All You Need to Know About Auto Scaling

(based on CMP 201 from re:Invent 2015)

Julien Simon
Principal Technical Evangelist
Amazon Web Services

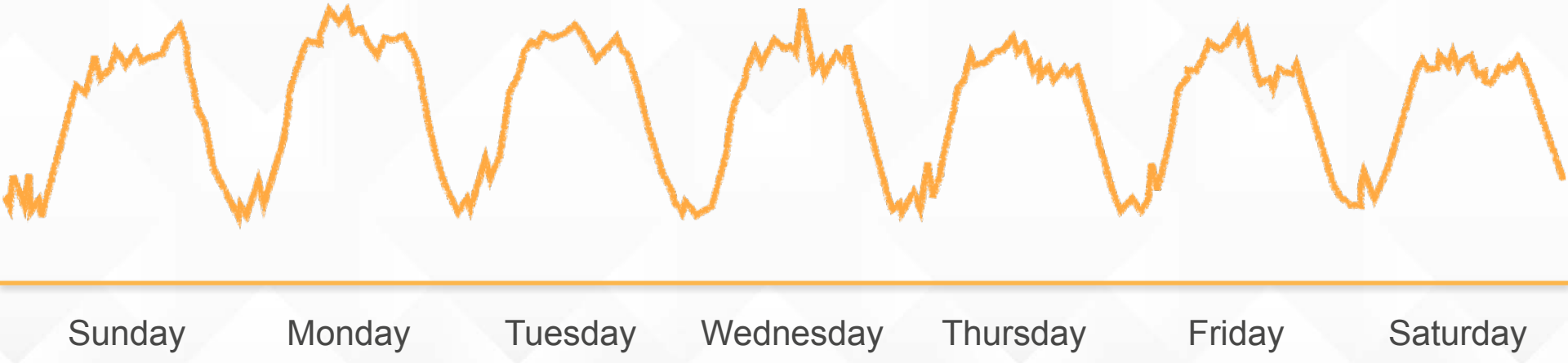
julsimon@amazon.fr
@julsimon

amazon web services | Pop-up Loft TEL AVIV

Auto Scaling – Why it’s needed

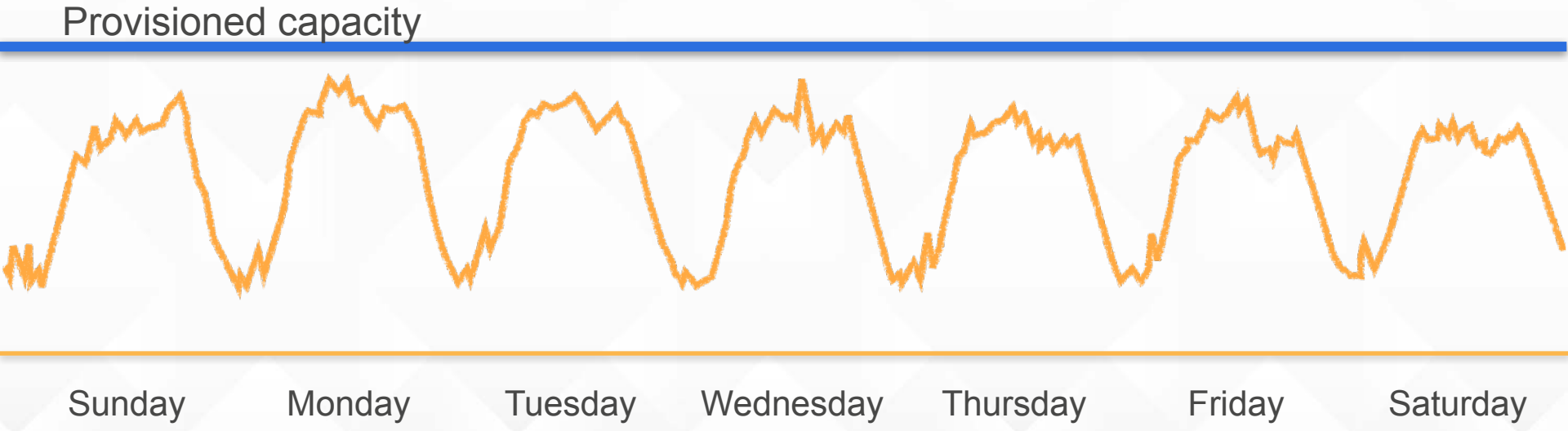
amazon web services

Typical weekly traffic to Amazon.com



amazon web services

Typical weekly traffic to Amazon.com



amazon web services

November traffic to Amazon.com

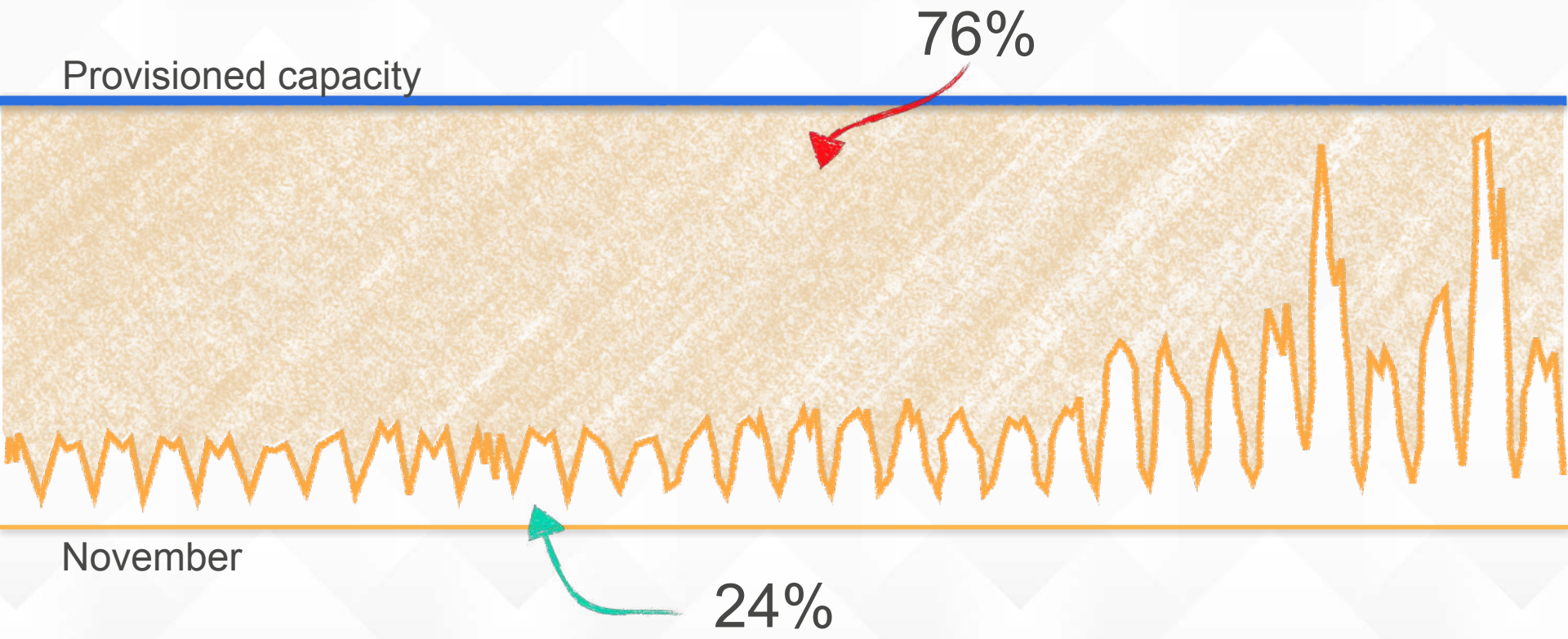
November

amazon web services

