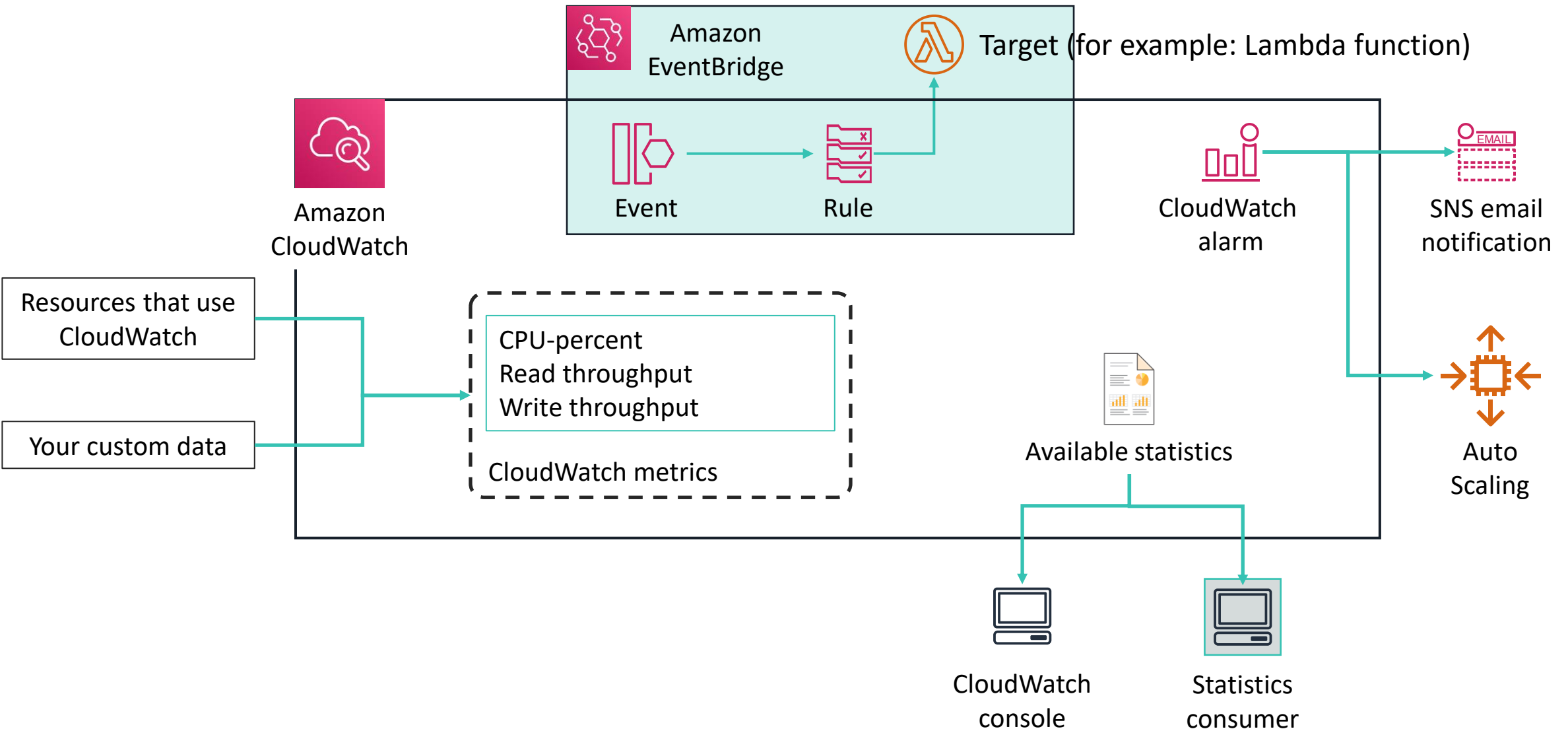
- EC2 instances
- AWS Lambda
- Kinesis streams
- Amazon ECS
- Step Functions
- Amazon SNS
- Amazon SQS



