CLOUD COMPUTING

Session 3 :

High Availability and Scalability In The AWS Cloud

APROBAT

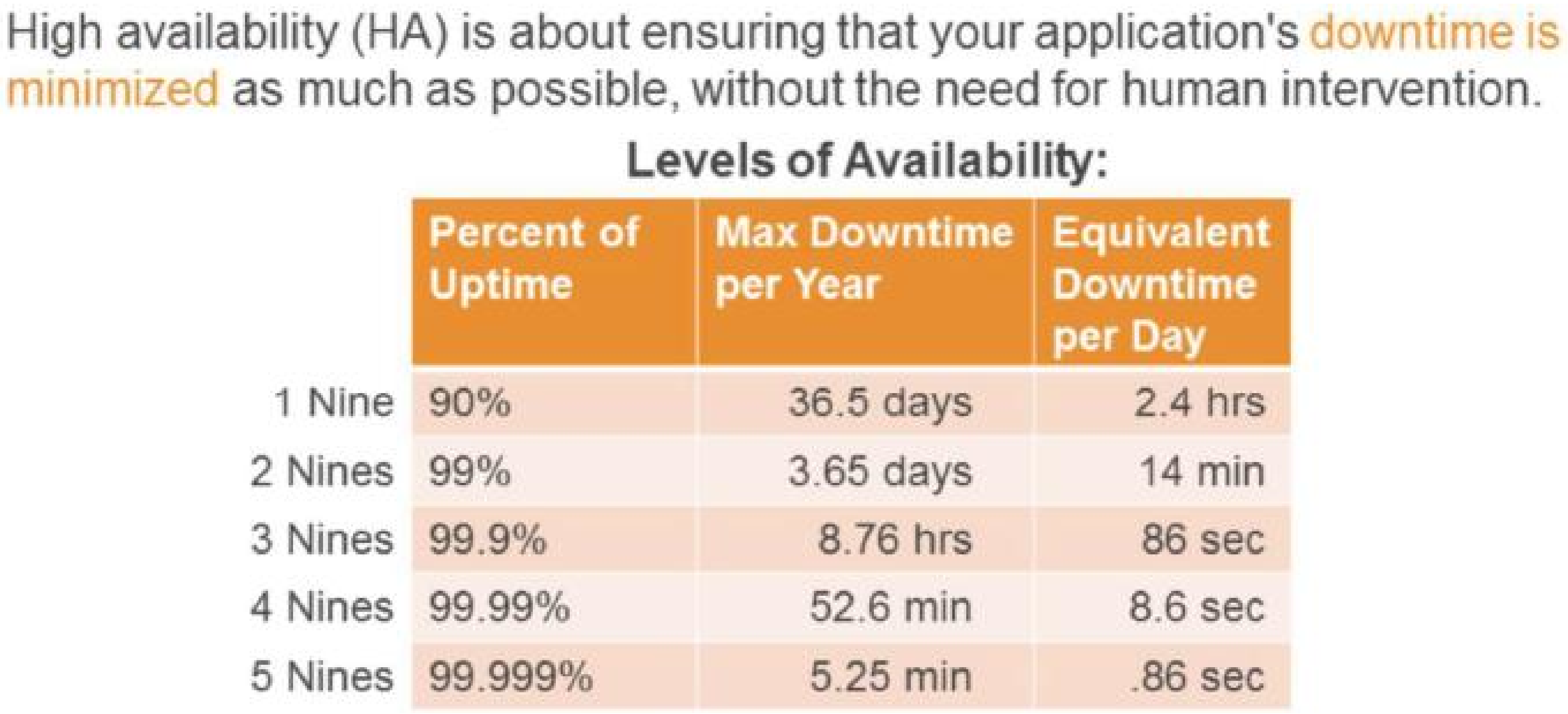
Conf. univ. dr. ing. IUSTIN PRIESCU - UTM
Dr. ing. Sebastian NICOLAESCU - Verizone Bussines-US

Design your application assuming that:

“*Everything fails, all the time*”

Werner Vogel, PhD – Amazon CTO

What is High Availability?



High availability (HA) is about ensuring that your application's downtime is minimized as much as possible, without the need for human intervention.

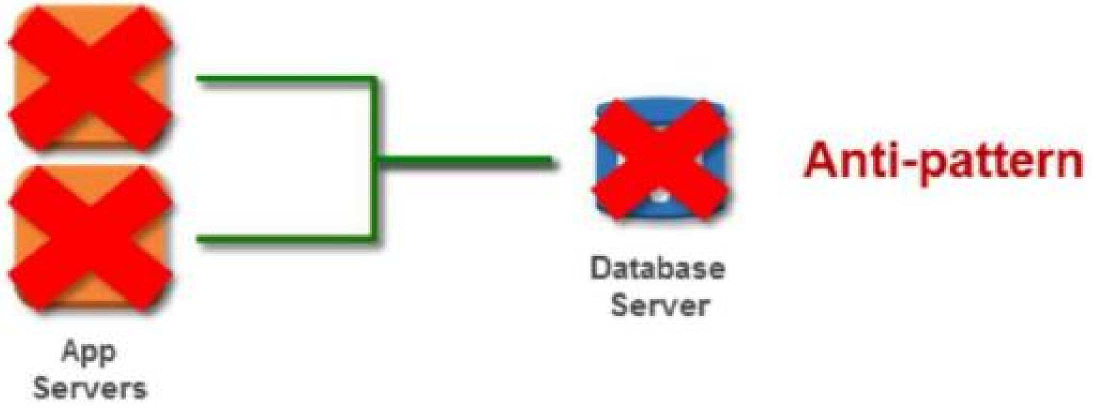
Levels of Availability:

	Percent of Uptime	Max Downtime per Year	Equivalent Downtime per Day
1 Nine	90%	36.5 days	2.4 hrs
2 Nines	99%	3.65 days	14 min
3 Nines	99.9%	8.76 hrs	86 sec
4 Nines	99.99%	52.6 min	8.6 sec
5 Nines	99.999%	5.25 min	.86 sec

Best Practice: Avoid Single Points Of Failure

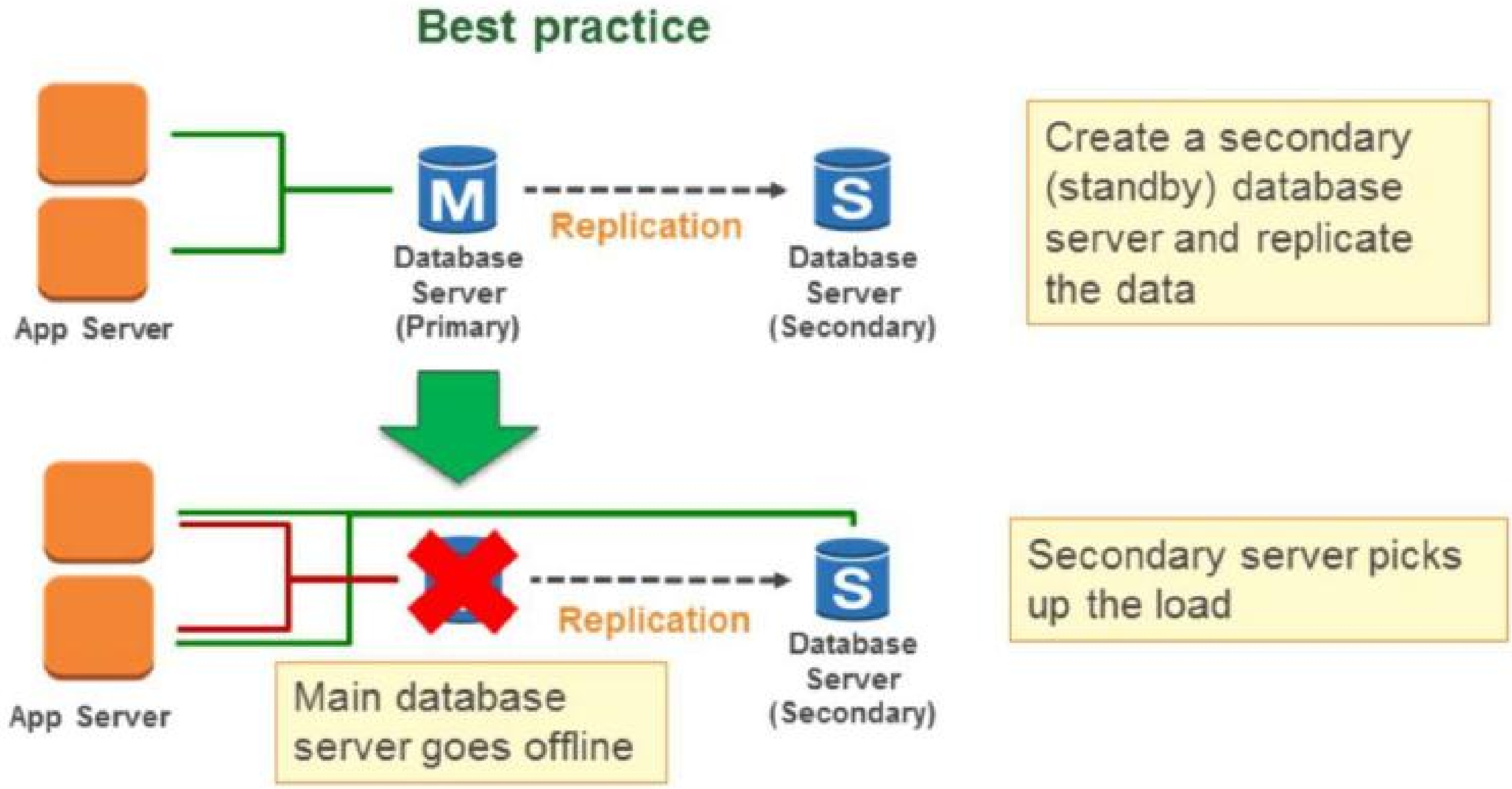
> *Assume everything fails, and design backwards.*

Implement redundancy where possible in order to prevent single failures from bringing down an entire system.



-It does not mean you have to duplicate the resources
- More important is recovery automation and the use of services that provide HA

Best Practice: Avoid Single Points Of Failure

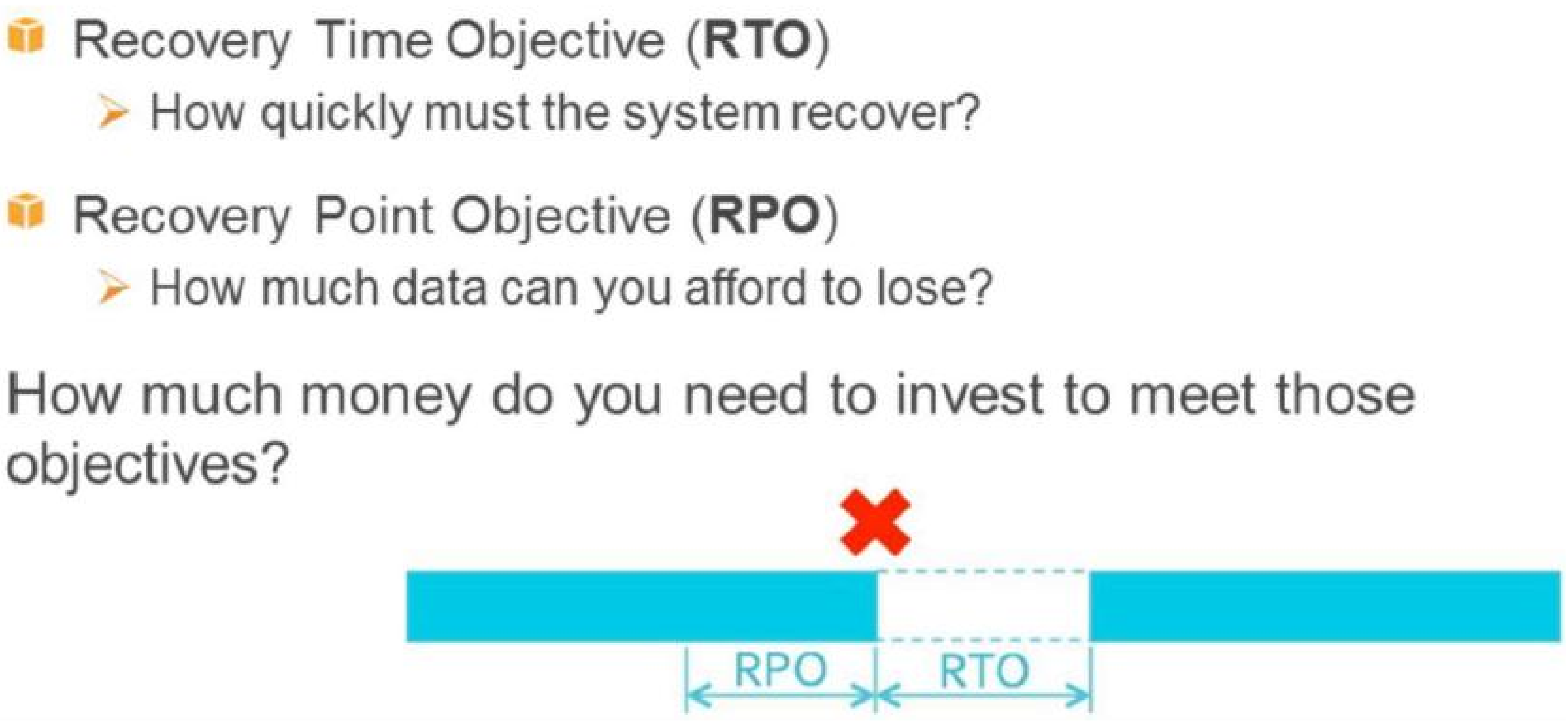

Best practice
App Server
M
Database Server (Primary)
Replication
S
Database Server (Secondary)
App Server
Replication
S
Database Server (Secondary)

Create a secondary (standby) database server and replicate the data

Main database server goes offline

Secondary server picks up the load

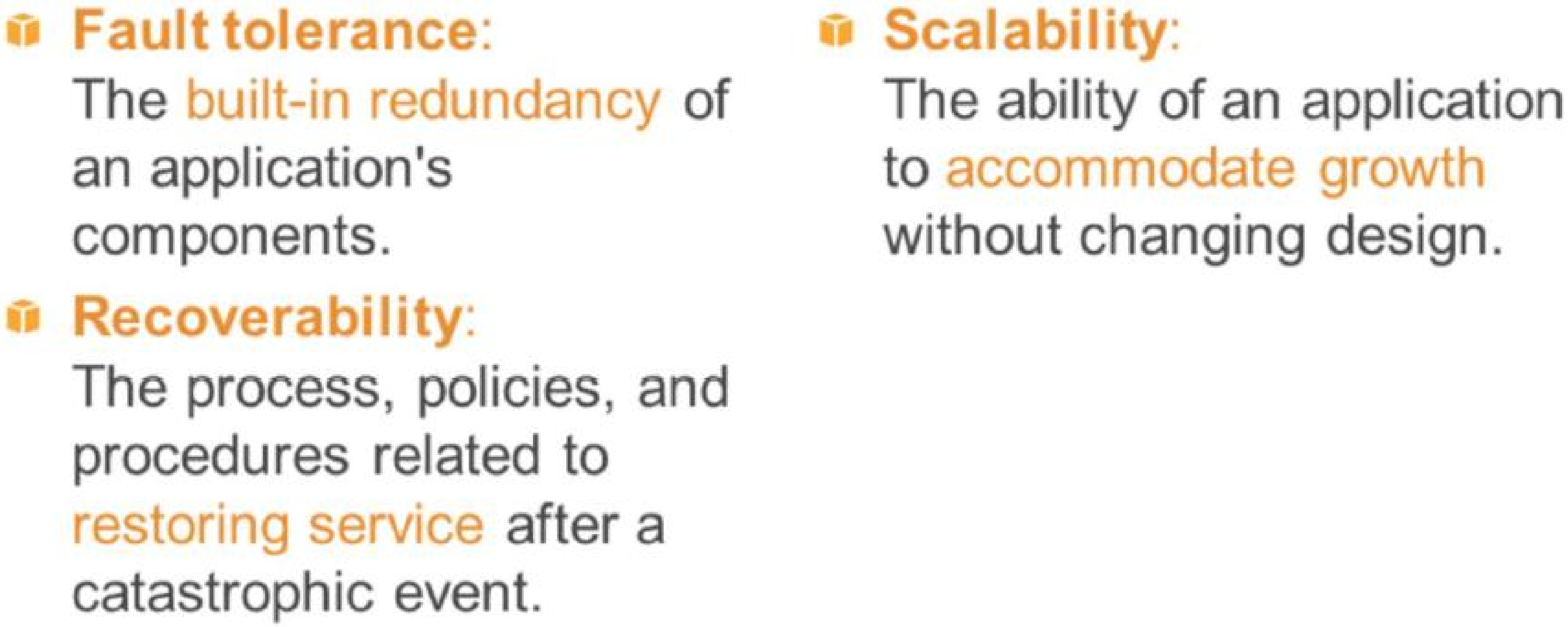
High Availability: Driven By Your Requirements



- Recovery Time Objective (**RTO**)
  - How quickly must the system recover?
- Recovery Point Objective (**RPO**)
  - How much data can you afford to lose?

How much money do you need to invest to meet those objectives?

High Availability Factors



- **Fault tolerance**:
  The built-in redundancy of an application's components.
- **Recoverability**:
  The process, policies, and procedures related to restoring service after a catastrophic event.
- **Scalability**:
  The ability of an application to accommodate growth without changing design.

High Availability and AWS Regions

Multi-region deployment increases:

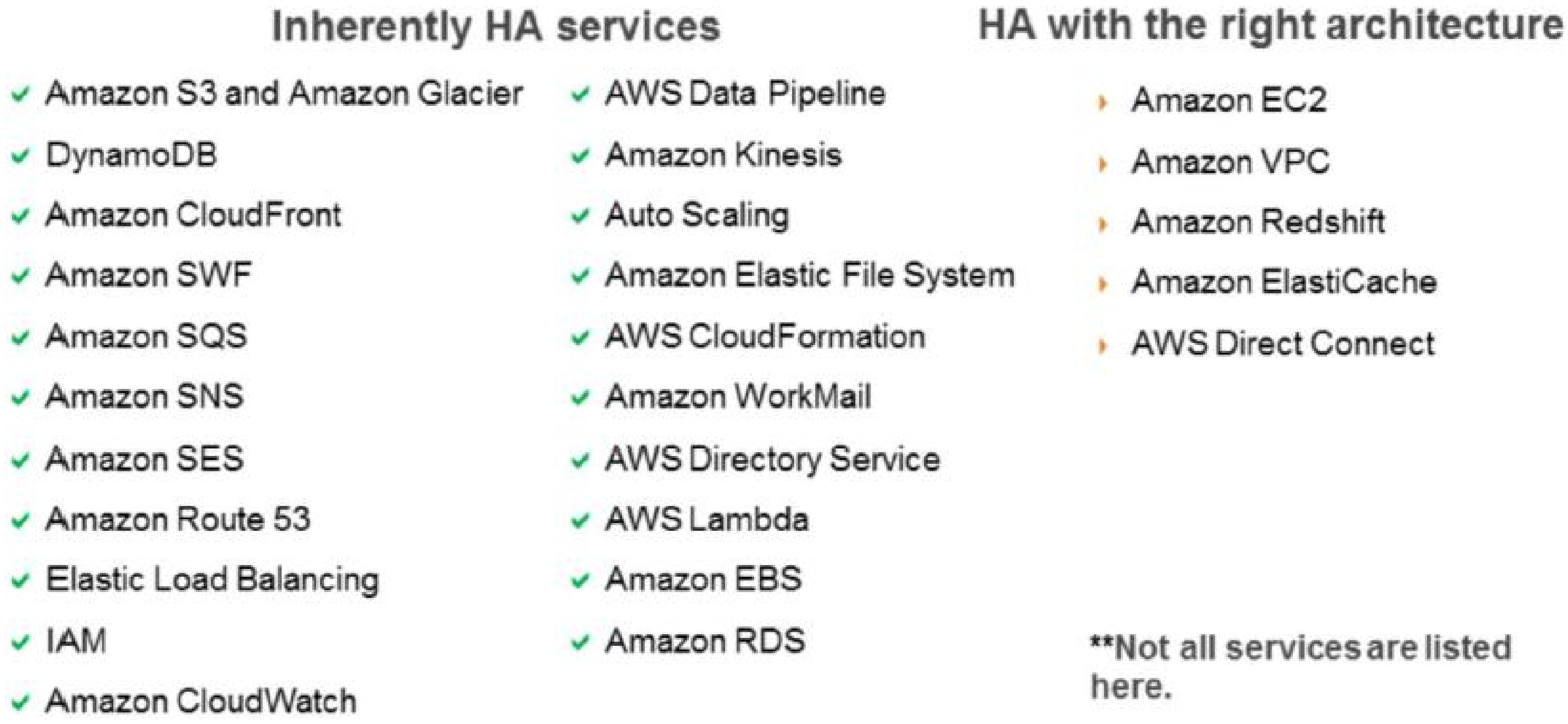
- Availability
- Cost
- Complexity

QUEBEC (Coming soon)
OHIO (Coming soon)
OREGON
N. CALIFORNIA
AWS GOVCLOUD
N. VIRGINIA
IRELAND
UK (Coming soon)
FRANKFURT
BEIJING
SEOUL
NINGXIA (Coming soon)
TOKYO
INDIA (Coming soon)
SINGAPORE
SÃO PAULO
SYDNEY
Region & Number of Availability Zones
New Region Coming Soon

Default to a single region, unless multi-region deployment is necessary.

If you use a single region, multi-AZ is the bare minimum HA solution.