November traffic to Amazon.com

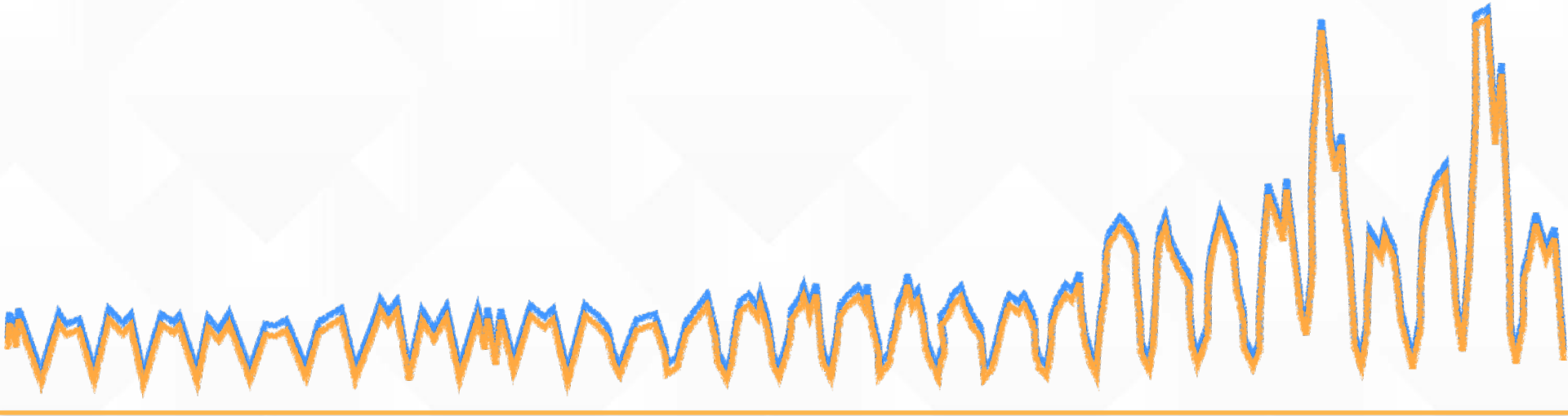
amazon web services

November traffic to Amazon.com



amazon web services

November traffic to Amazon.com



November

amazon web services

Auto Scaling – The Basics

amazon web services

Auto Scaling Concepts

Auto Scaling Groups

- EC2 instances are managed by Auto Scaling *groups*.
- Create Auto Scaling groups by defining the minimum, maximum, and, optionally, the desired number of running EC2 instances.

Launch Configuration

- Auto Scaling groups use a *launch configuration* to launch EC2 instances.