Amazon  
EventBridge



# How CloudWatch and EventBridge work





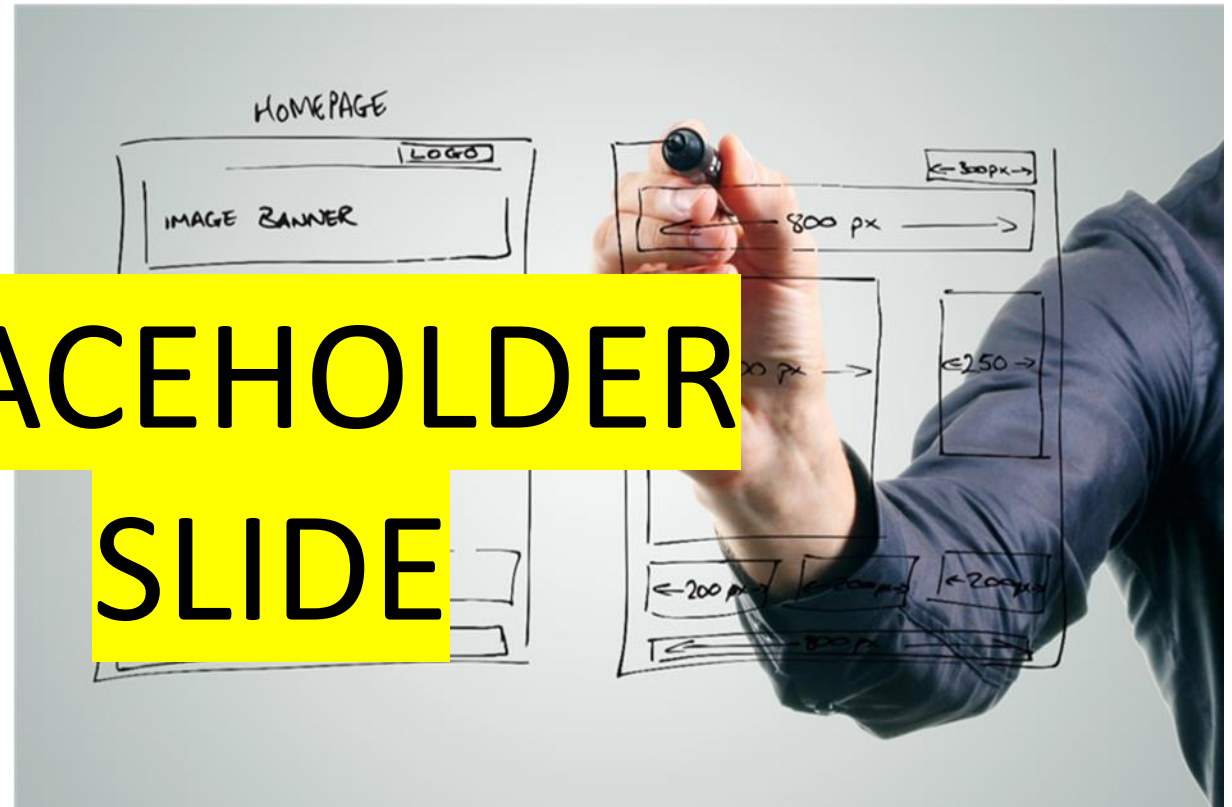
# Section 5 key takeaways



- [AWS Cost Explorer](#), [AWS Budgets](#), [AWS Cost and Usage Report](#), and the [Cost Optimization Monitor](#) can help you understand and manage the [cost of your AWS infrastructure](#).
- [CloudWatch](#) collects monitoring and operational data in the form of logs, metrics, and events. It visualizes the data by using automated dashboards so you can get a unified view of your AWS resources, applications, and services that run in AWS and on-premises.
- [EventBridge](#) is a serverless event bus service that connects your applications with data from various sources. EventBridge delivers a stream of real-time data from your own applications, SaaS applications, and AWS services. It then routes that data to targets.

# Module 9 – Challenge Lab: <Exact Lab Title>

PLACEHOLDER  
SLIDE



# The business need: <summary>

<if applicable, otherwise delete this slide>

<tie the service or technical concept into whatever example use case the challenge labs revolve around>