AWS Services and High Availability



Inherently HA services

- ✓ Amazon S3 and Amazon Glacier
- ✓ DynamoDB
- ✓ Amazon CloudFront
- ✓ Amazon SWF
- ✓ Amazon SQS
- ✓ Amazon SNS
- ✓ Amazon SES
- ✓ Amazon Route 53
- ✓ Elastic Load Balancing
- ✓ IAM
- ✓ Amazon CloudWatch
- ✓ AWS Data Pipeline
- ✓ Amazon Kinesis
- ✓ Auto Scaling
- ✓ Amazon Elastic File System
- ✓ AWS CloudFormation
- ✓ Amazon WorkMail
- ✓ AWS Directory Service
- ✓ AWS Lambda
- ✓ Amazon EBS
- ✓ Amazon RDS

HA with the right architecture

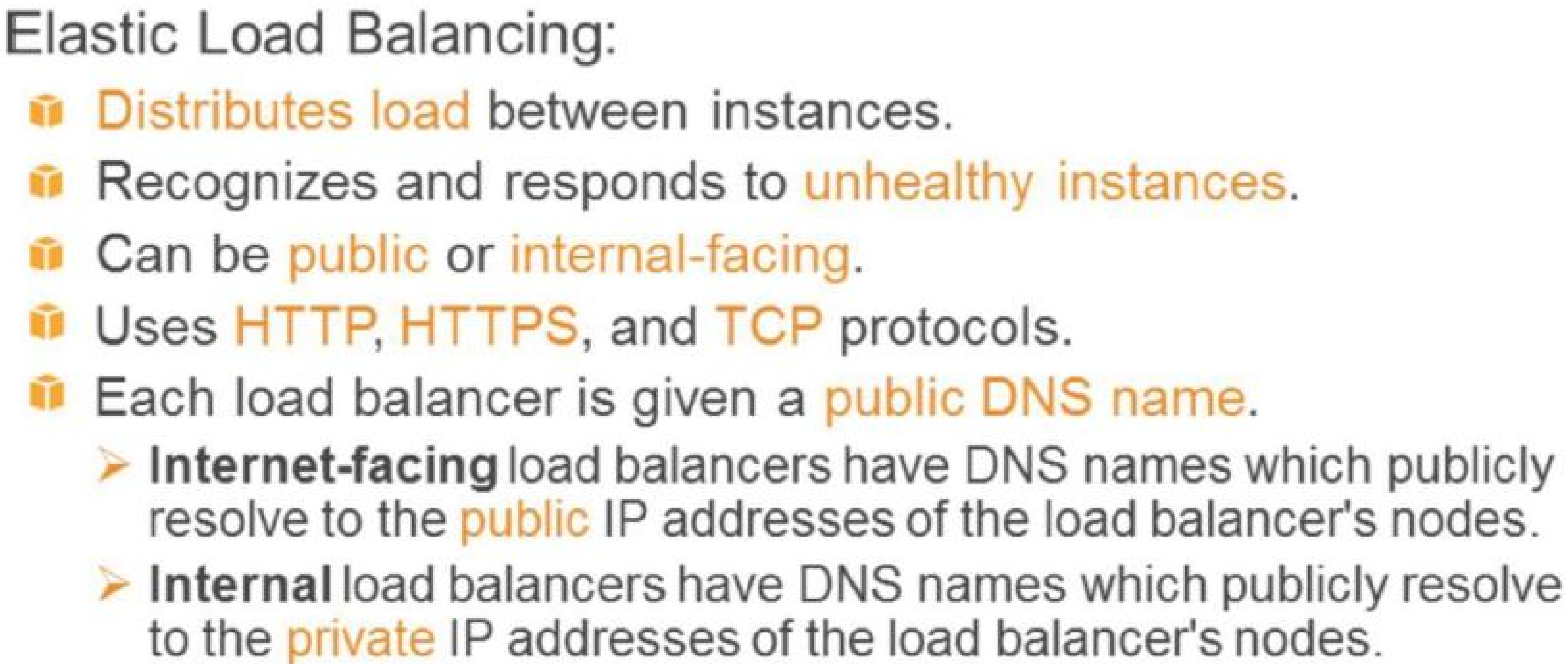
- ▸ Amazon EC2
- ▸ Amazon VPC
- ▸ Amazon Redshift
- ▸ Amazon ElastiCache
- ▸ AWS Direct Connect

****Not all services are listed here.**

ELB – Elastic Load Balancer

A managed load balancing service that distributes incoming application traffic across multiple Amazon EC2 instances.

AWS Load Balancing Solutions - ELB



Elastic Load Balancing:

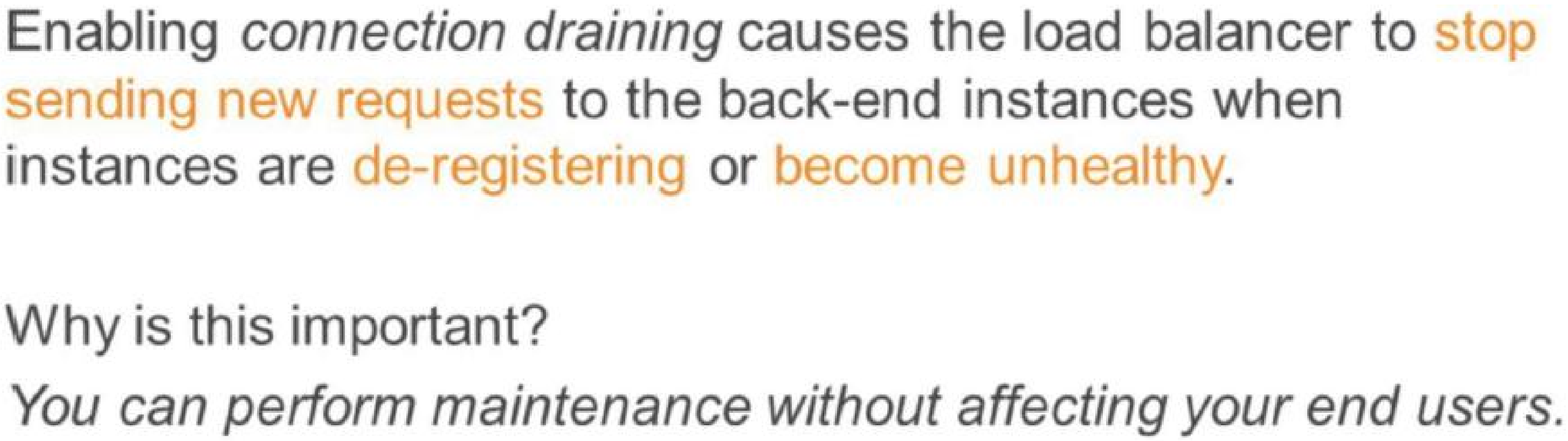
- Distributes load between instances.
- Recognizes and responds to unhealthy instances.
- Can be public or internal-facing.
- Uses HTTP, HTTPS, and TCP protocols.
- Each load balancer is given a public DNS name.
  - **Internet-facing** load balancers have DNS names which publicly resolve to the public IP addresses of the load balancer's nodes.
  - **Internal** load balancers have DNS names which publicly resolve to the private IP addresses of the load balancer's nodes.

Reasons for Using ELB

Elastic Load Balancing provides the following features:

- Health checks
- Cross-zone load balancing
- Proxy Protocol
- Sticky sessions
- Connection draining

ELB – Connection Draining



Enabling *connection draining* causes the load balancer to stop sending new requests to the back-end instances when instances are de-registering or become unhealthy.

Why is this important?

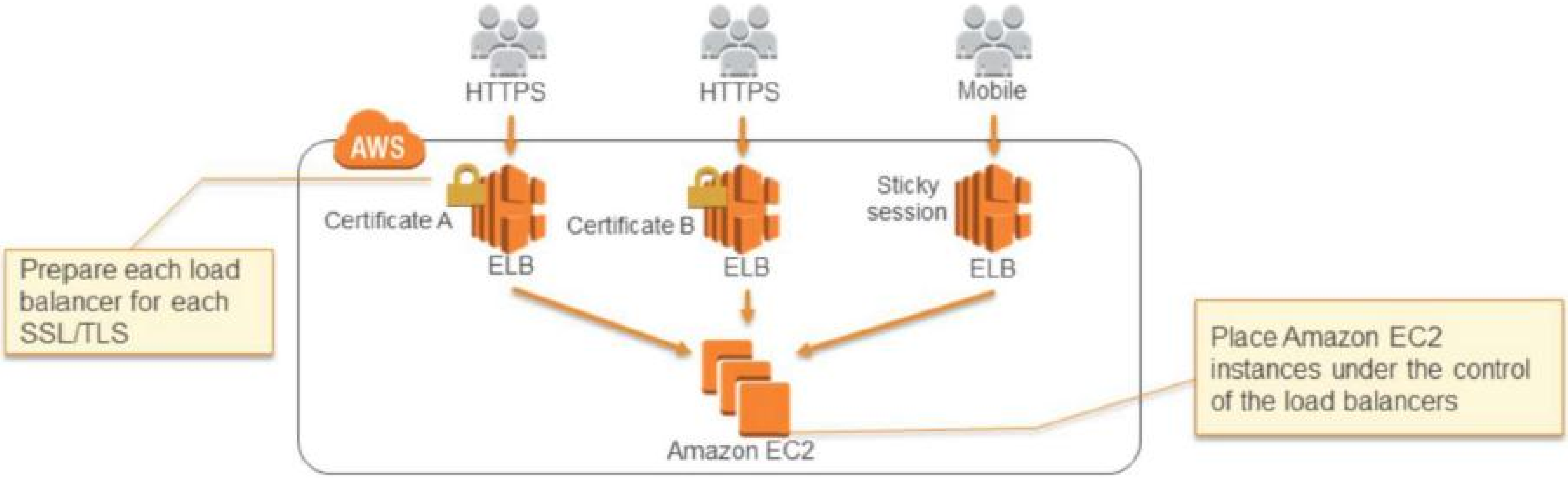
You can perform maintenance without affecting your end users.

Cloud Design Pattern: Multi Load Balancer Pattern

Problem to be solved:
When a web application is multi-device compatible, there will be access from PCs, mobile phones, and smart phones. At this time, if it is necessary to perform a setup such as for SSL/TLS or to assign sessions for individual access devices, and if the setup is performed by the EC2 instances themselves, any change to the settings would become extremely laborious as the number of servers increases.

Cloud Pattern:

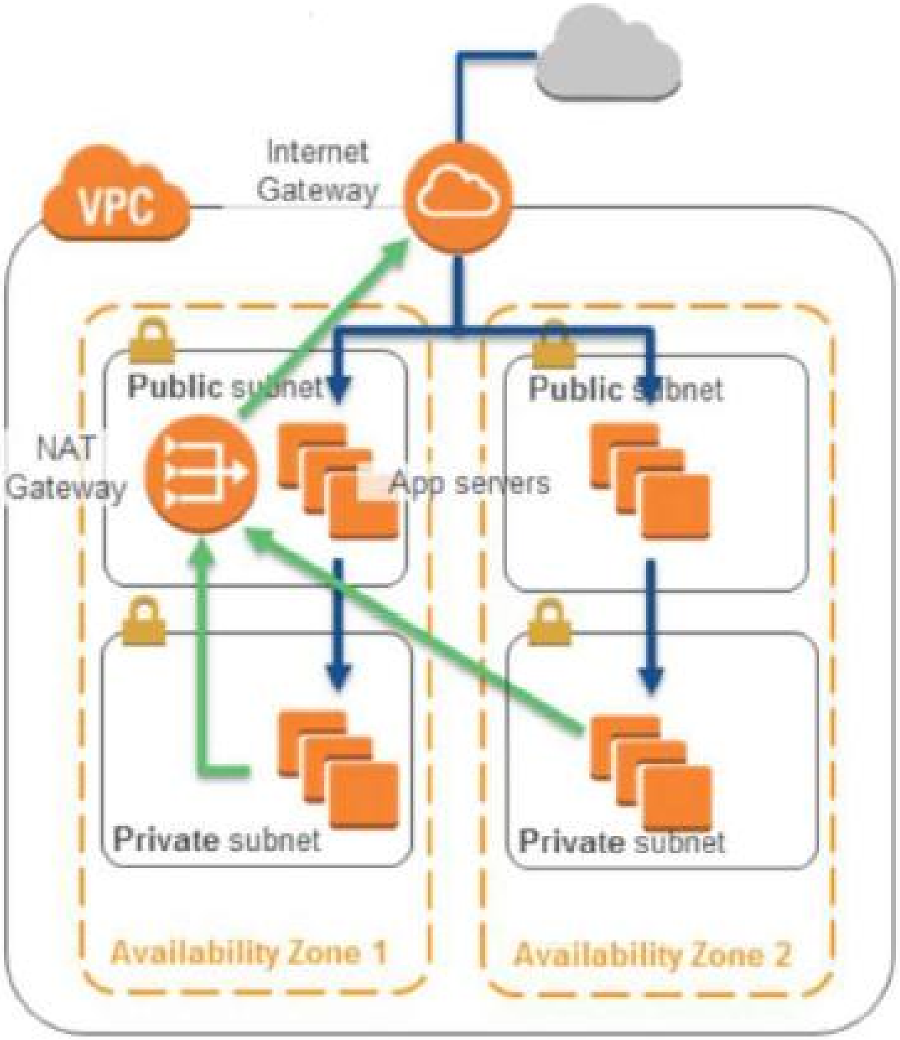
Assign multiple load balancers with different settings for the types of devices that are accessing the web application.



Prepare each load balancer for each SSL/TLS

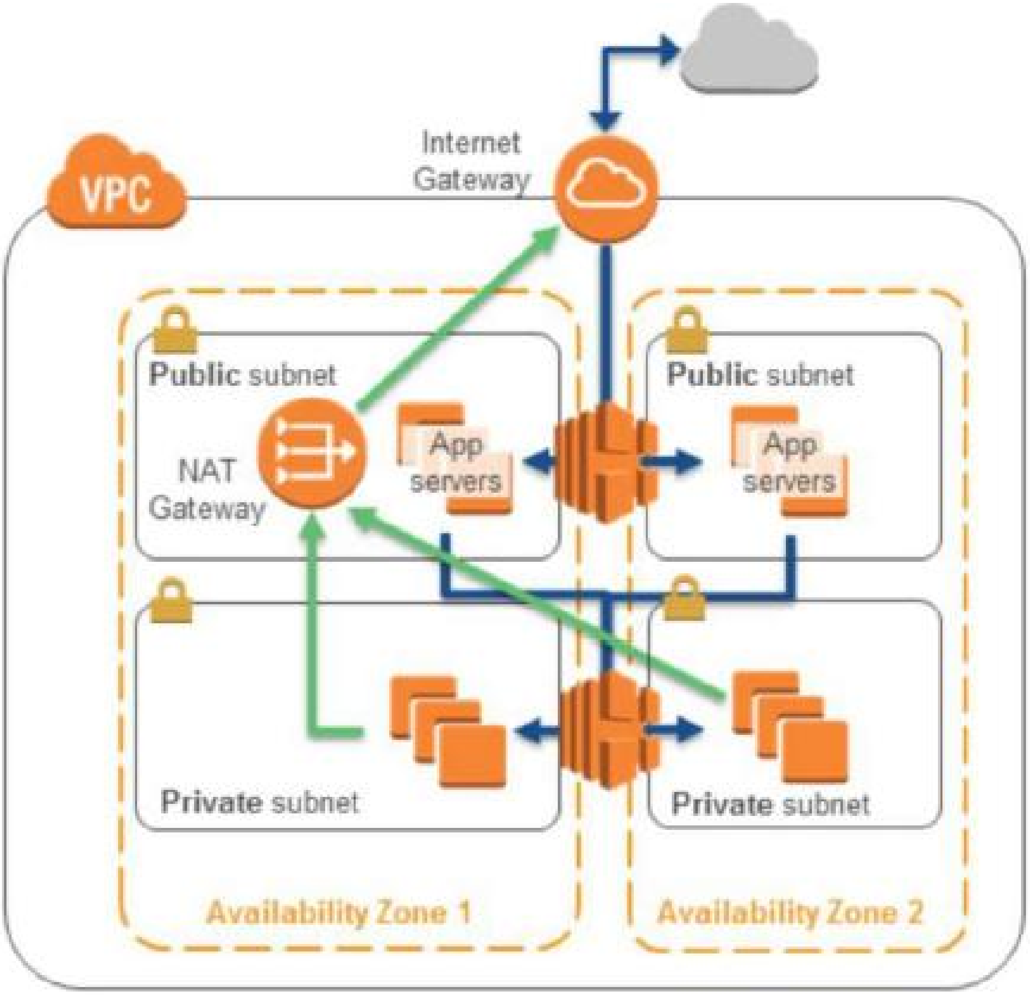
Place Amazon EC2 instances under the control of the load balancers

Exercise: How Can Availability of This Environment Be Improved?

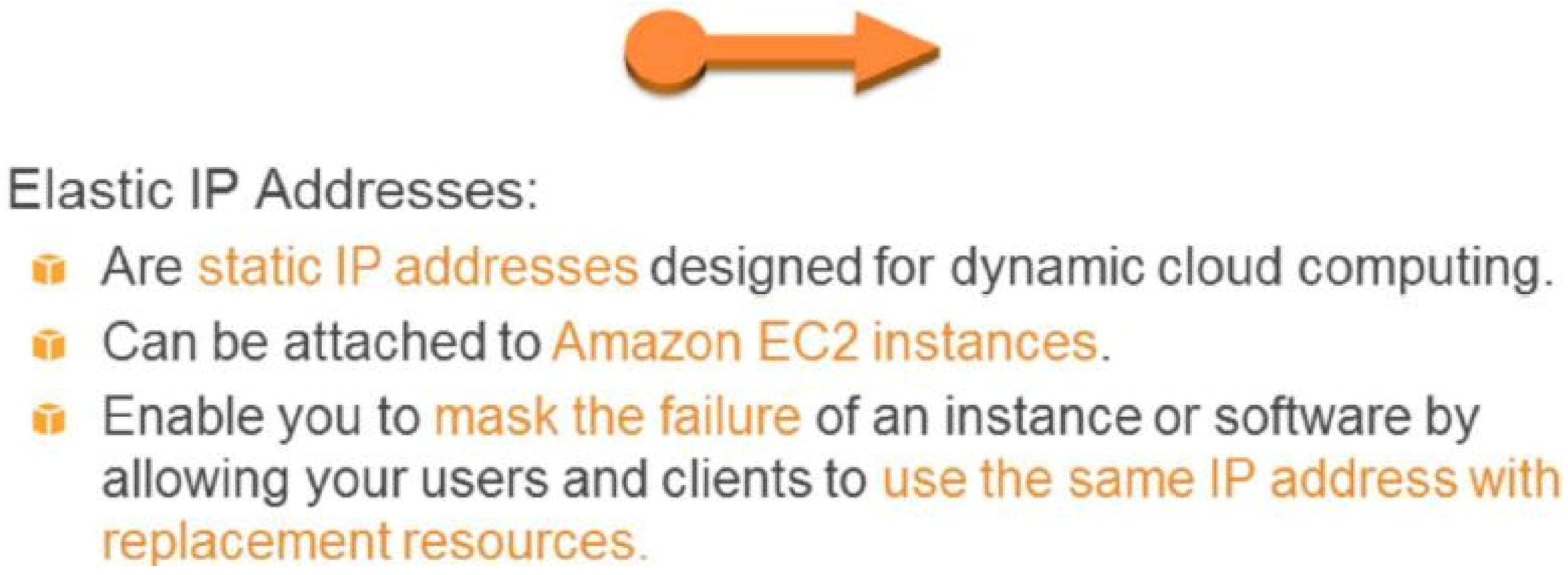


High Availability With Load Balancing

- **Internet-facing** load balancer
  - Deployed in **public** subnet
  - Balancing traffic to app servers in two Availability Zones
- **Internal** load balancer
  - Intranet-facing or internal



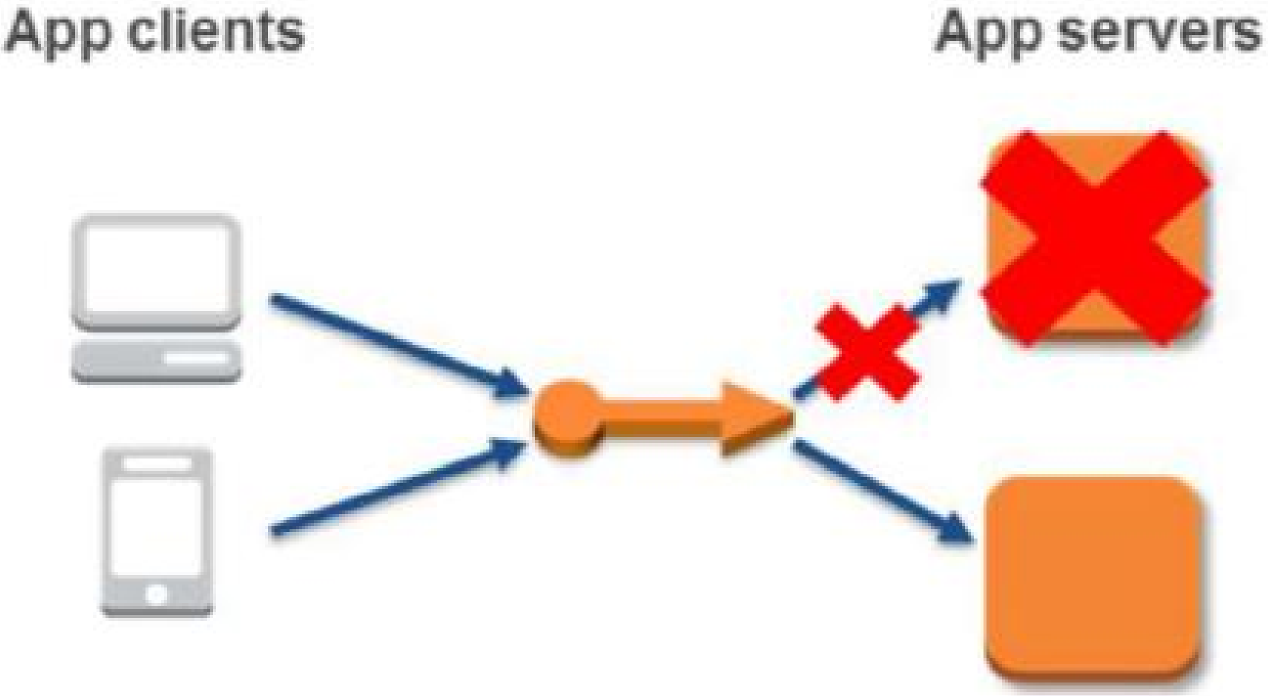
Elastic IP: For Greater Fault Tolerance



Elastic IP Addresses:

- Are static IP addresses designed for dynamic cloud computing.
- Can be attached to Amazon EC2 instances.
- Enable you to mask the failure of an instance or software by allowing your users and clients to use the same IP address with replacement resources.