- Provides information about the AMI and EC2 instance types/size

Scaling Policy

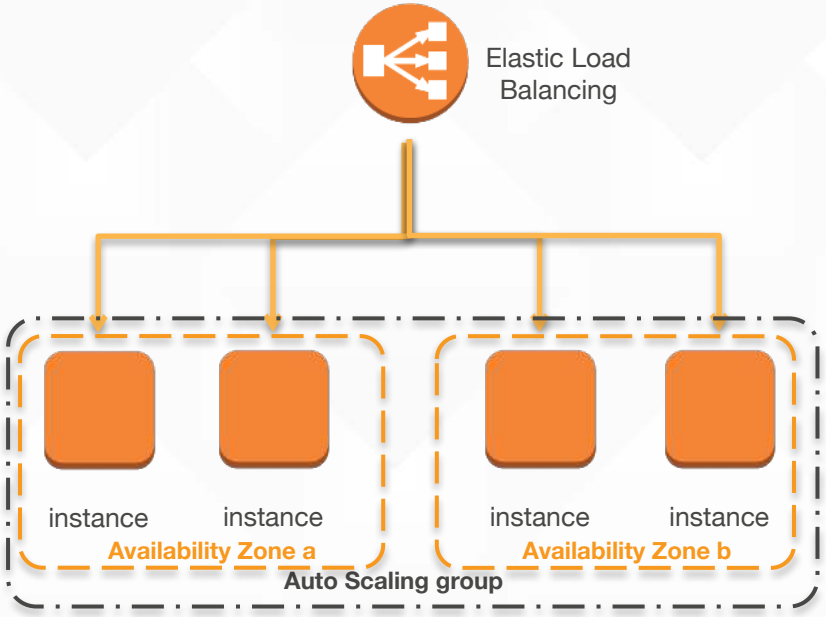
- A scaling policy tells Auto Scaling when and how to scale.
- Create a scaling policybased on the occurrence of specified conditions (dynamic scaling) or create a policy based on a specific schedule.

amazon web services

Auto Scaling Groups

- Always keep **minimum** number of instances running
- Launch or terminate instances to meet **desired capacity**
- Never start more than **maximum** number of instances
- Keeps capacity **balanced** across AZs