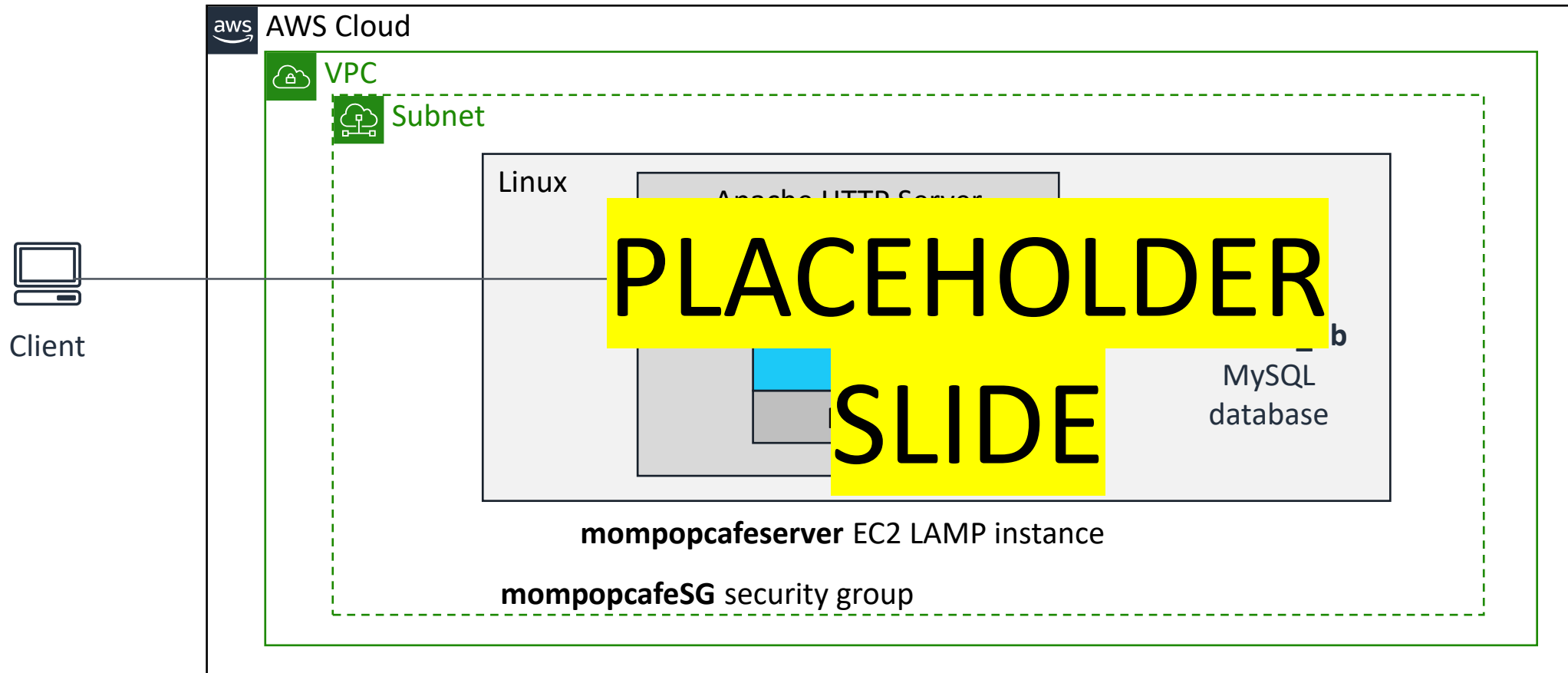
**PLACEHOLDER  
SLIDE**

# Challenge lab: Customer requirements

- <Customer requirement #>

**PLACEHOLDER  
SLIDE**

# Challenge lab: Final product







~ 45 minutes

PLACEHOLDER

SLIDE

Rule 9 –  
Challenge Lab: <Exact  
Title>



# Challenge lab debrief: Key takeaways



## Module 9: Implementing Elasticity, High Availability, and Monitoring

# Module wrap-up



In summary, in this module, you learned how to:

- Use Amazon EC2 Auto Scaling within an architecture to promote elasticity
- Explain how to scale your database resources
- Deploy an Application Load Balancer to create a highly available environment
- Use Amazon Route 53 for DNS failover
- Create a highly available environment
- Design architectures that use Amazon CloudWatch to monitor resources and react accordingly

# Complete the knowledge check



# Sample exam question

A web application enables customers to upload orders to an S3 bucket. The resulting Amazon S3 events trigger a Lambda function that inserts a message into an SQS queue. A single EC2 instance reads the messages from the queue, processes them, and stores them in a DynamoDB table partitioned by unique order ID. Next month, traffic is expected to increase by a factor of 10 and a Solutions Architect is reviewing the architecture for possible scaling problems.

Which component is MOST likely to need re-architecting to be able to scale to accommodate the new traffic?

- A. Lambda function
- B. SQS queue
- C. EC2 instance
- D. DynamoDB table

# Additional resources

- [Set it and Forget it: Auto Scaling Target Tracking Policies](#)
- [Introduction to Amazon Elastic Load Balancer – Application](#)
- [Configuring Auto Scaling Group with ELB Elastic Load Balancer](#)
- [What Is an Application Load Balancer?](#)

# Thank you

© 2020 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. Corrections or feedback on the course, please email us at: [aws-course-feedback@amazon.com](mailto:aws-course-feedback@amazon.com). For all other questions, contact us at: <https://aws.amazon.com/contact-us/aws-training/>. All trademarks are the property of their owners.