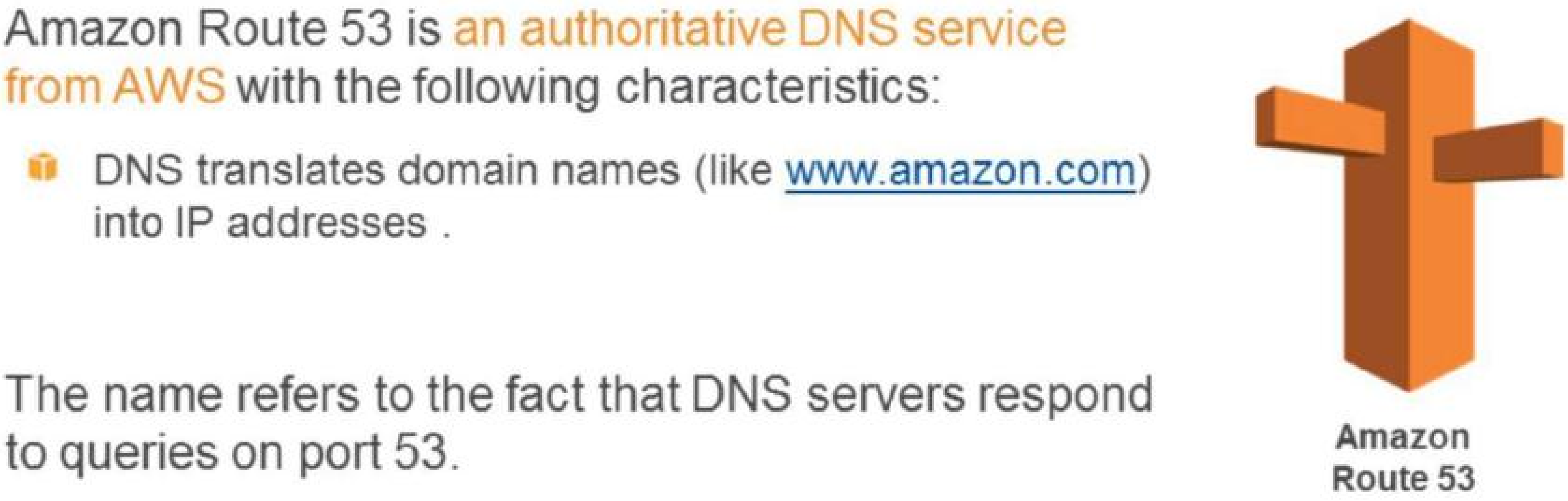
Using Elastic IP Addresses to Enable High Availability



If one instance goes down, clients can use the same IP address to reach the replacement instance.

High Availability Across Regions via Route 53

Route 53 –Highly Available Domain Name Service (DNS)



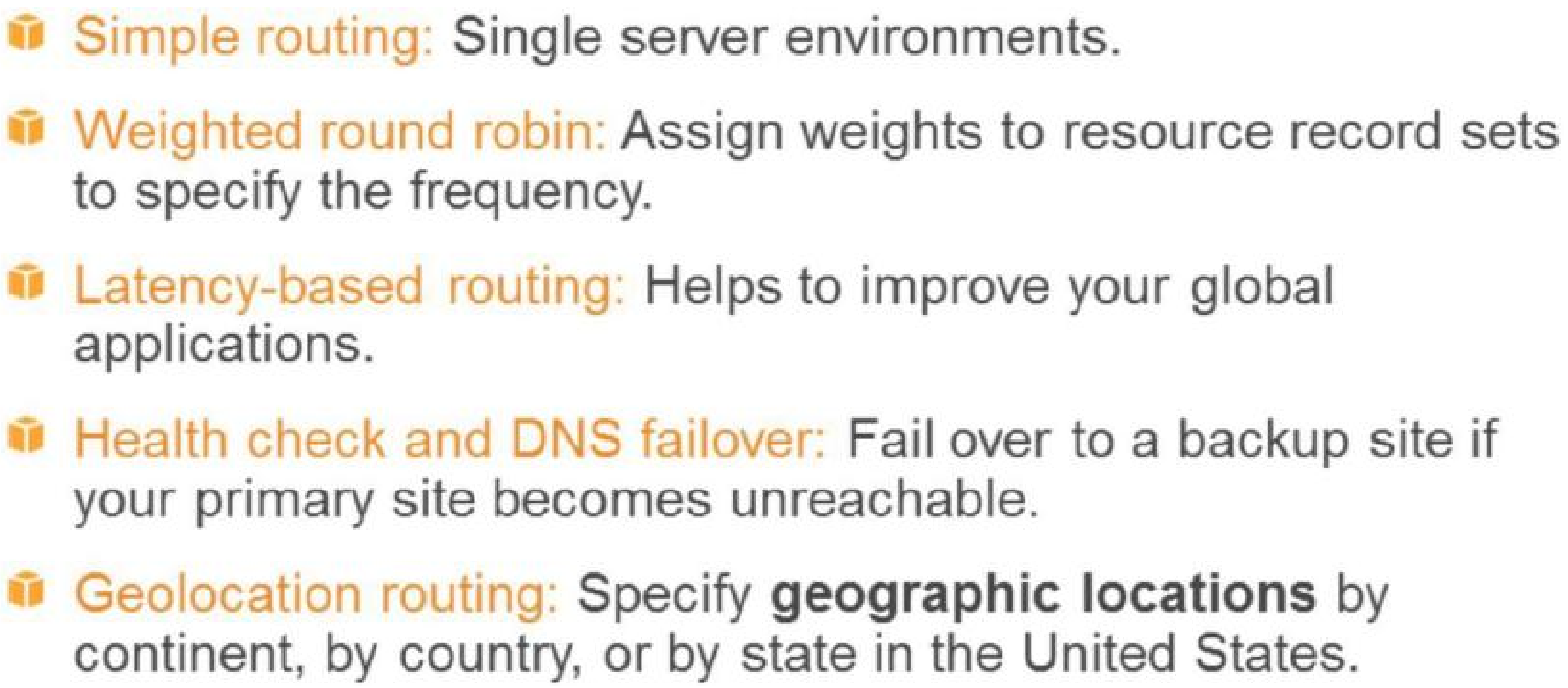
Amazon Route 53 is an authoritative DNS service from AWS with the following characteristics:

- DNS translates domain names (like www.amazon.com) into IP addresses .

The name refers to the fact that DNS servers respond to queries on port 53.

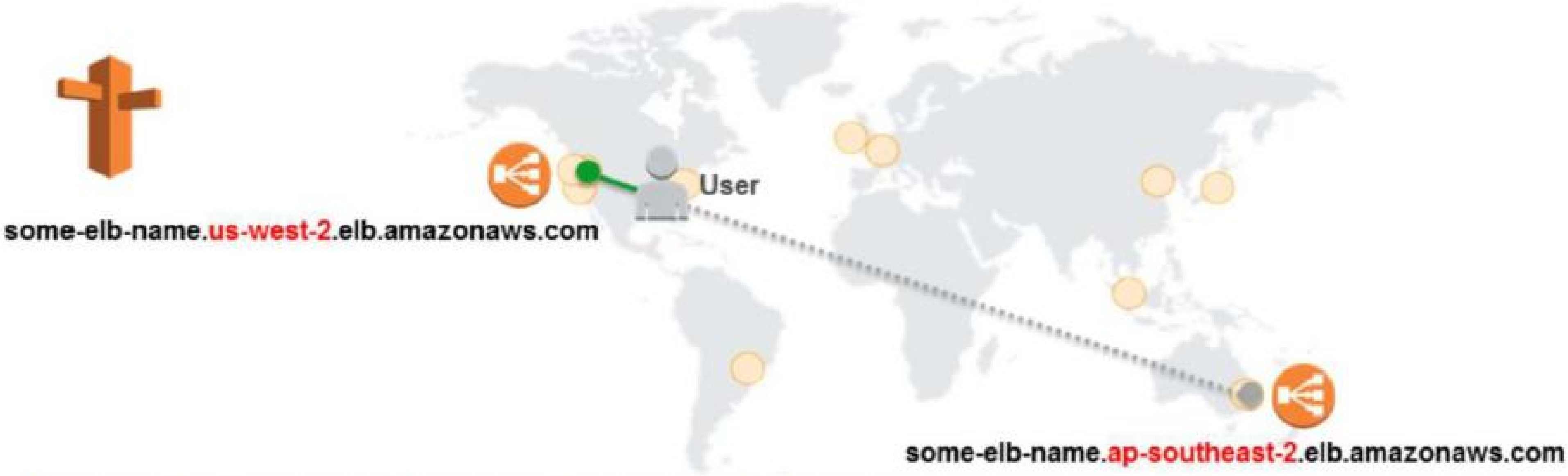
Amazon
Route 53

AWS Route 53 – Routing Options



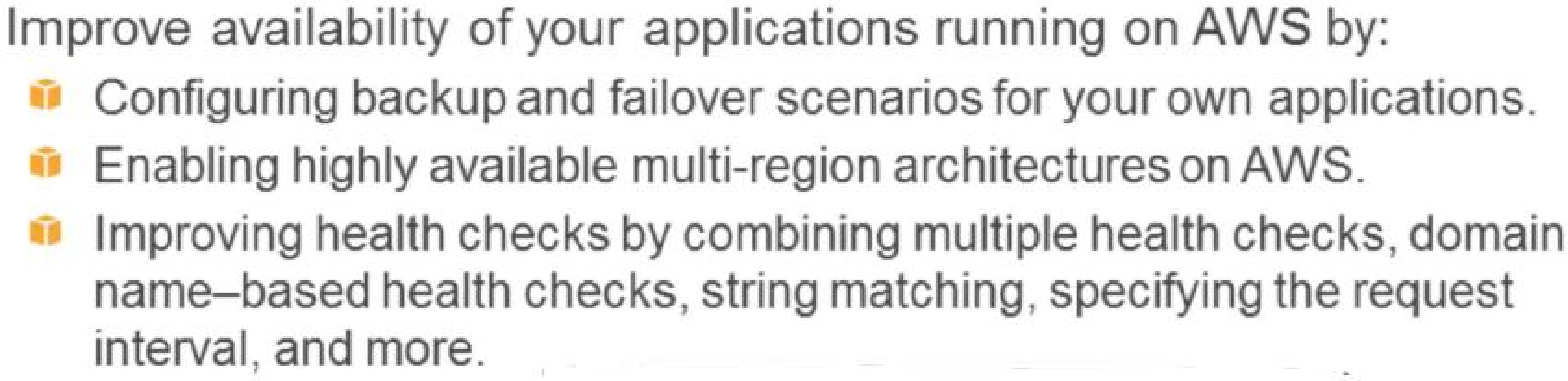
- Simple routing: Single server environments.
- Weighted round robin: Assign weights to resource record sets to specify the frequency.
- Latency-based routing: Helps to improve your global applications.
- Health check and DNS failover: Fail over to a backup site if your primary site becomes unreachable.
- Geolocation routing: Specify **geographic locations** by continent, by country, or by state in the United States.

Use Case: Multi-Region Deployment


User
some-elb-name.us-west-2.elb.amazonaws.com
some-elb-name.ap-southeast-2.elb.amazonaws.com

Name	Type	Value
amgogreen.com	ALIAS	some-elb-name.us-west-2.elb.amazonaws.com
amgogreen.com	ALIAS	some-elb-name.ap-southeast-2.elb.amazonaws.com

AWS High Availability



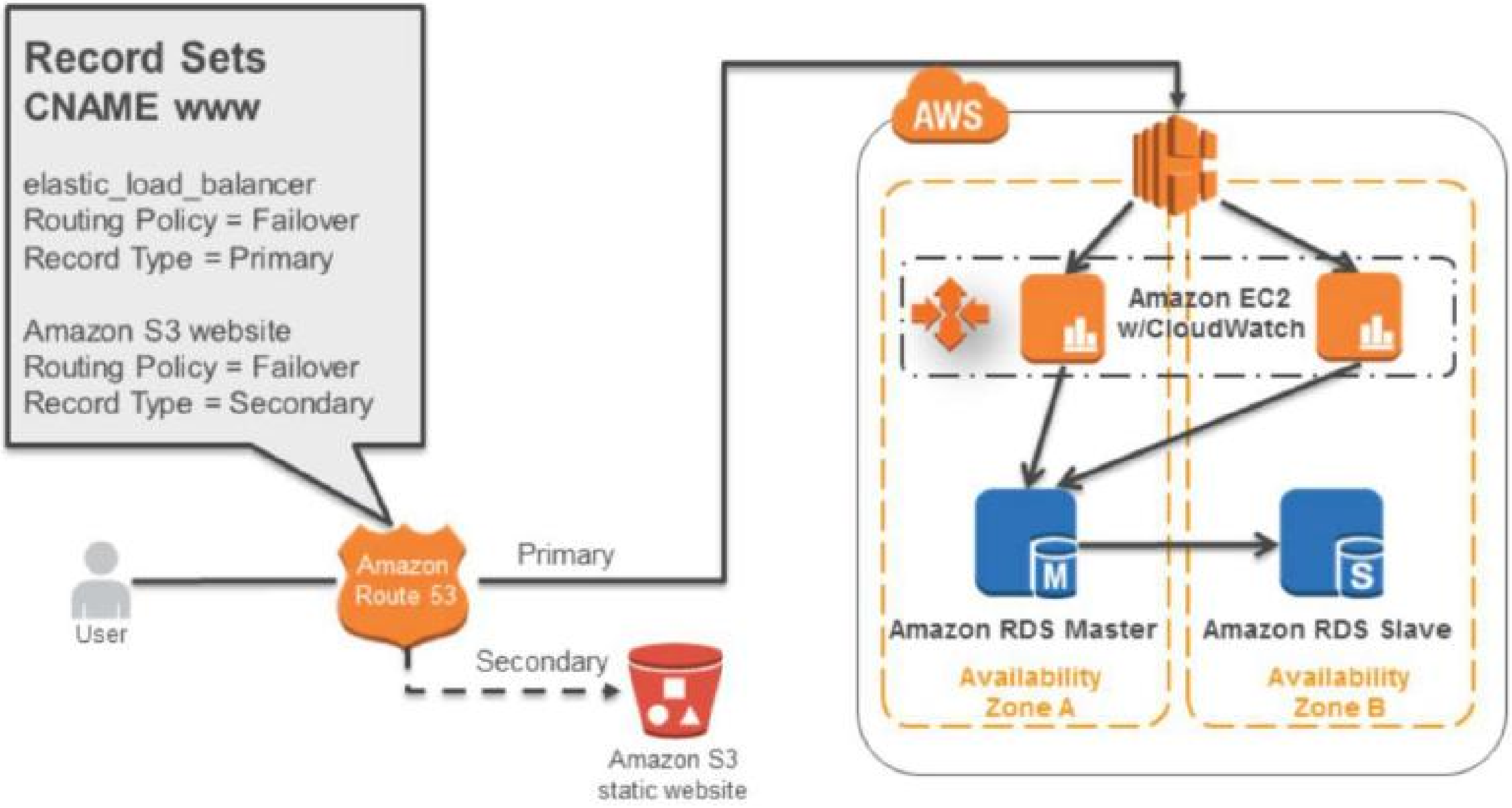
Improve availability of your applications running on AWS by:

- Configuring backup and failover scenarios for your own applications.
- Enabling highly available multi-region architectures on AWS.
- Improving health checks by combining multiple health checks, domain name–based health checks, string matching, specifying the request interval, and more.

- *Solution is driven by your SLA*
- *Robust HA solutions are mainly determined by the* ***perfection*** *of the* ***recovery chain:***
  - *failure detection*
  - *flawless failover code*
  - *working backups*

http://ucare.cs.uchicago.edu/pdf/socc16-cos.pdf

Typical Multi Region Architecture



Record Sets
CNAME www

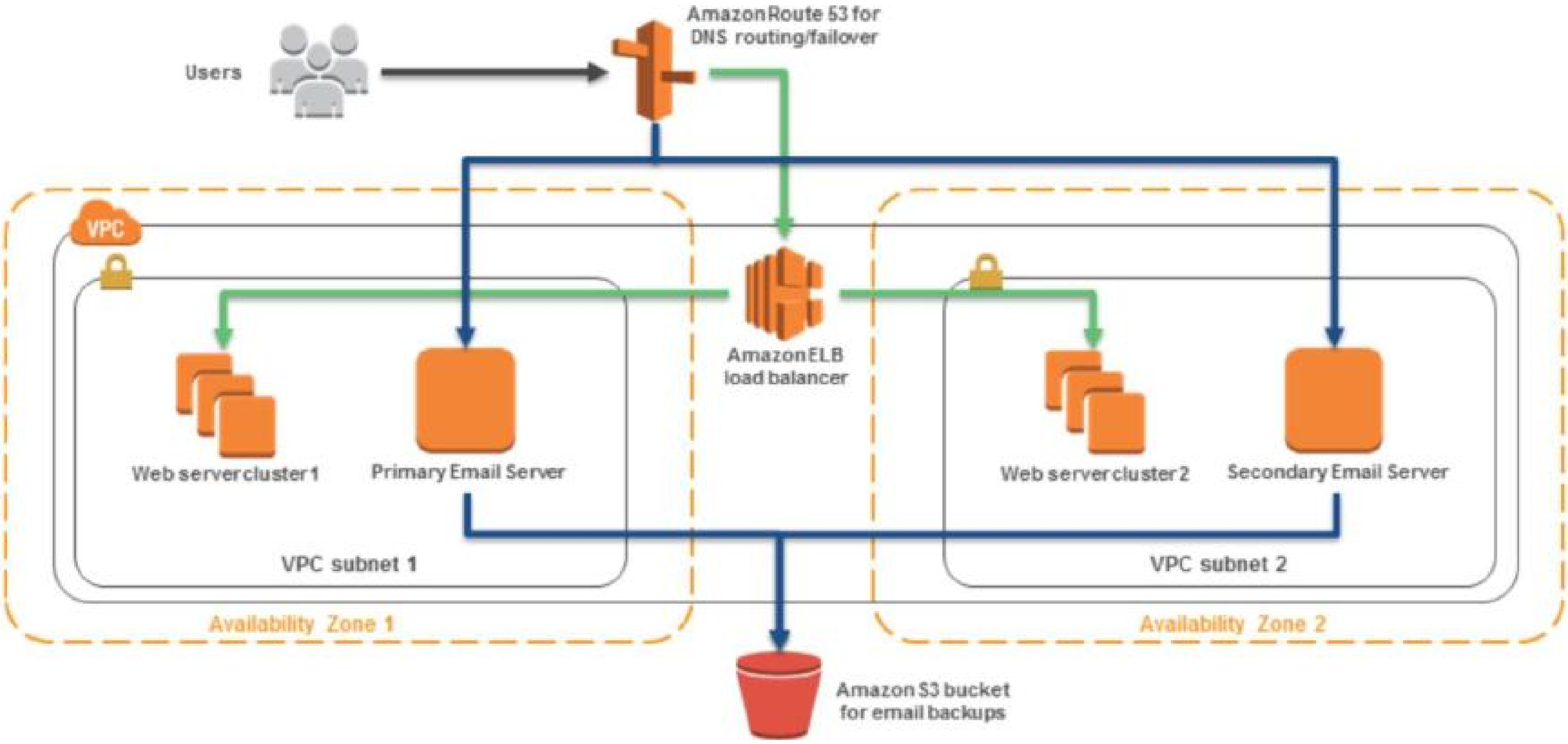
elastic_load_balancer
Routing Policy = Failover
Record Type = Primary