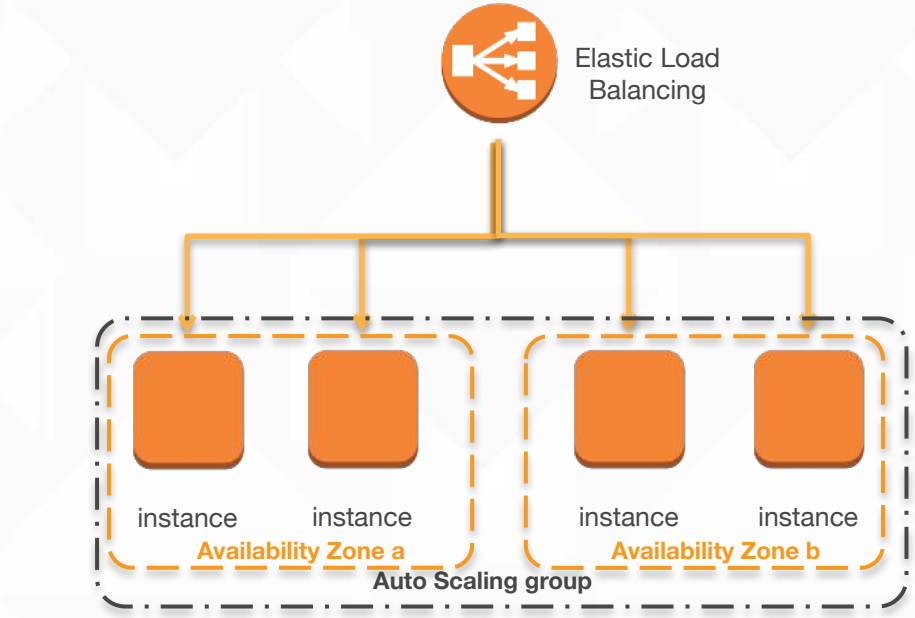
amazon web services



Minimum = 2

Maximum = 10

Desired # of instances = 4

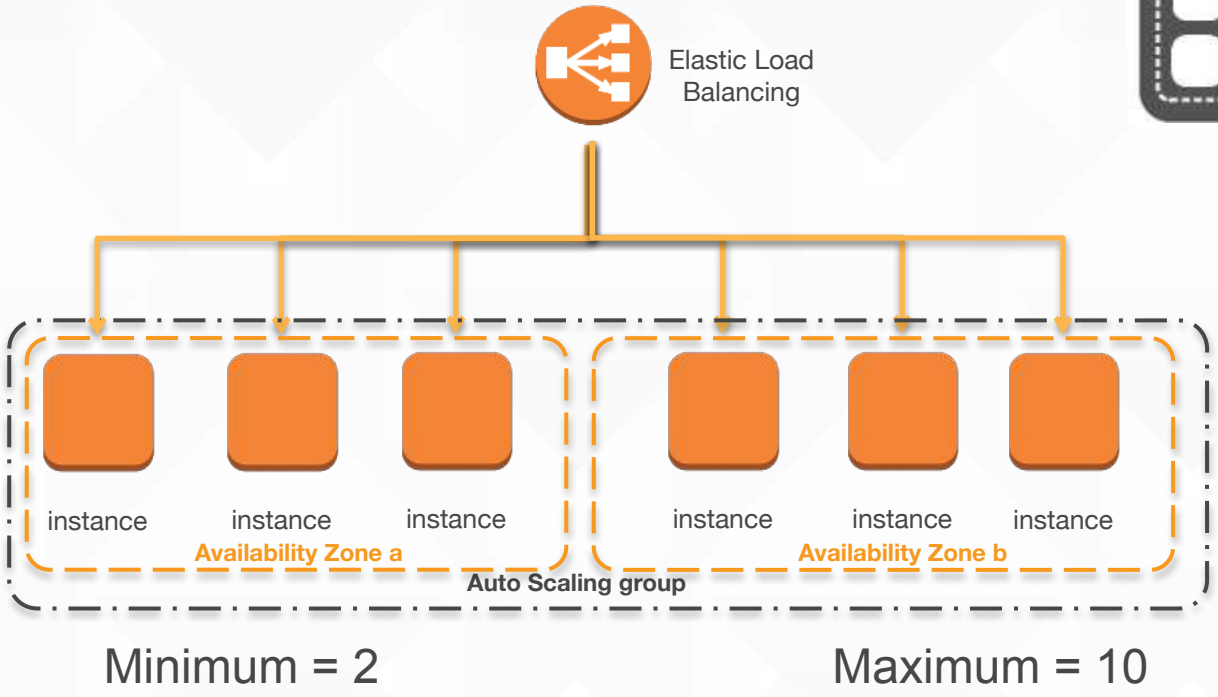


Minimum = 2

Maximum = 10

Desired # of instances = 6

amazon web services



Desired # of instances = 6

Launch Configurations

Determine **what** is going to be launched:

- EC2 instance type & size
- Amazon Machine Image (AMI)
- Security groups, SSH keys, IAM instance profile
- User data

…

amazon web services

Bootstrapping

Installation & setup needs to be fully automated:

- Use Amazon Machine Image (AMI) with all required configuration & software (“golden image”)
- Base AMI + install code & configuration as needed
  - Via Userdata
  - Via Chef/Puppet/Ansible/…
  - Using AWS CodeDeploy

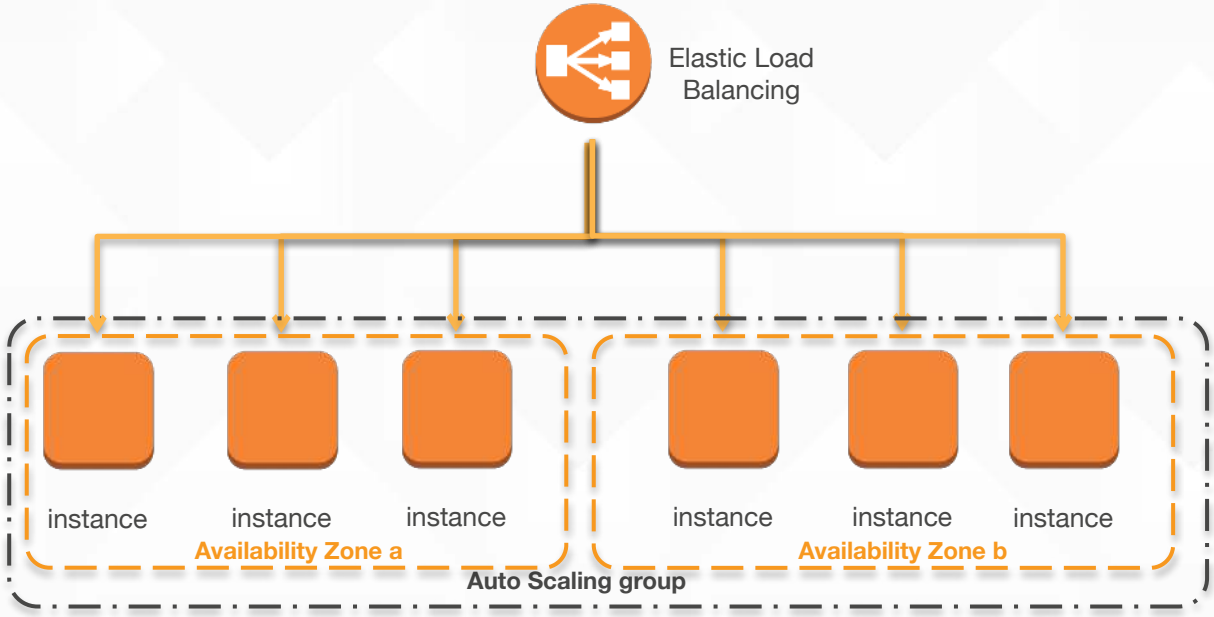
…

amazon web services

Bootstrapping

```
#!/bin/bash
yum -y update;
yum install -y ruby; yum install -y aws-cli;
cd /home/ec2-user;
aws s3 cp s3://aws-codedeploy-us-east-1/latest/ \ install . --region us-east-1;
chmod +x ./install;
./install auto;
```