Amazon S3 website
Routing Policy = Failover
Record Type = Secondary

Alta

http://www.altavw.com.br/

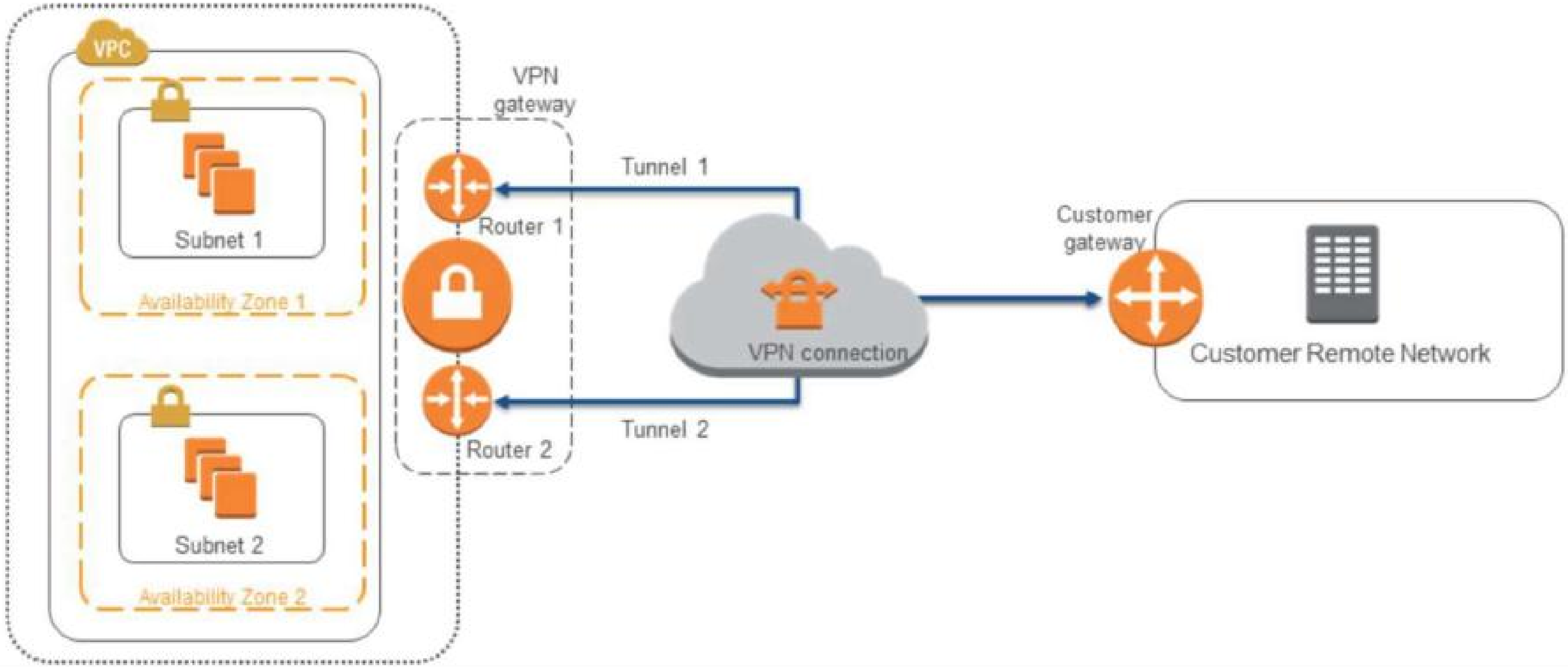
Alta Veiculos Use Case



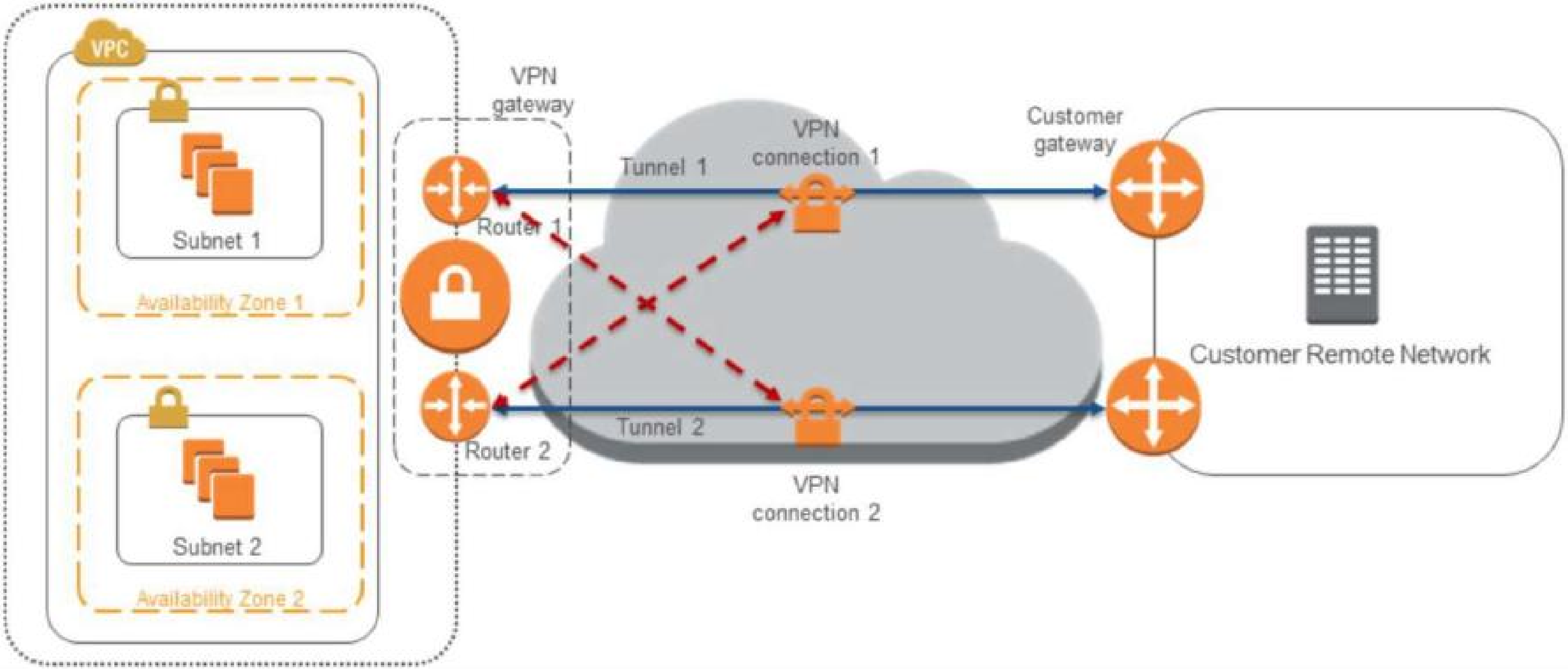
For more on this case study, see: http://aws.typepad.com/brasil/case-studies/

Availability Considerations About Connections to Components Outside of AWS

VPN Connection : Is This Highly Available?



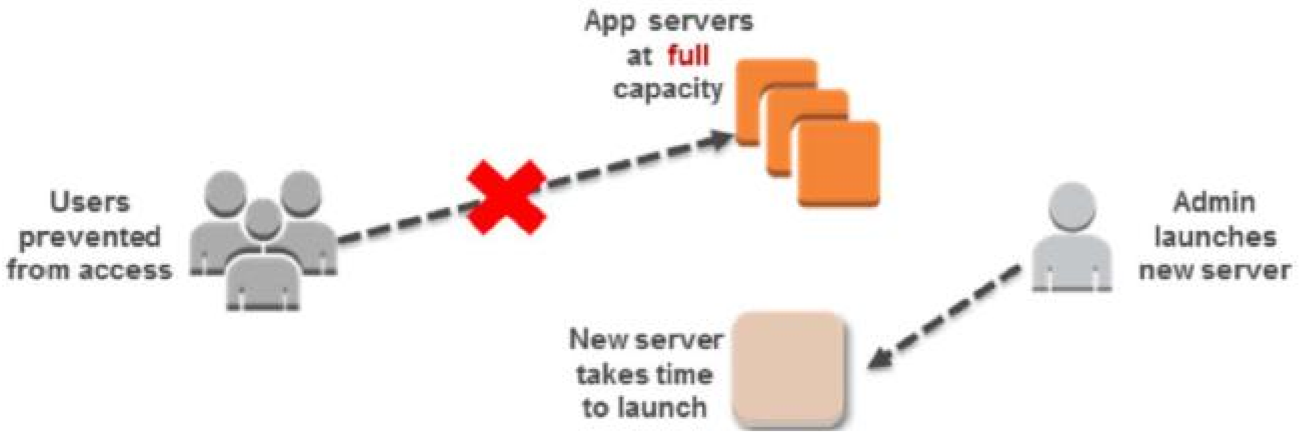
Adding Connection Redundancy With AWS HW VPNs



Best Practice : Enable Scalability

Ensure that your architecture can handle changes in demand.

A key advantage of a cloud-based infrastructure is how quickly you can respond to changes in resource needs.

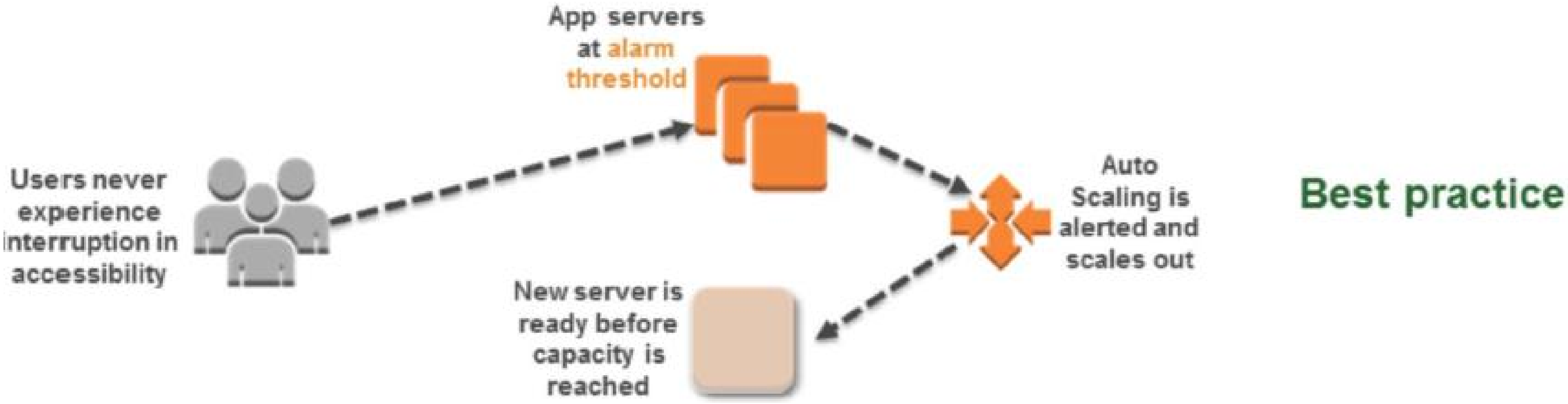

App servers at full capacity
Users prevented from access
Admin launches new server
New server takes time to launch

Anti-pattern

Best Practice : Enable Scalability

Ensure that your architecture can handle changes in demand.

A key advantage of a cloud-based infrastructure is how quickly you can respond to changes in resource needs.



App servers at alarm threshold

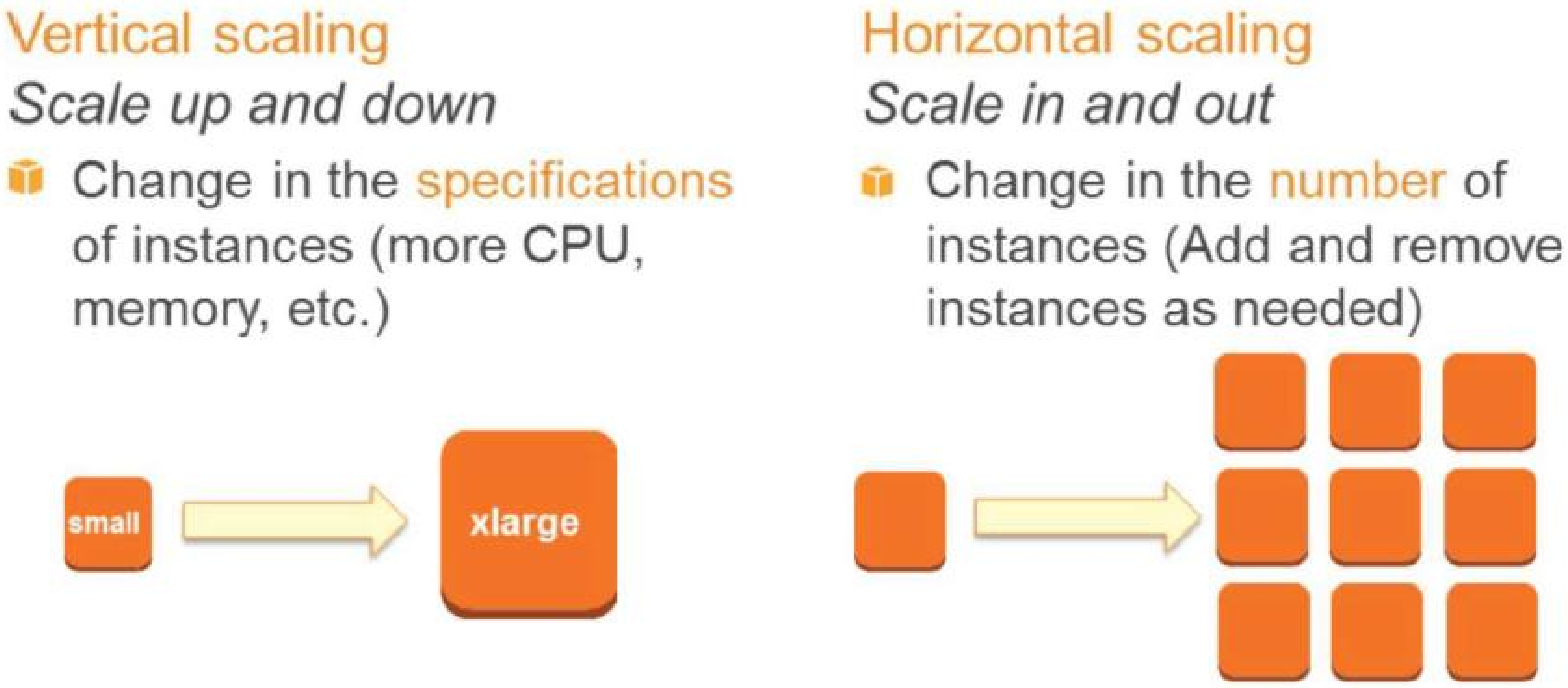
Users never experience interruption in accessibility

Auto Scaling is alerted and scales out

New server is ready before capacity is reached

Best practice

Vertical vs Horizontal Scaling



Vertical scaling

Scale up and down

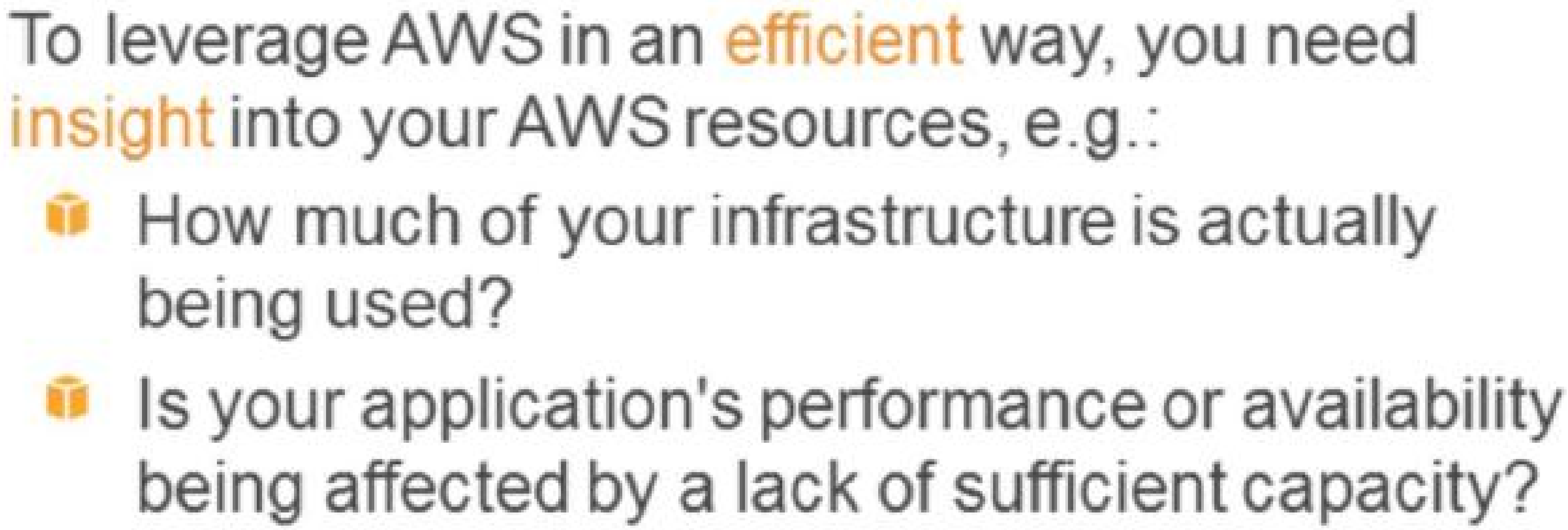
- Change in the specifications of instances (more CPU, memory, etc.)

Horizontal scaling

Scale in and out

- Change in the number of instances (Add and remove instances as needed)

What Information Is Needed For Scaling

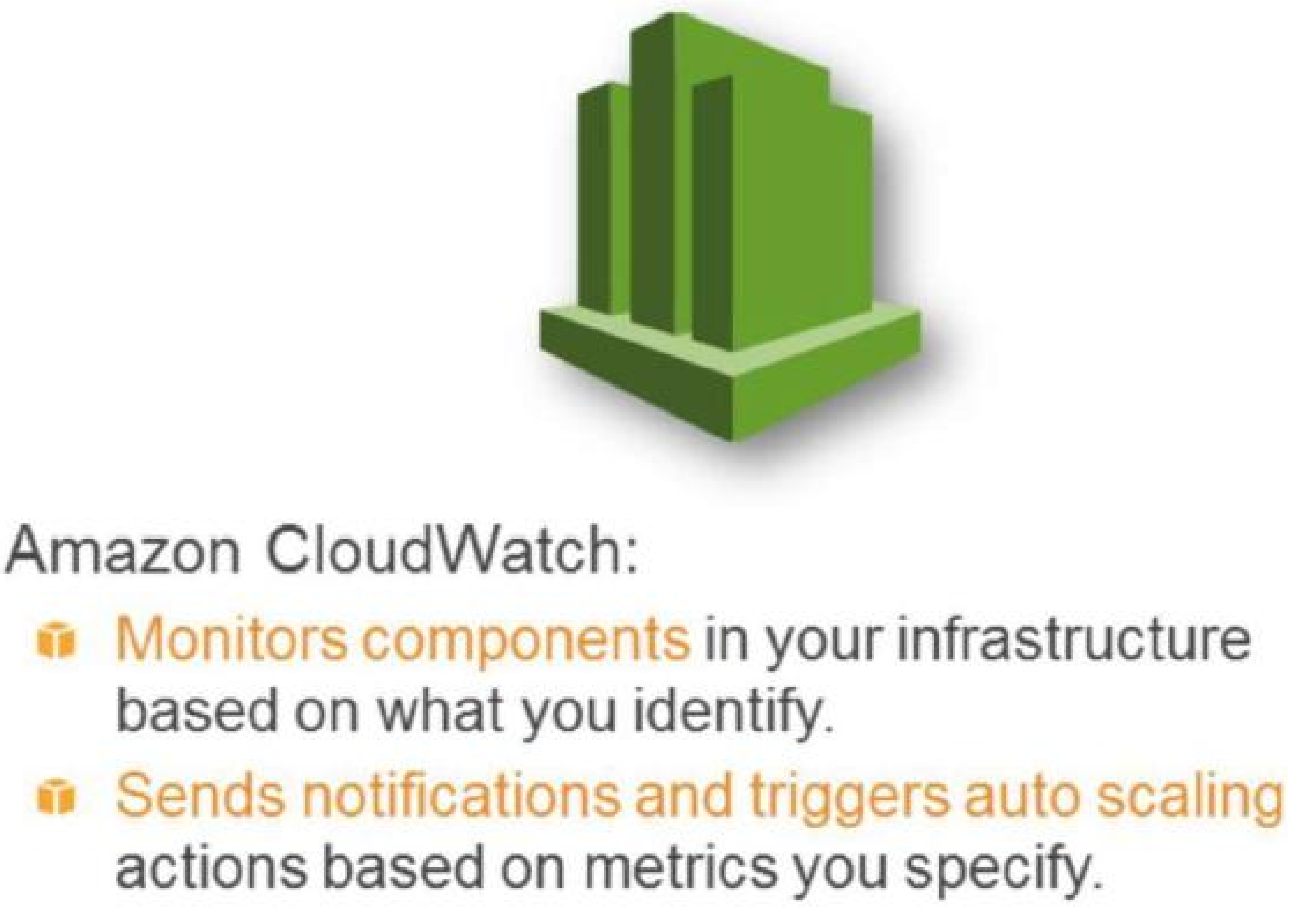


To leverage AWS in an efficient way, you need insight into your AWS resources, e.g.:

- How much of your infrastructure is actually being used?
- Is your application's performance or availability being affected by a lack of sufficient capacity?

How do we collect this information ??

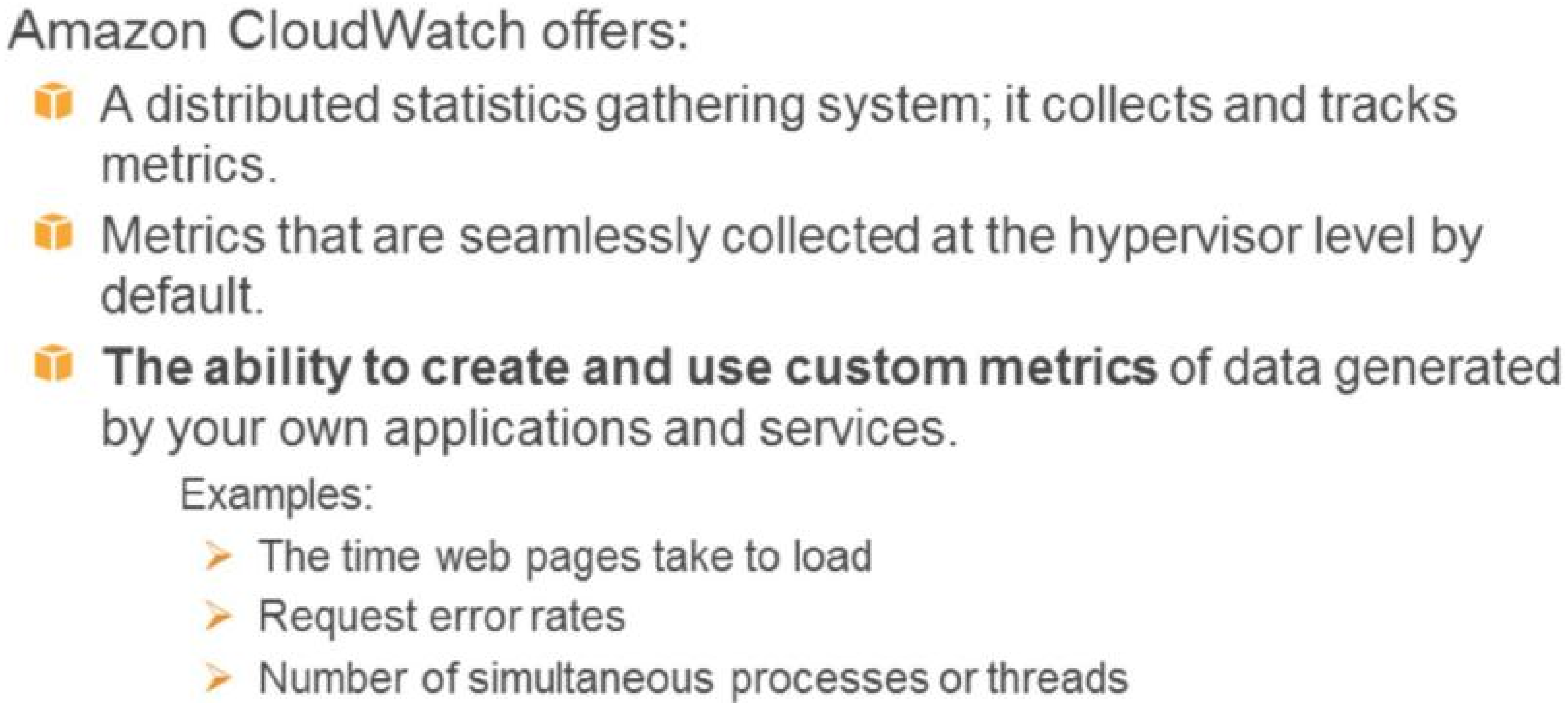
Collecting Metrics: AWS Cloud Watch



Amazon CloudWatch:

- Monitors components in your infrastructure based on what you identify.
- Sends notifications and triggers auto scaling actions based on metrics you specify.