amazon web services

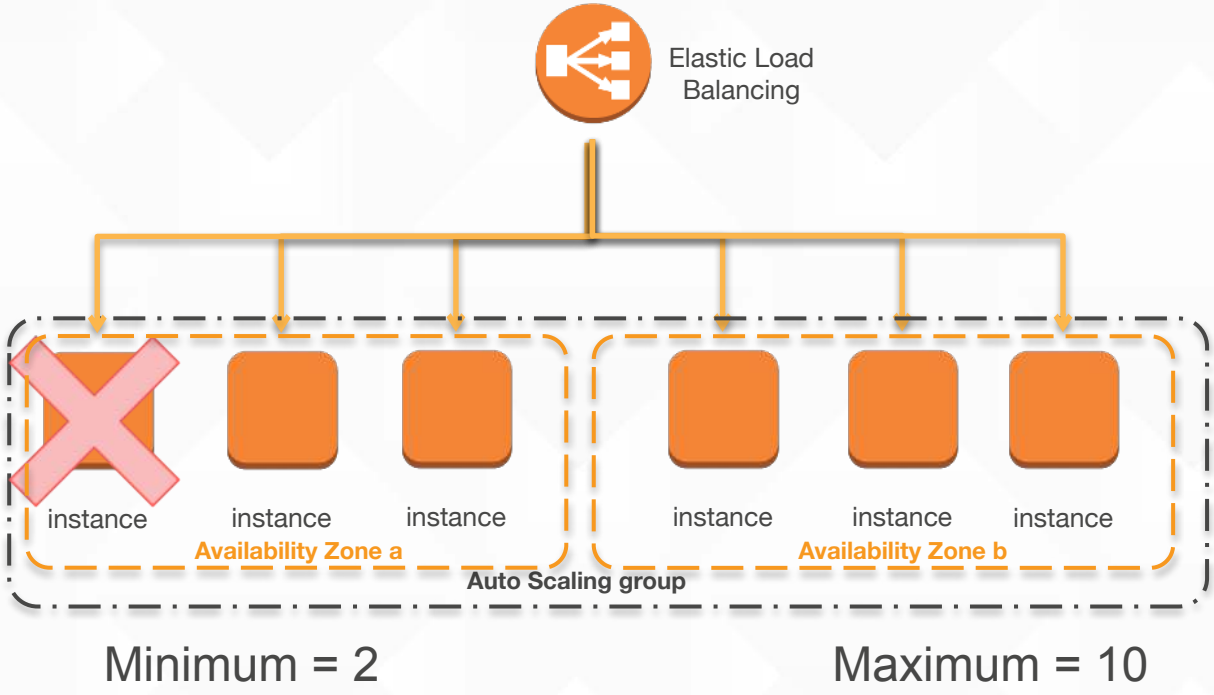


Minimum = 2

Maximum = 10

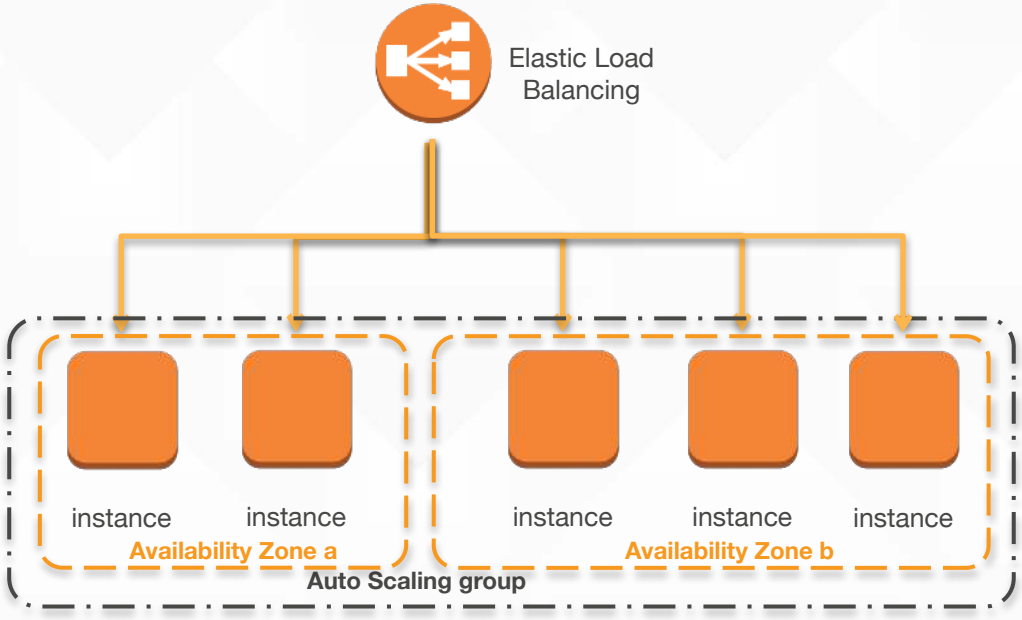
Desired # of instances = 6

amazon web services



Desired # of instances = 5

amazon web services



Minimum = 2

Maximum = 10

Desired # of instances **= 5**

amazon web services

Termination Policies

Determine which instances are terminated first:

- Longest running
- Oldest launch configuration
- Closest to full billing hour

But: rebalancing of capacity across AZs takes precedence!

amazon web services

Scaling Policies

Determine when the Auto Scaling group will scale in or out:

desired capacity > current capacity: launch instances
desired capacity < current capacity: terminate instances

amazon web services

Scaling Policies

- Default: ensure current capacity of **healthy** instances remains within boundaries (never less than minimum)
- ‘Manual scaling’: modify desired capacity (via API, console, CLI) to trigger a scaling event
- Scheduled: scale in / out based on timed events
- Dynamic scaling: scale on Amazon CloudWatch metrics

amazon web services

Integration with Amazon Elastic Load Balancing

amazon web services

Load Balance your Auto Scaling Group

Elastic Load Balancing

- Distribute incoming web traffic automatically.
- Single point of entry for your application.
- Sends data about your load balancers and EC2 instances to Amazon CloudWatch.
- Use Elastic Load Balancing metrics to scale your application.
- Use connection draining to wait for the in-flight requests to complete.

amazon web services

Load Balance your Auto Scaling Group

Elastic Load Balancing

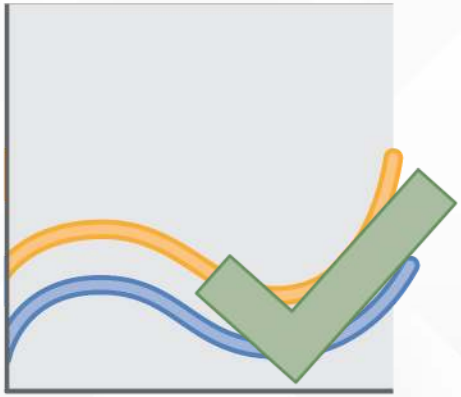
- Configure your Auto Scaling Group to work with one or more Elastic Load Balancers
- Automatically registers new instances, deregisters on termination
- Use ELB health checks in your Auto Scaling Group
- Use Elastic Load Balancing metrics in scaling policies

amazon web services

Benefits of Auto Scaling

Elasticity:

Automatically adapt capacity to demand



amazon web services

Availability & Reliability

amazon web services

Health Checks

- Performed periodically
- Instances are marked as unhealthy or healthy
- Unhealthy instances are terminated and replaced

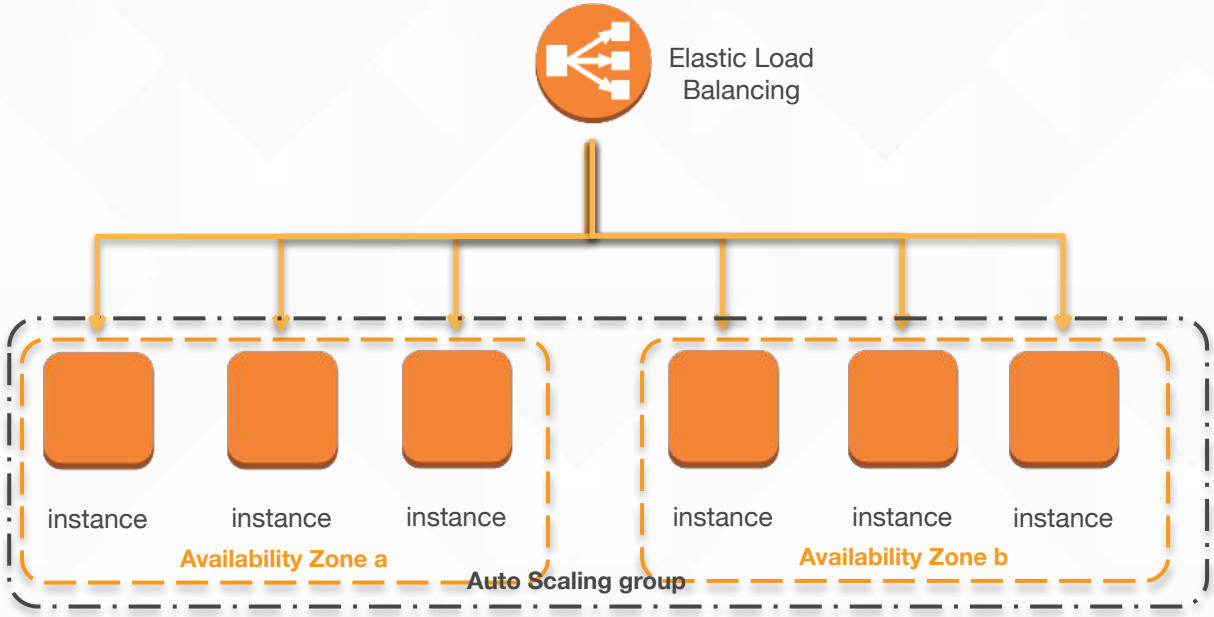
  (if new number of instances < minimum or < desired capacity)

amazon web services

Health Checks

- EC2 instance status:
  - Instance is unhealthy when instance state `!=` ‘`running`’ **or** system health check `==` ‘`impaired`’
- ELB health checks:
  - instance is unhealthy when ELB health check results in “`OutOfService`” (**or** EC2 health check failed)
- Manual: mark individual instances as ‘unhealthy’
  - Instance unhealthy when marked as such **or** EC2 health check failed. Use to integrate with external monitoring systems.