Monitor AWS Resources With Cloud Watch



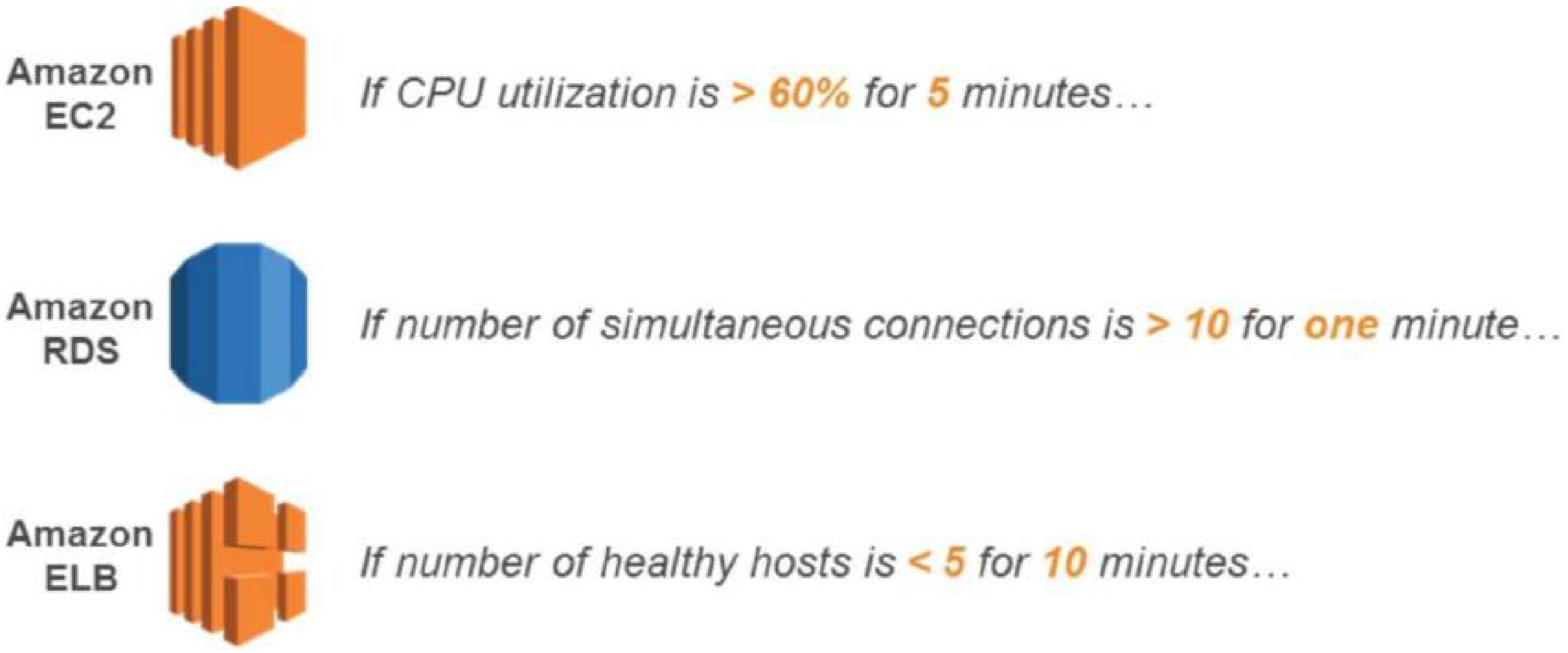
Amazon CloudWatch offers:

- A distributed statistics gathering system; it collects and tracks metrics.
- Metrics that are seamlessly collected at the hypervisor level by default.
- **The ability to create and use custom metrics** of data generated by your own applications and services.

  Examples:

  - The time web pages take to load
  - Request error rates
  - Number of simultaneous processes or threads

Cloud Watch Alarm Examples



Amazon
EC2

*If CPU utilization is **> 60%** for **5** minutes…*

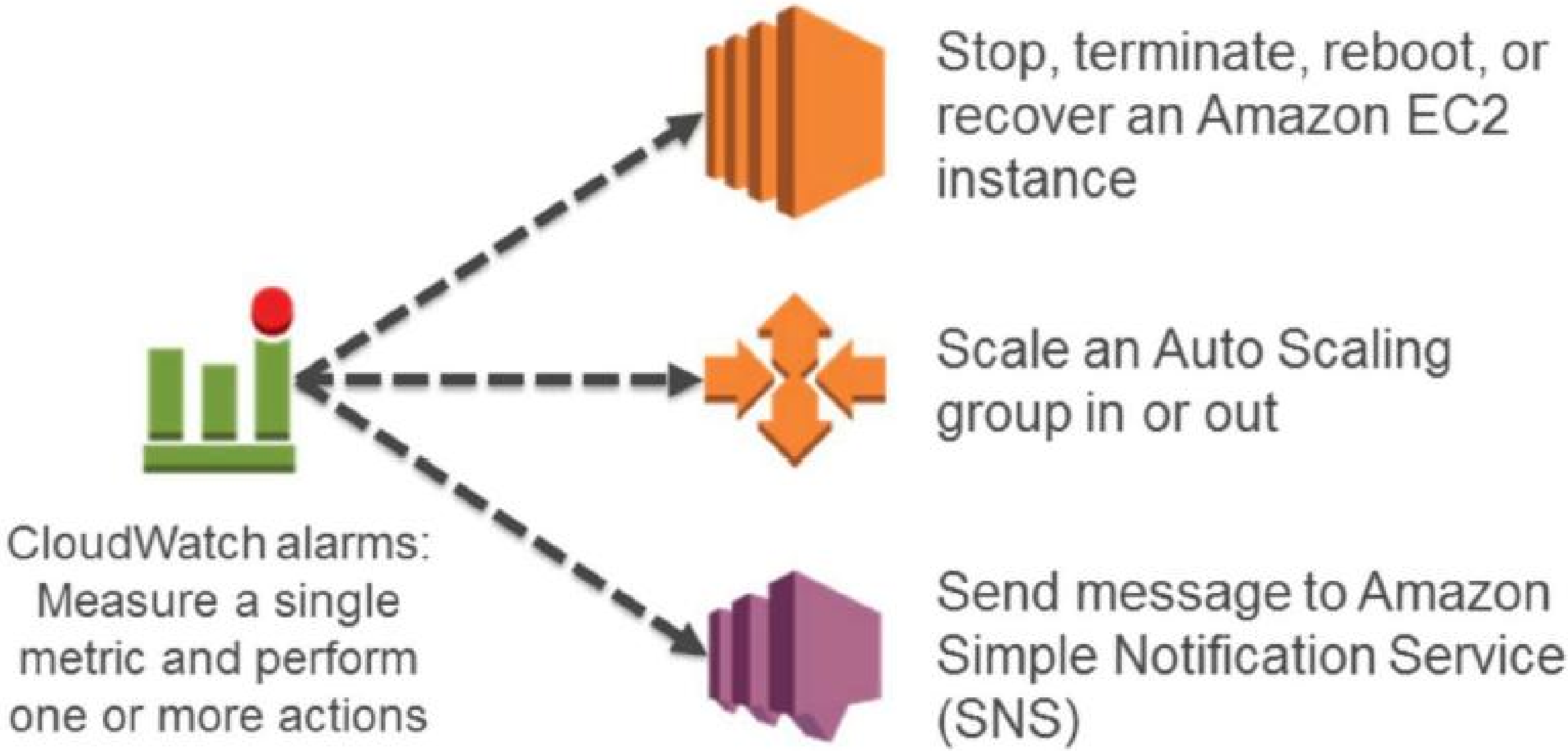
Amazon
RDS

*If number of simultaneous connections is **> 10** for **one** minute…*

Amazon
ELB

*If number of healthy hosts is **< 5** for **10** minutes…*

Cloud Watch Alarms and Actions



CloudWatch alarms: Measure a single metric and perform one or more actions

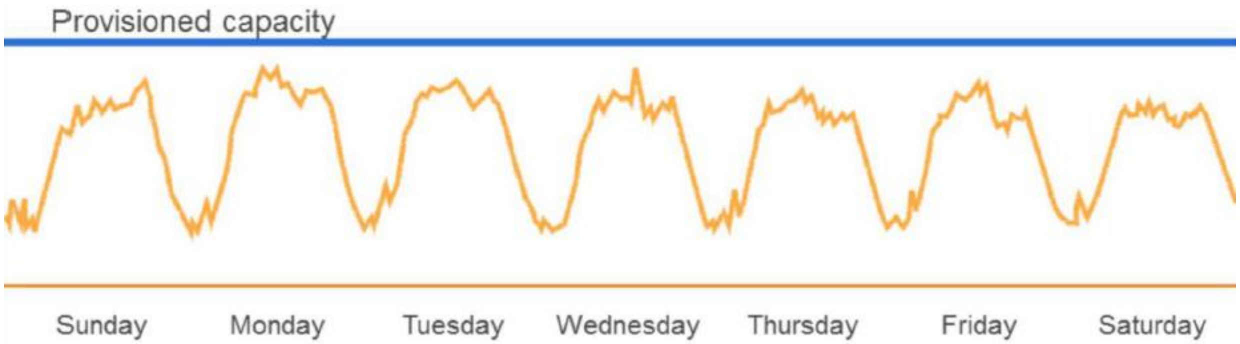
Stop, terminate, reboot, or recover an Amazon EC2 instance

Scale an Auto Scaling group in or out

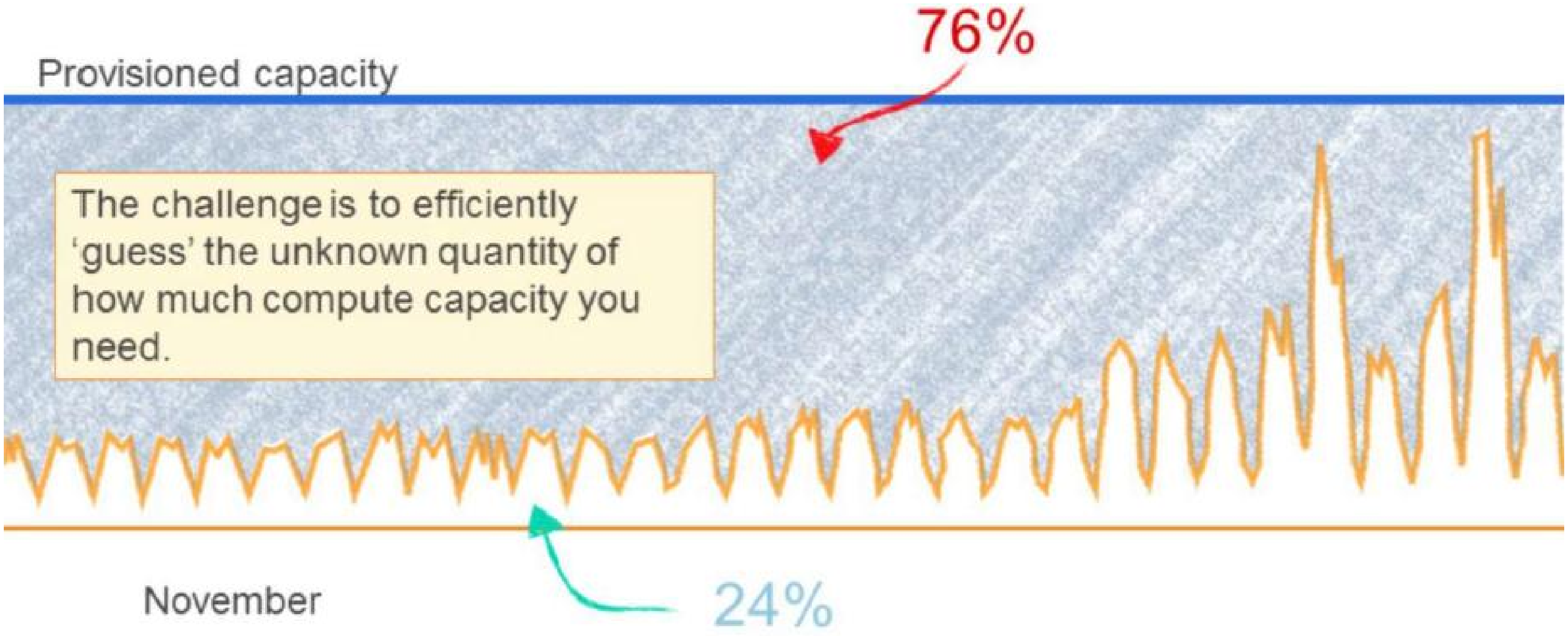
Send message to Amazon Simple Notification Service (SNS)

Autoscaling

Traffic Profile For A Major E-Commerce Site



Typical Week



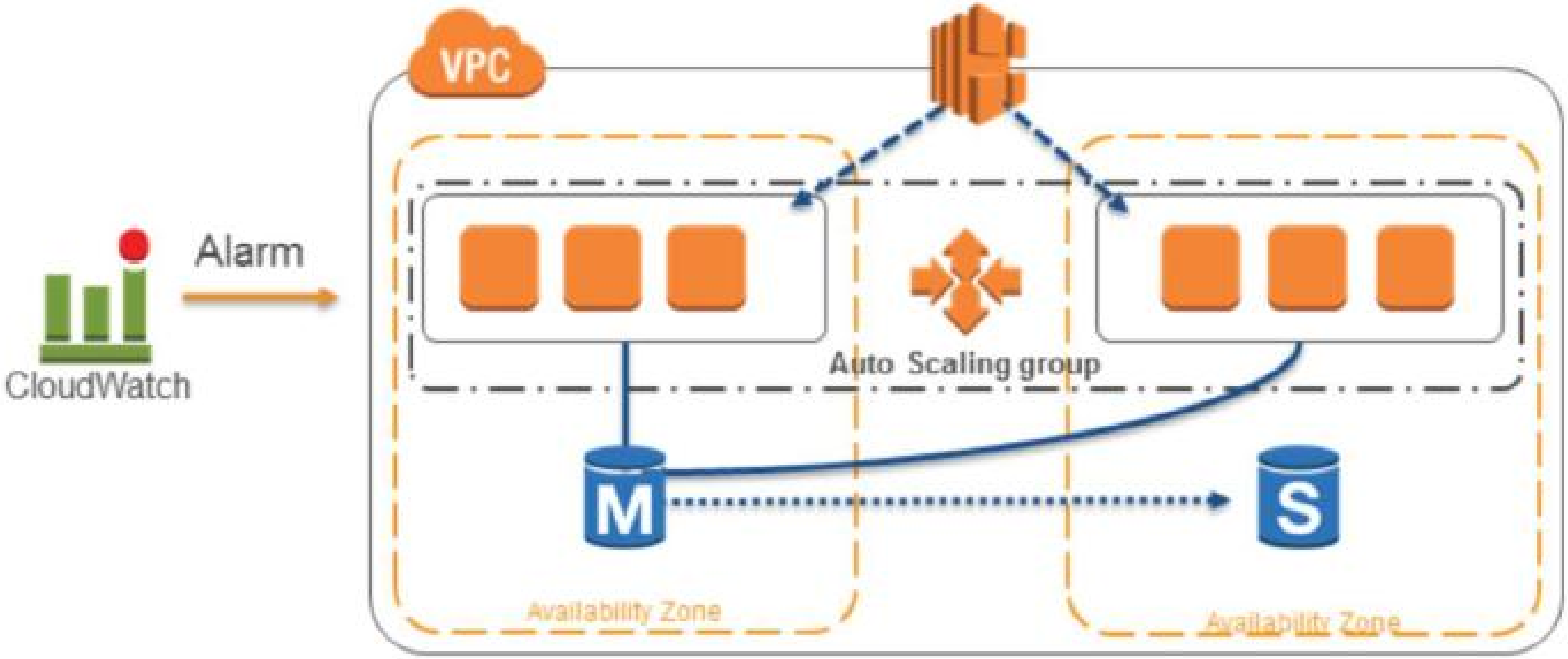
The challenge is to efficiently 'guess' the unknown quantity of how much compute capacity you need.

Auto Scaling

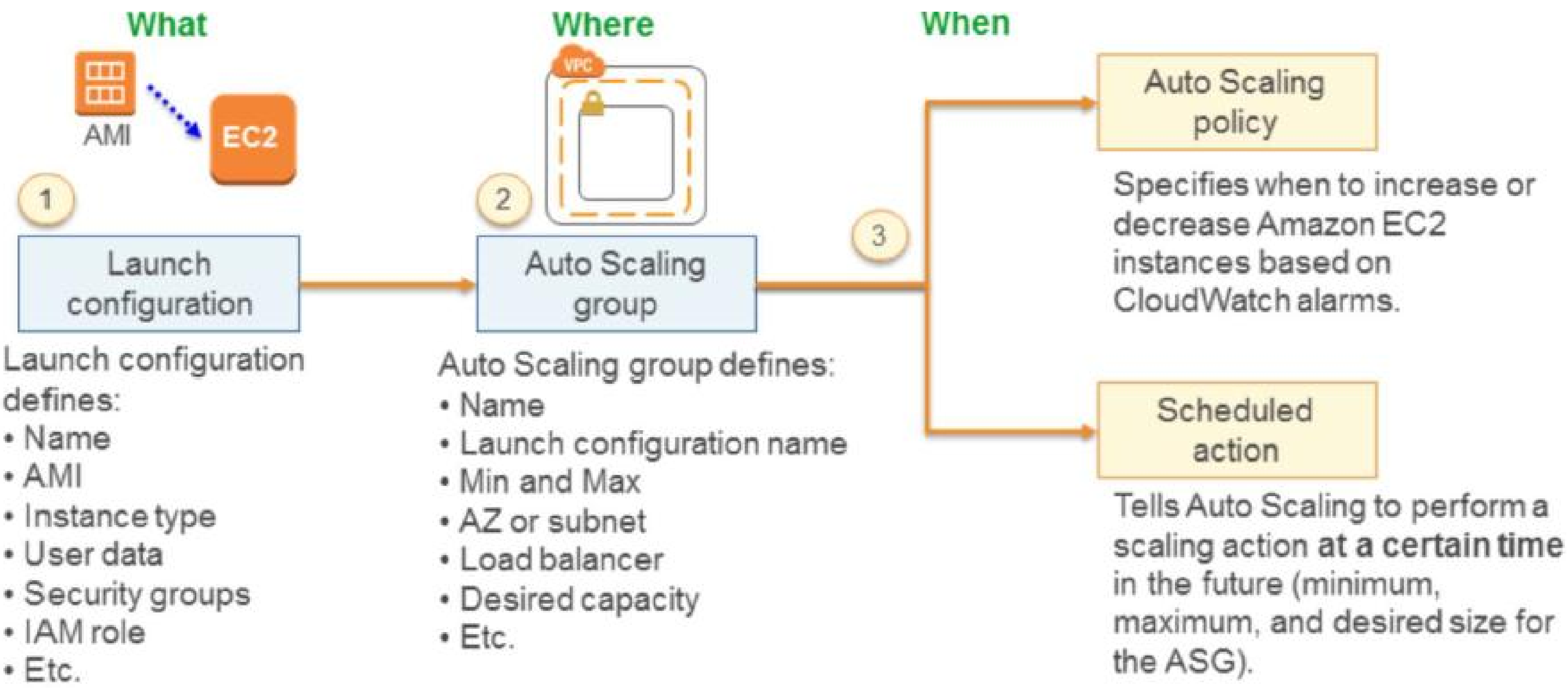
- Launches or terminates instances based on specified conditions.
- Automatically registers new instances with load balancers when specified.
- Can launch across Availability Zones.

No More Guessing About Resource Needs

Build a **flexible** system that will react to changes in customer demand and manage costs dynamically.



How Does Auto Scaling Work?



What

Launch configuration

Launch configuration defines:

- Name
- AMI
- Instance type
- User data
- Security groups
- IAM role
- Etc.

Where

Auto Scaling group

Auto Scaling group defines:

- Name
- Launch configuration name
- Min and Max
- AZ or subnet
- Load balancer
- Desired capacity
- Etc.

When

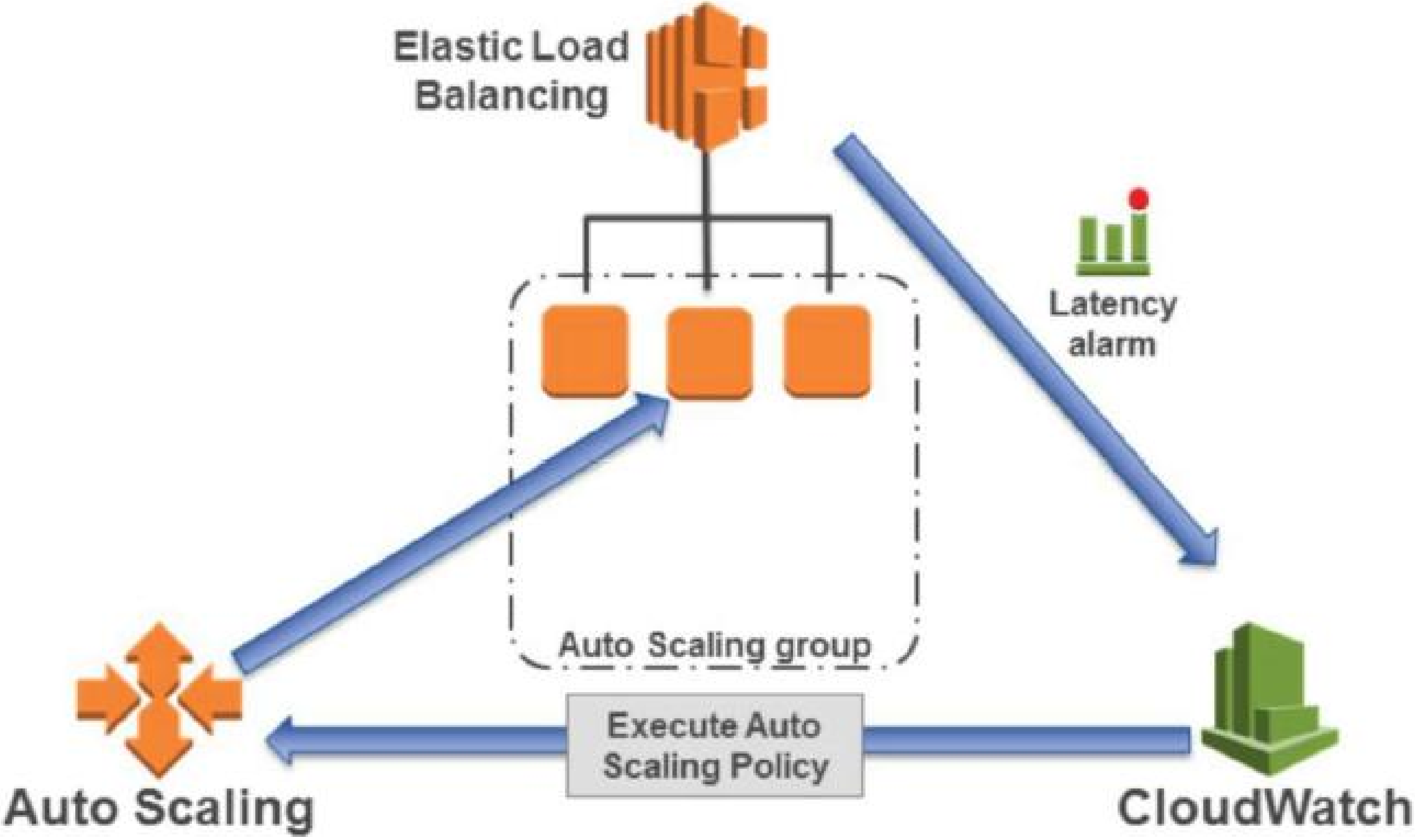
Auto Scaling policy

Specifies when to increase or decrease Amazon EC2 instances based on CloudWatch alarms.

Scheduled action

Tells Auto Scaling to perform a scaling action **at a certain time** in the future (minimum, maximum, and desired size for the ASG).

ELB + AS + CW: More Power Putting Connecting The Services



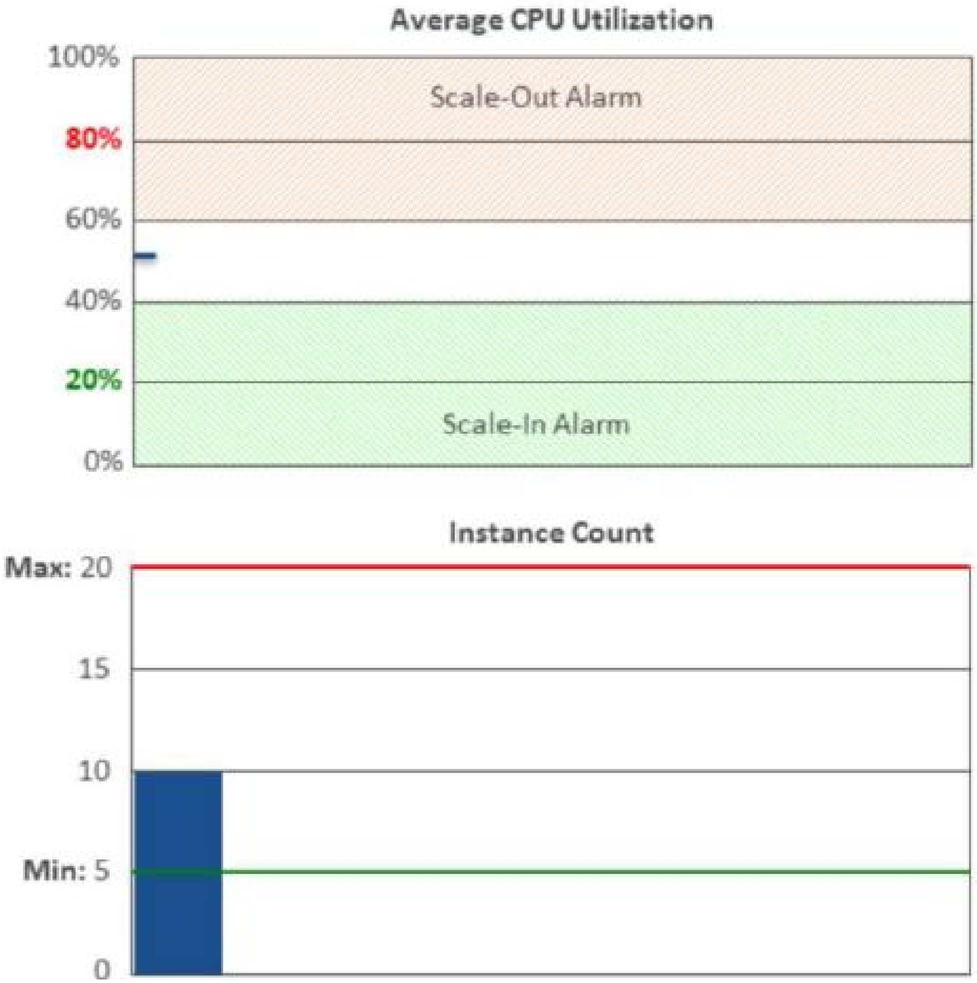
Auto Scaling Steps

Alarms:

- Add 2 instances when average CPU is 80-100%
- Add 1 instance when average CPU is 60-80%
- Remove 1 instance when average CPU is 20-40%
- Remove 2 instances when average CPU is 0-20%

Limits:

- Minimum: 5 instances
- Maximum: 20 instances



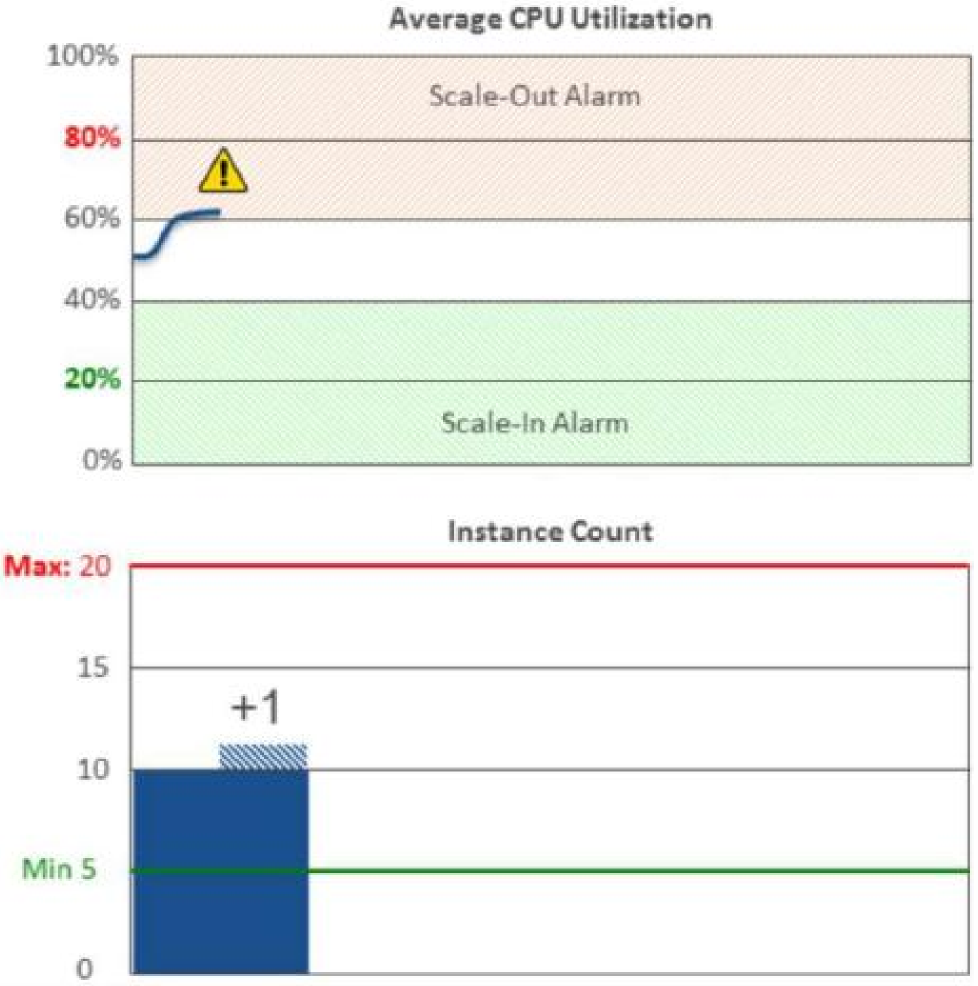
Auto Scaling Steps

As usage increases:

- CPU utilization goes up.

When CPU utilization is 60-80%:

- Scale-out alarm is triggered.
- Add 1 step policy is applied.
- New instance is launched but not added to the aggregated group metrics until after warm up period expires.



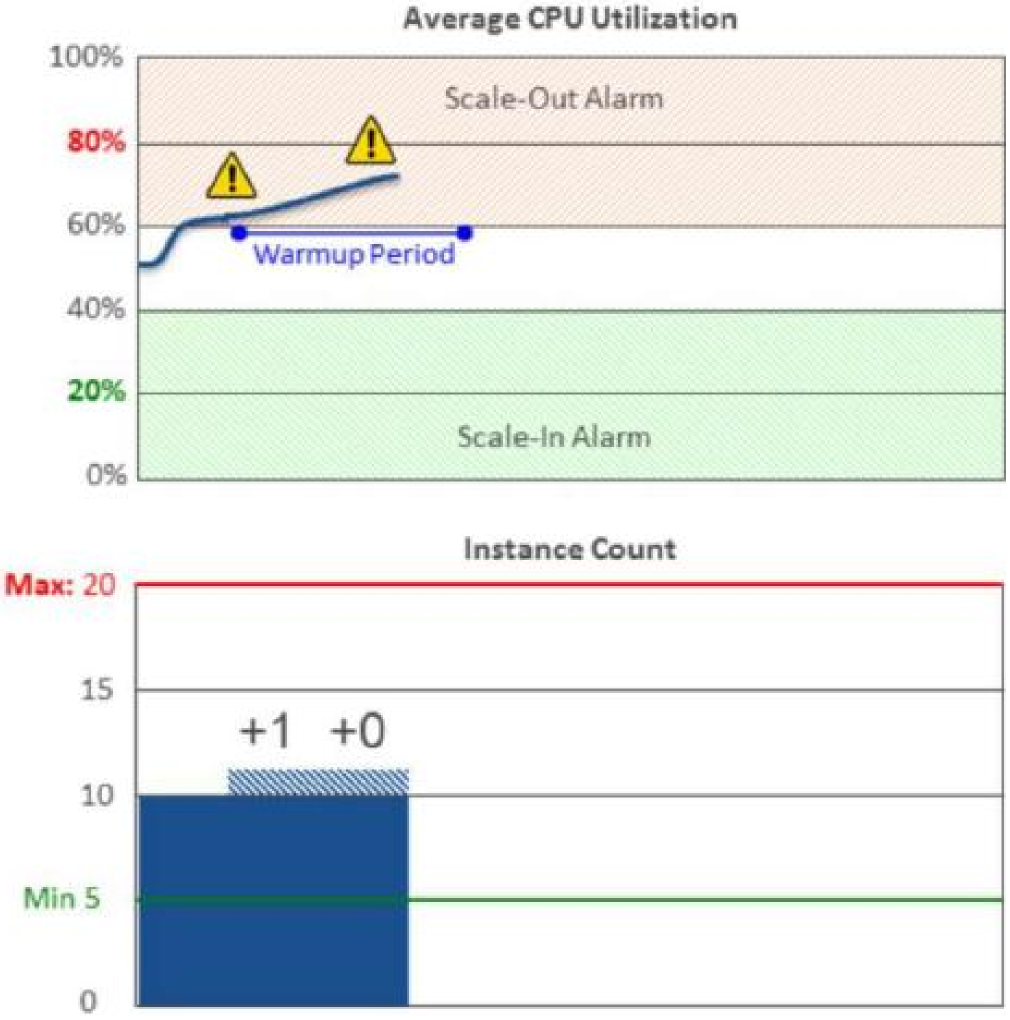
Auto Scaling Steps

As usage increases:

- CPU utilization goes up.

While waiting for new instance:

- CPU utilization remains high.
- Another alarm period is triggered.
- Since current capacity is still 10 during the warmup period, and desired capacity is already 11, no additional instances are launched.



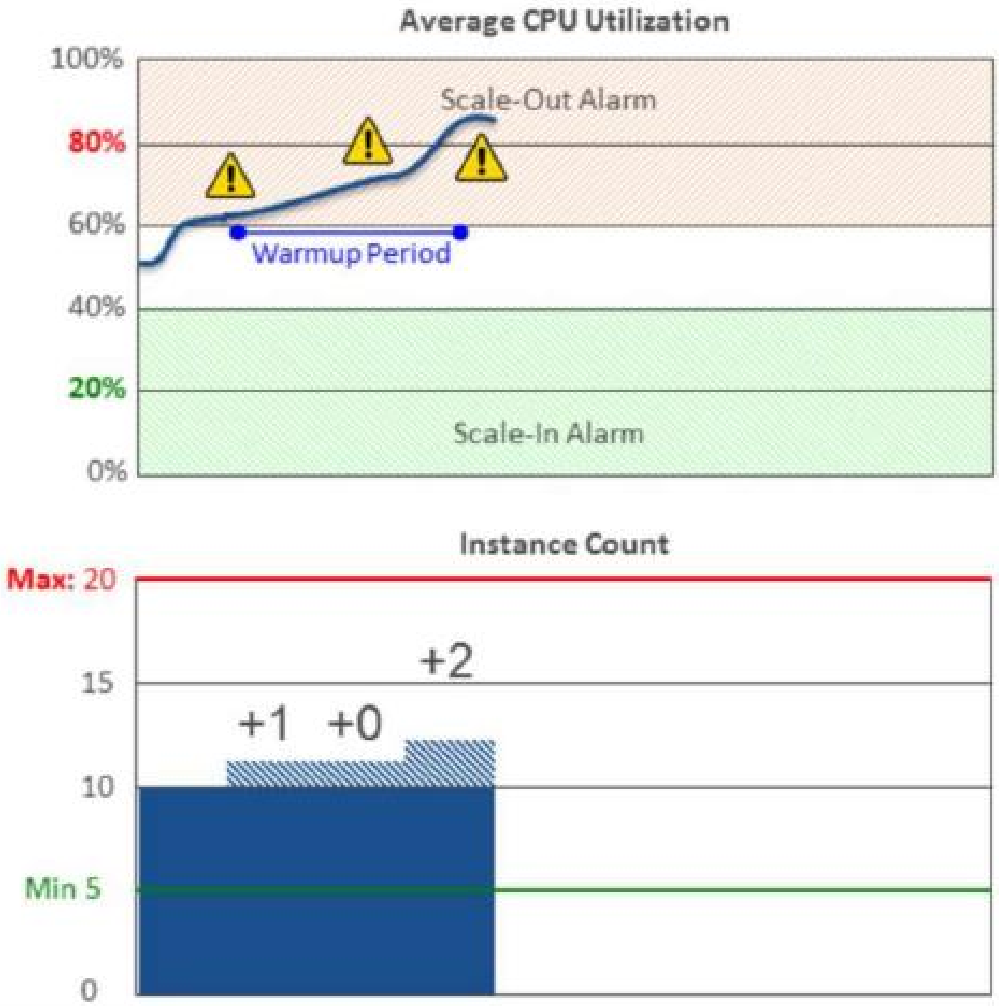
Auto Scaling Steps

As usage increases further:

- CPU utilization goes up.

When CPU utilization is 80-100%:

- Scale-out alarm is triggered.
- Add 2 step policy is applied.
- Since the alarm occurred during a warm up period, two instances are launched less the one instance added during the first alarm.
- Again new instances are not added to aggregated group metrics.



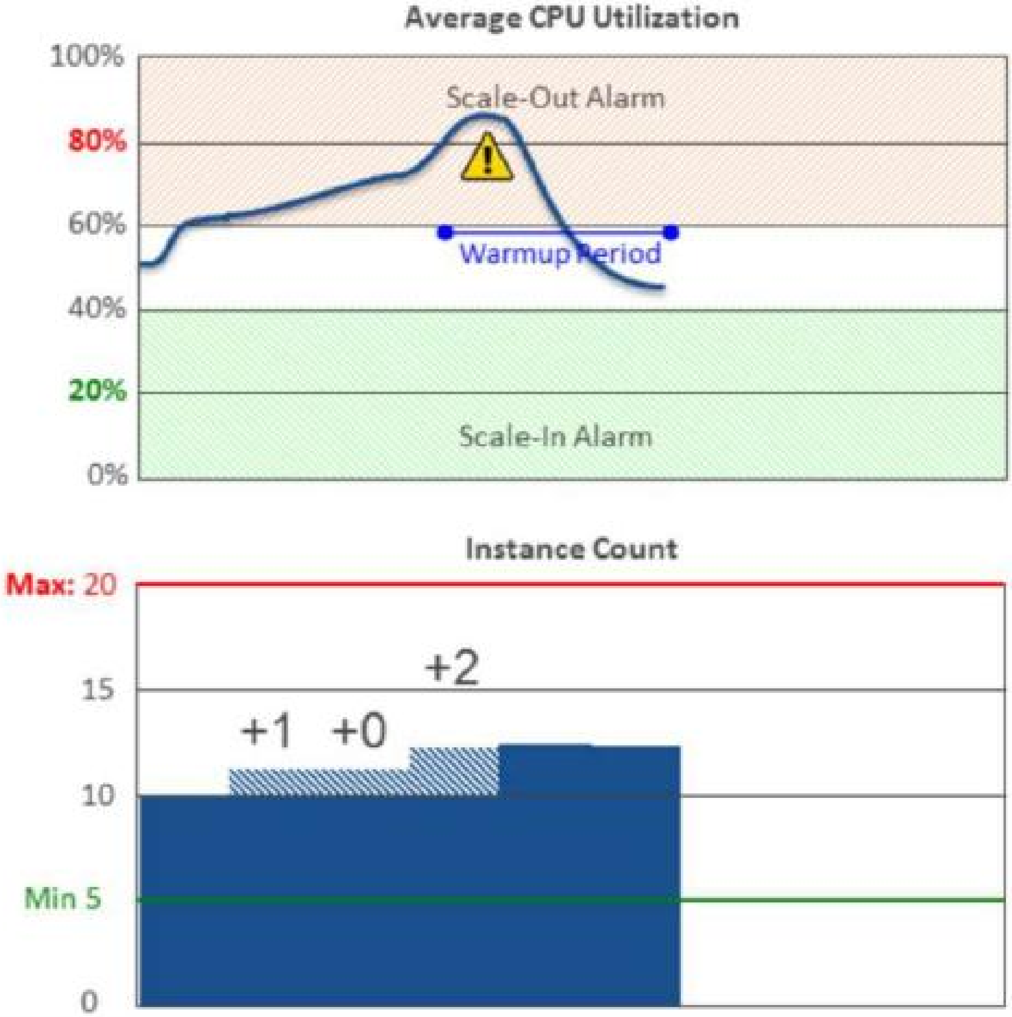
Auto Scaling Steps

As capacity matches usage:

- CPU utilization stabilizes.

When CPU utilization is 40-60%:

- No alarms are triggered.
- After warm up periods expire, new instances are added to the aggregated group metrics.



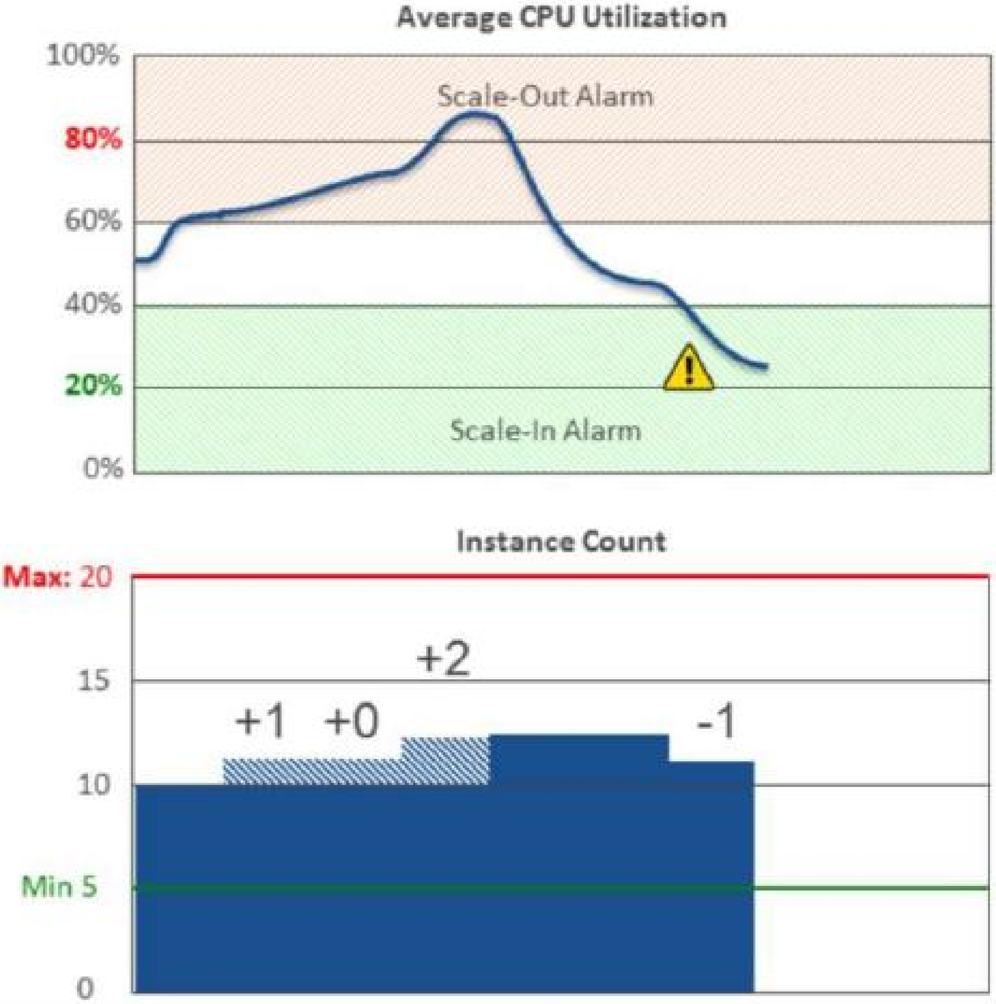
Auto Scaling Steps

As usage decreases:

- CPU utilization goes down.

When CPU utilization is 20-40%:

- Scale-in alarm is triggered.
- Remove 1 step policy is applied.
- An instance is removed from the Auto Scaling group and from the aggregated group metrics.



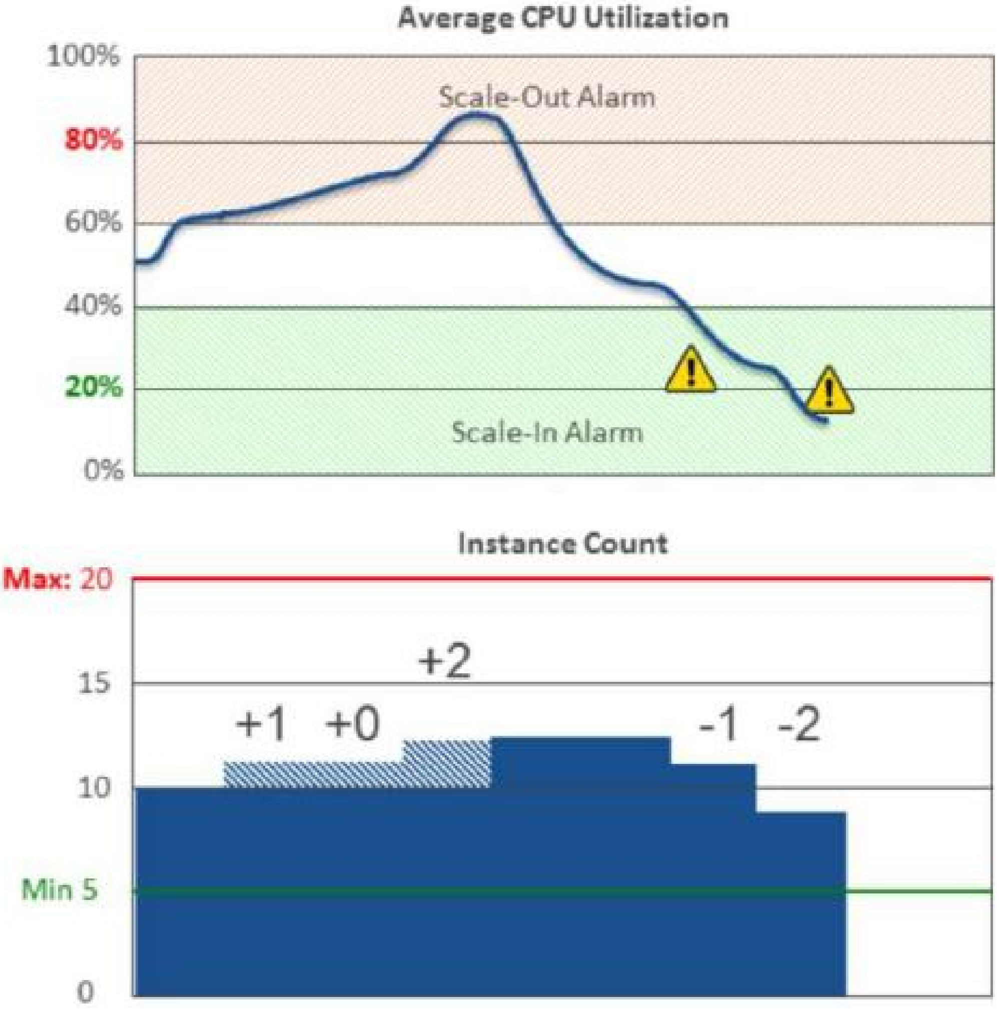
Auto Scaling Steps

As usage decreases:

- CPU utilization goes down further.

When CPU utilization is 0-20%:

- Scale in alarm is triggered.
- Remove 2 step policy is applied.
- Two instances are removed from the Auto Scaling group and from the aggregated group metrics.



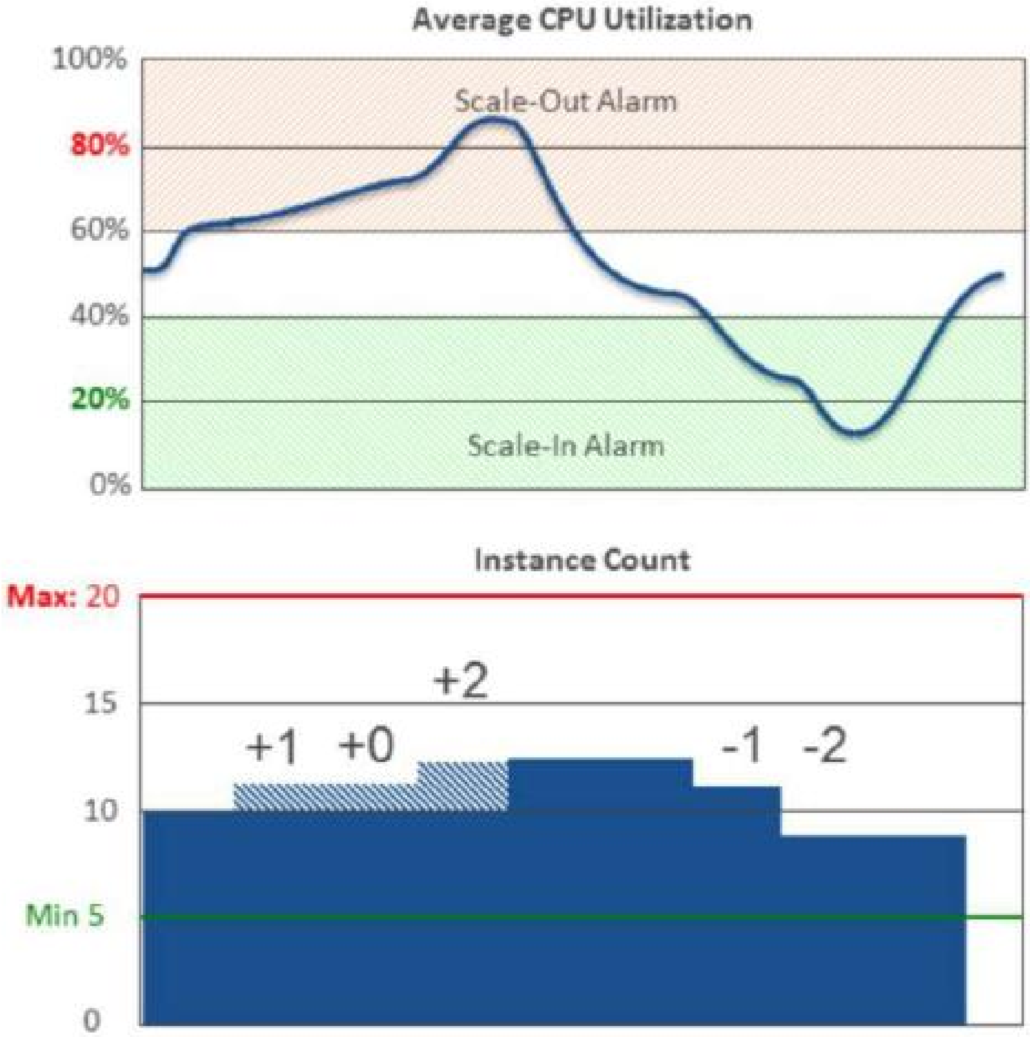
Auto Scaling Steps

As capacity matches usage:

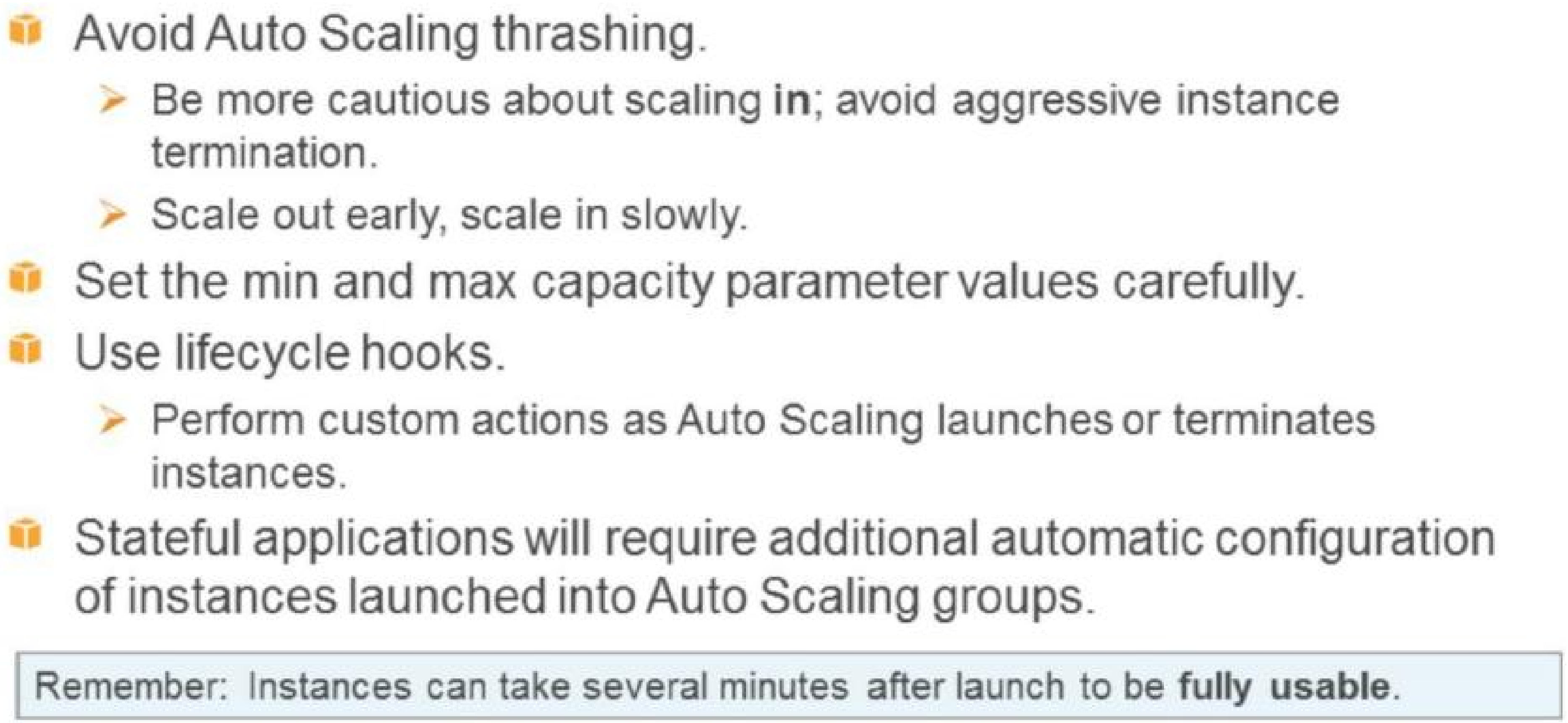
- CPU utilization stabilizes.

When 40% < CPU Utilization < 60%

- No alarm is triggered.
- No step policies are applied.
- No instances are added or removed from service.



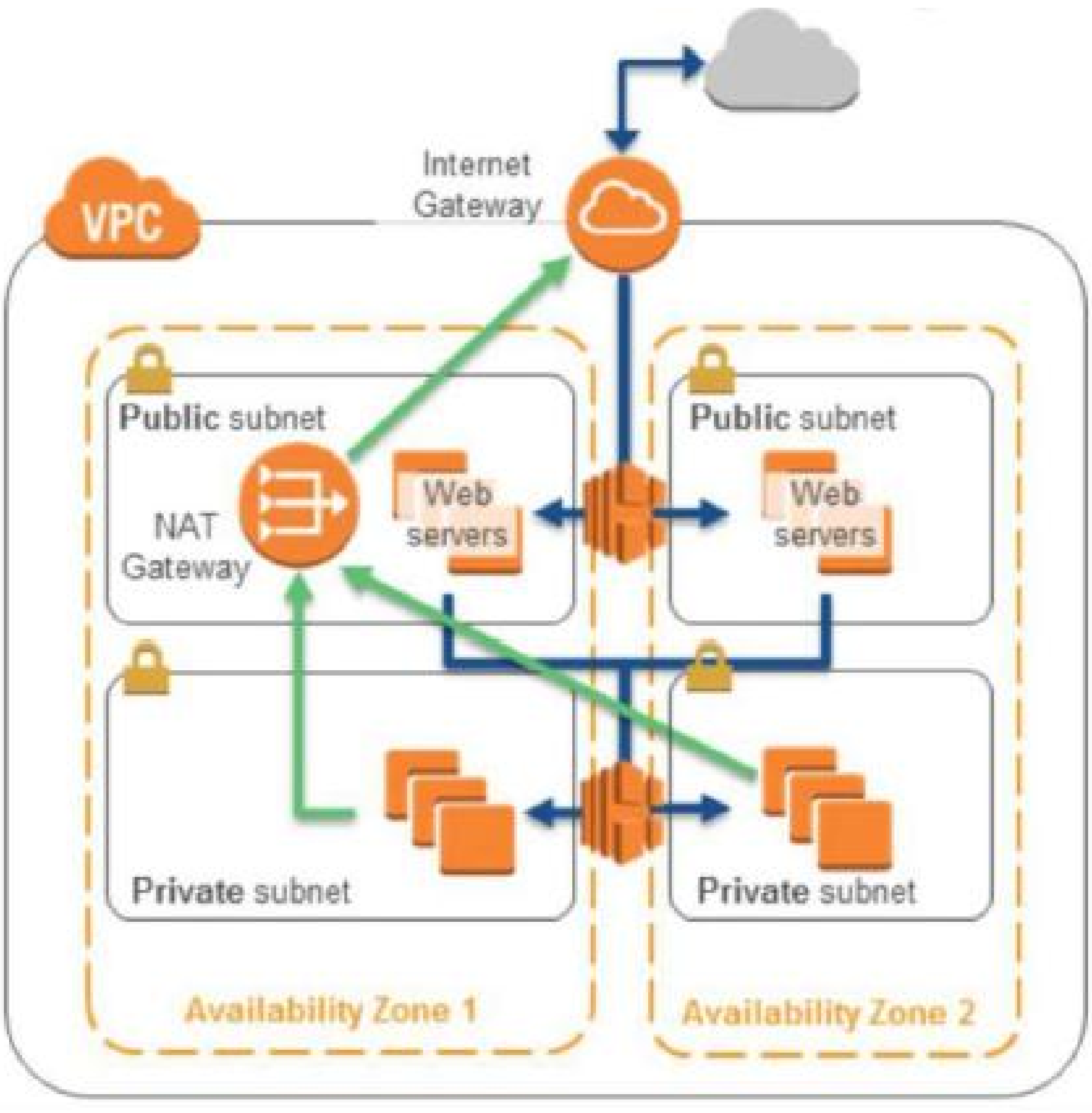
Auto Scaling Considerations



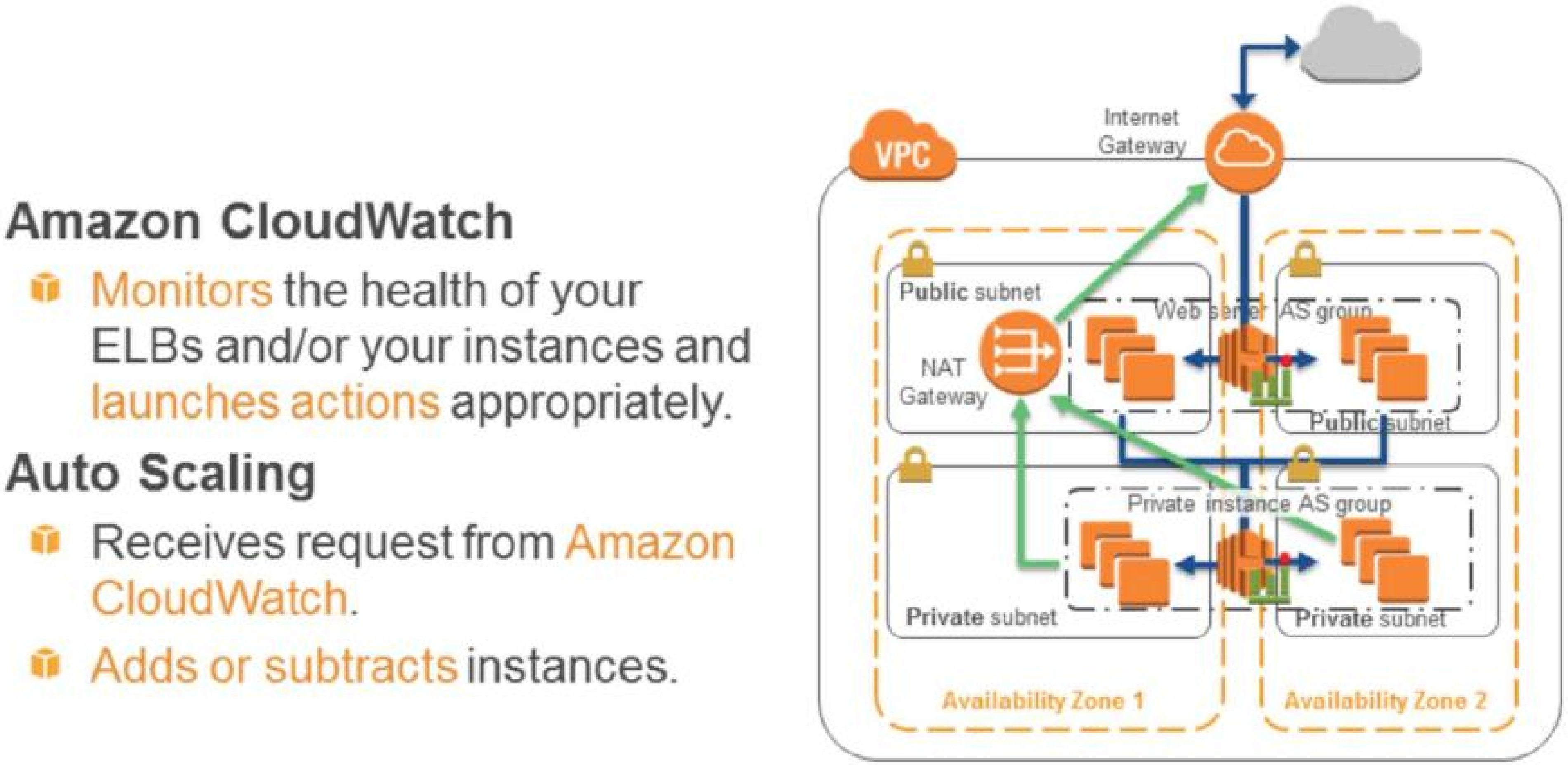
- Avoid Auto Scaling thrashing.
  - Be more cautious about scaling **in**; avoid aggressive instance termination.
  - Scale out early, scale in slowly.
- Set the min and max capacity parameter values carefully.
- Use lifecycle hooks.
  - Perform custom actions as Auto Scaling launches or terminates instances.
- Stateful applications will require additional automatic configuration of instances launched into Auto Scaling groups.

Remember: Instances can take several minutes after launch to be **fully usable**.

How Can You Increase Availability Of This Configuration?



HA With Auto Scaling and Cloud Watch



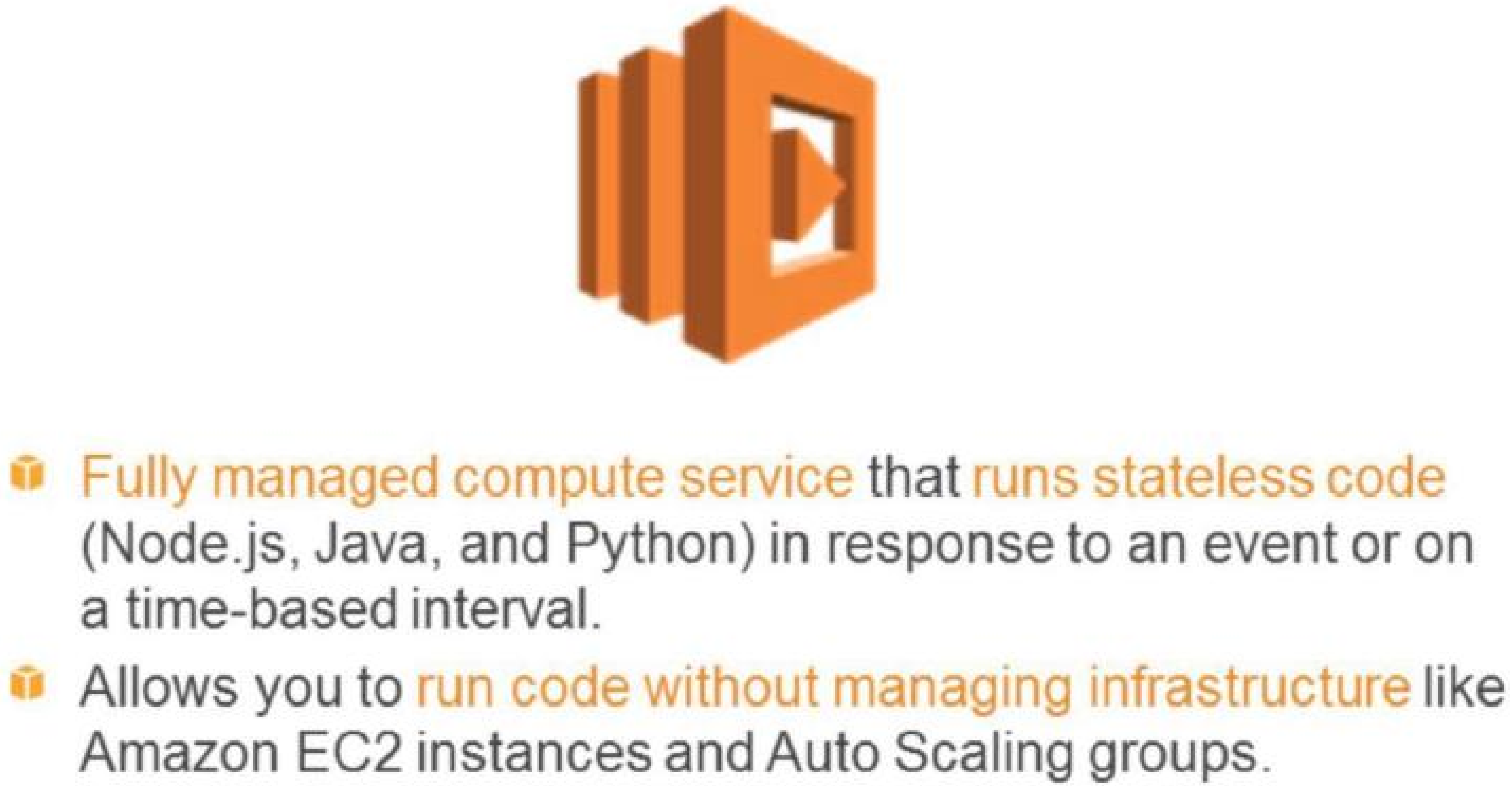
Amazon CloudWatch

- Monitors the health of your ELBs and/or your instances and launches actions appropriately.

Auto Scaling

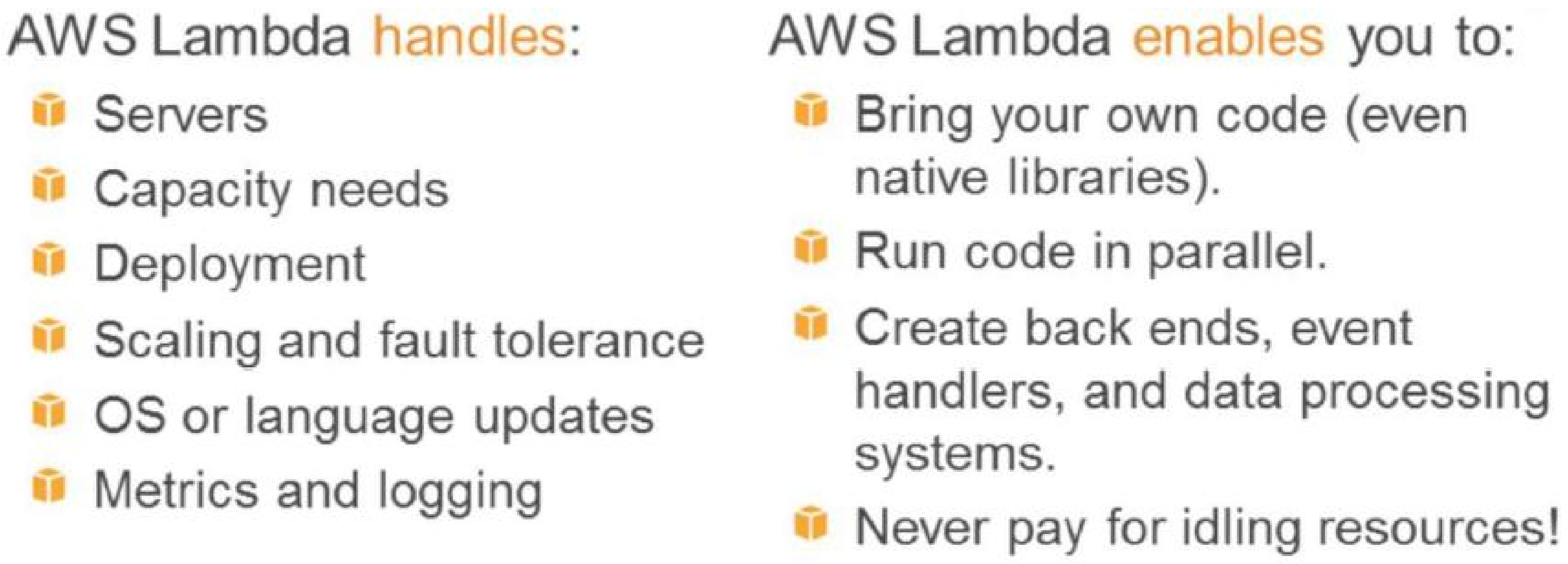
- Receives request from Amazon CloudWatch.
- Adds or subtracts instances.

AWS Lambda and Event Driven Scaling



- Fully managed compute service that runs stateless code (Node.js, Java, and Python) in response to an event or on a time-based interval.
- Allows you to run code without managing infrastructure like Amazon EC2 instances and Auto Scaling groups.

AWS Lambda Service



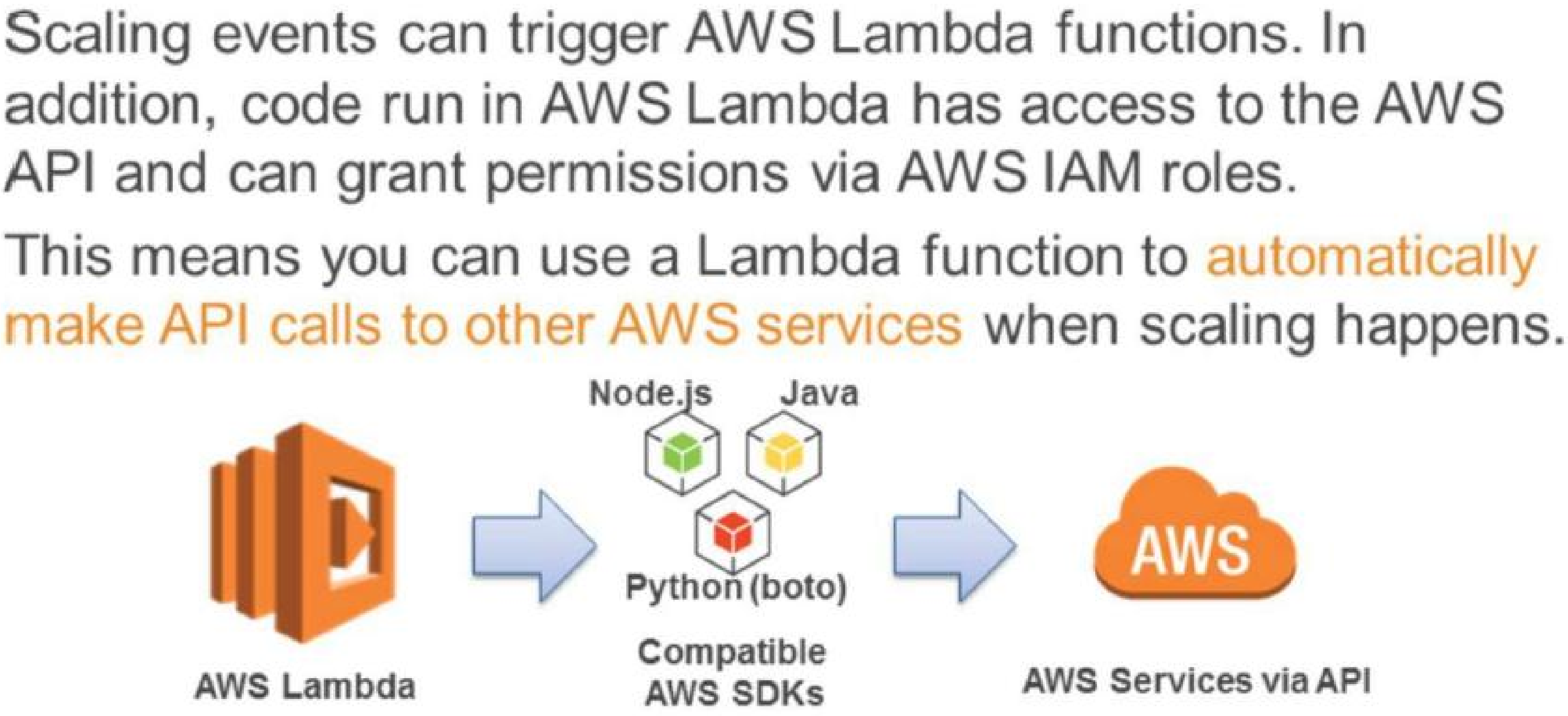
AWS Lambda handles:

- Servers
- Capacity needs
- Deployment
- Scaling and fault tolerance
- OS or language updates
- Metrics and logging

AWS Lambda enables you to:

- Bring your own code (even native libraries).
- Run code in parallel.
- Create back ends, event handlers, and data processing systems.
- Never pay for idling resources!

How AWS Lambda Can Be Used With Scaling?

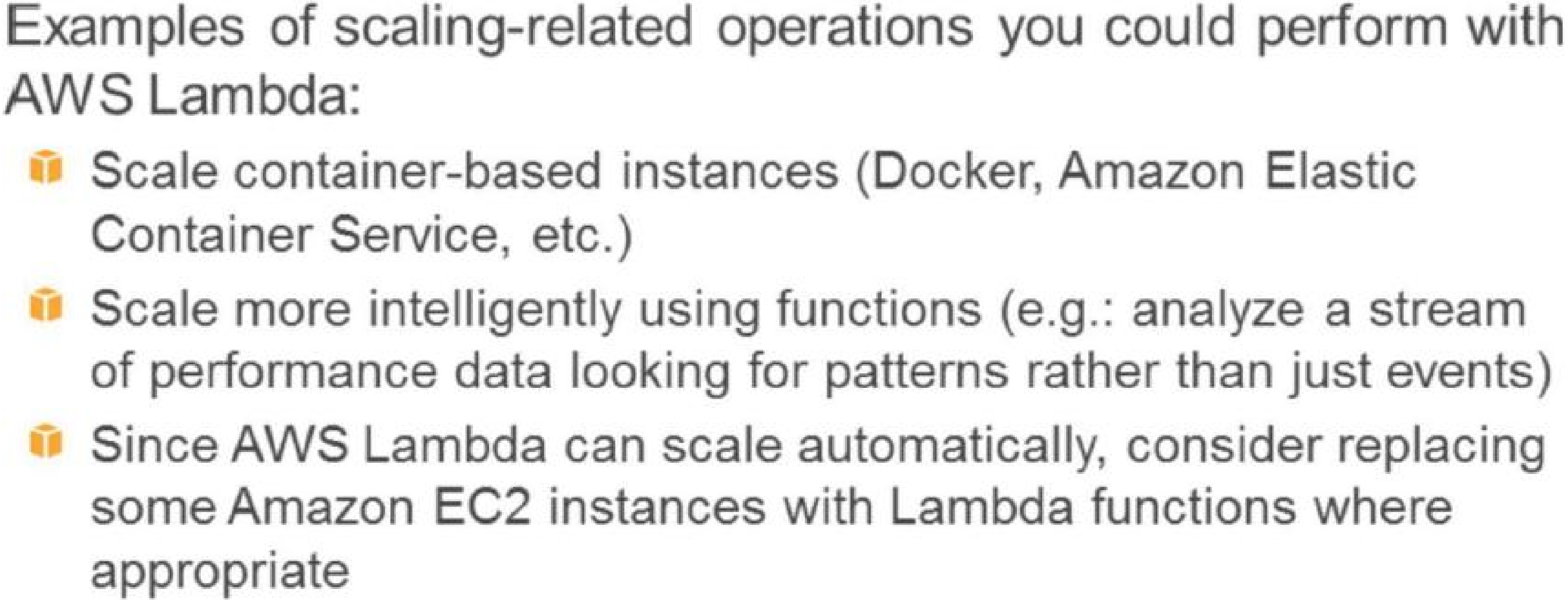


Scaling events can trigger AWS Lambda functions. In addition, code run in AWS Lambda has access to the AWS API and can grant permissions via AWS IAM roles.

This means you can use a Lambda function to automatically make API calls to other AWS services when scaling happens.

Node.js
Java
Python (boto)
AWS Lambda
Compatible AWS SDKs
AWS
AWS Services via API

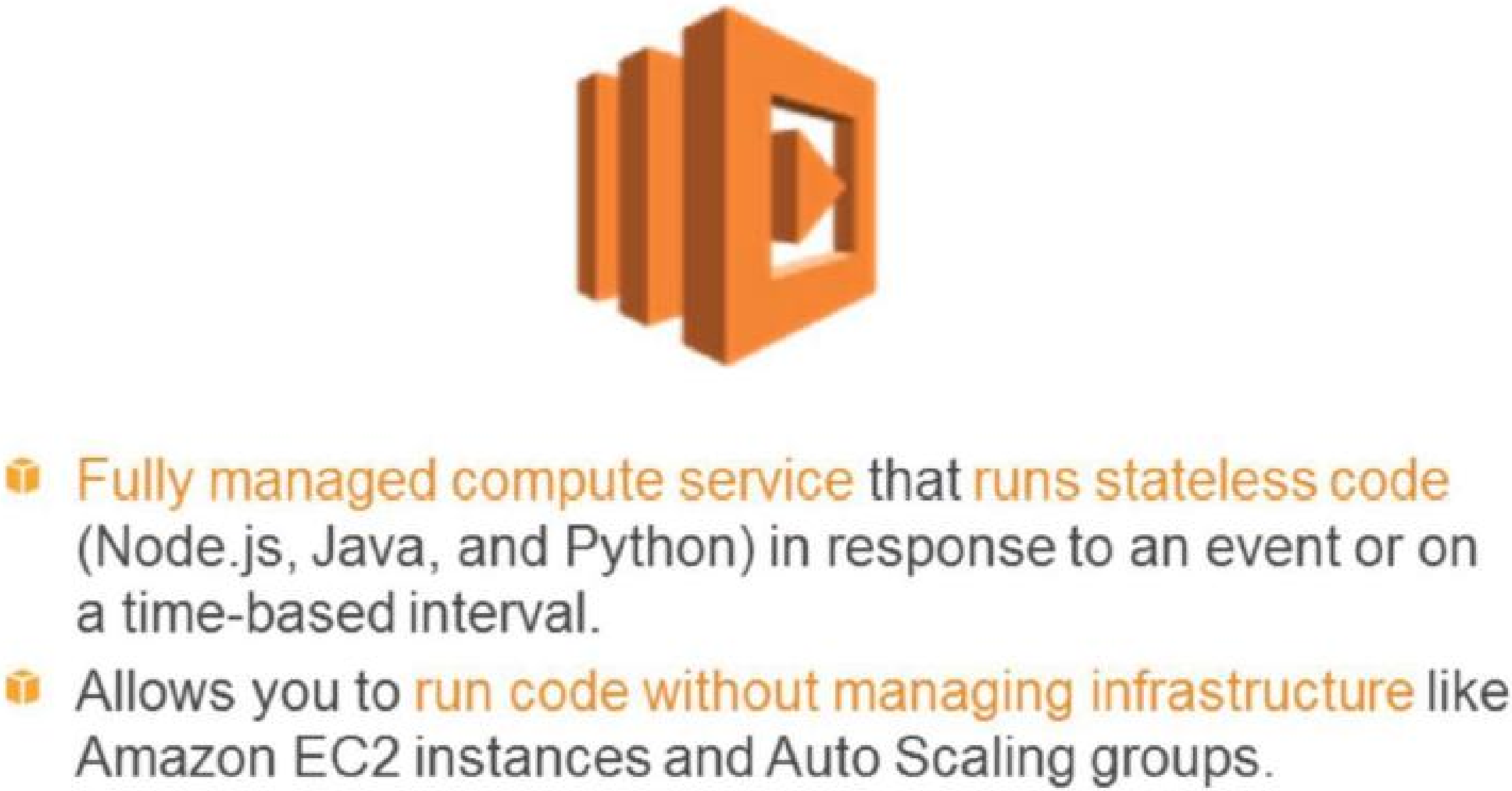
How AWS Lambda Can Be Used With Scaling?



Examples of scaling-related operations you could perform with AWS Lambda:

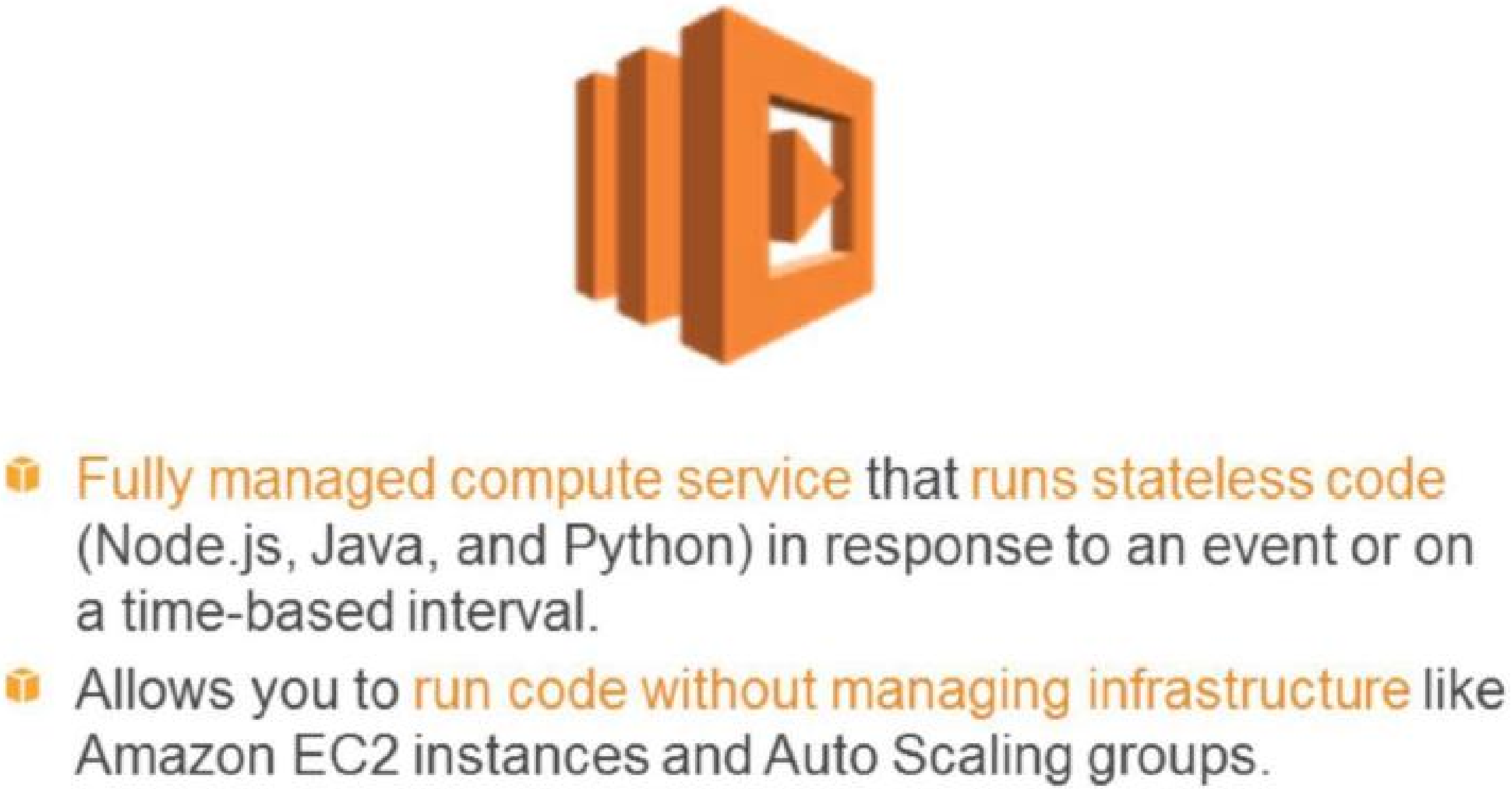
- Scale container-based instances (Docker, Amazon Elastic Container Service, etc.)
- Scale more intelligently using functions (e.g.: analyze a stream of performance data looking for patterns rather than just events)
- Since AWS Lambda can scale automatically, consider replacing some Amazon EC2 instances with Lambda functions where appropriate

AWS Lambda and Event Driven Scaling



- Fully managed compute service that runs stateless code (Node.js, Java, and Python) in response to an event or on a time-based interval.
- Allows you to run code without managing infrastructure like Amazon EC2 instances and Auto Scaling groups.

AWS Lambda and Event Driven Scaling



- Fully managed compute service that runs stateless code (Node.js, Java, and Python) in response to an event or on a time-based interval.
- Allows you to run code without managing infrastructure like Amazon EC2 instances and Auto Scaling groups.

Resources

- AWS Best Practices: https://d0.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf

- AWS Well Architected Framework: https://d0.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

- AWS SlideShare Channel: https://www.slideshare.net/AmazonWebServices/