amazon web services



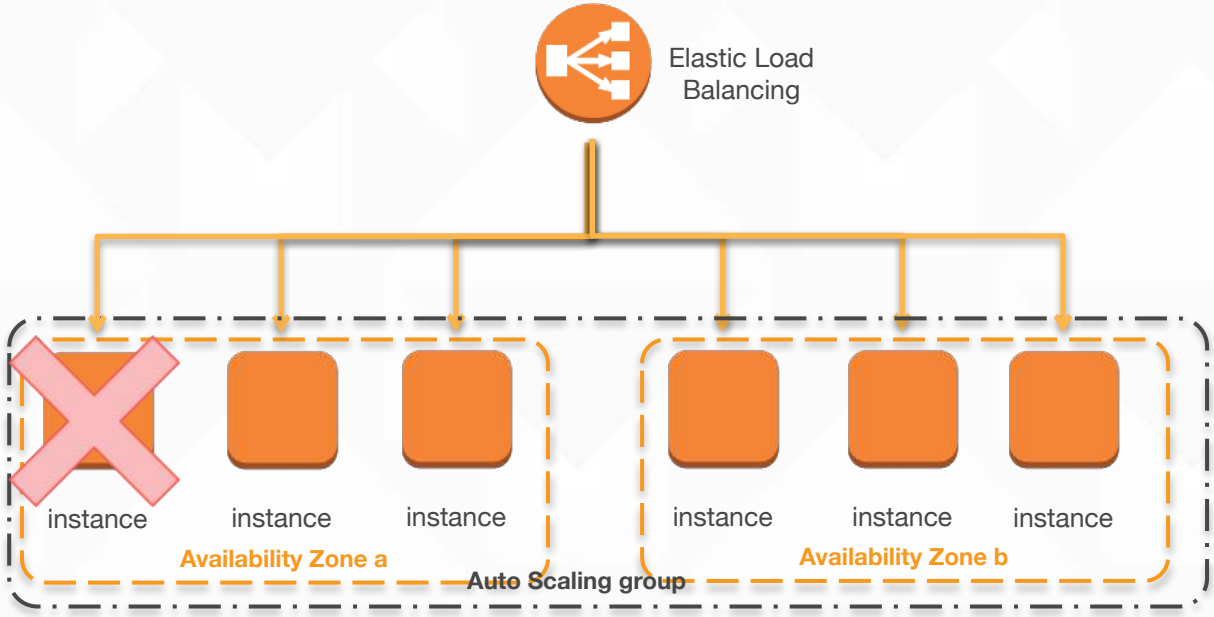
Minimum = 2

Maximum = 10

Desired # of instances = 6

amazon web services

Unhealthy Instances Get Replaced…



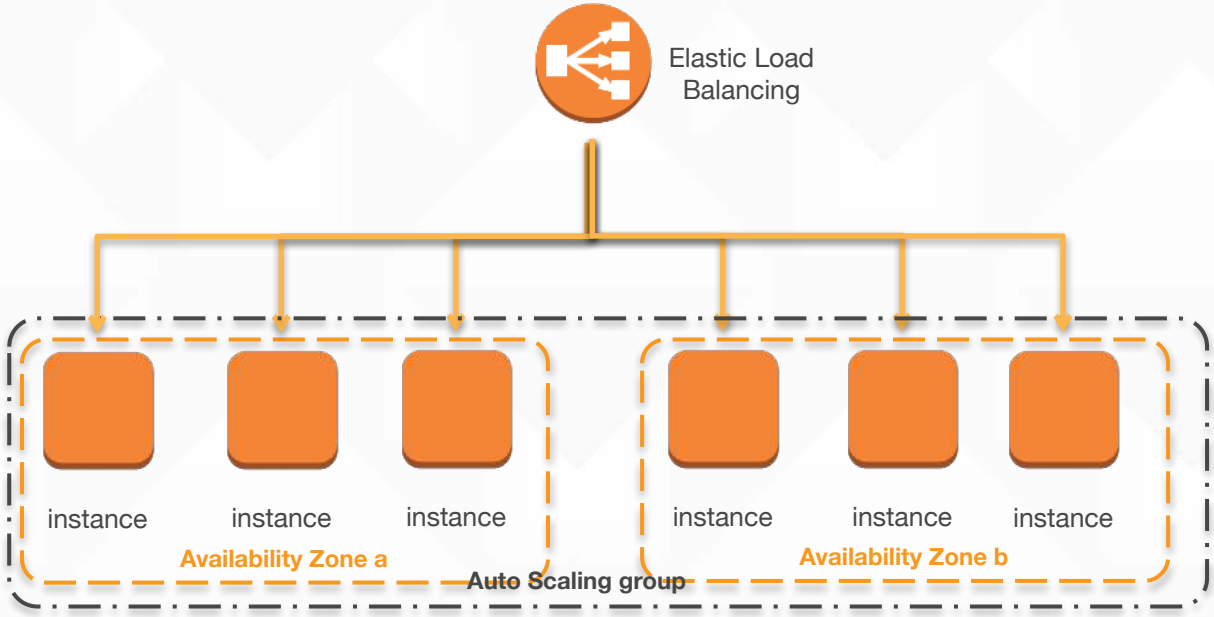
Minimum = 2

Maximum = 10

Desired # of instances = 6

amazon web services

Unhealthy Instances Get Replaced…



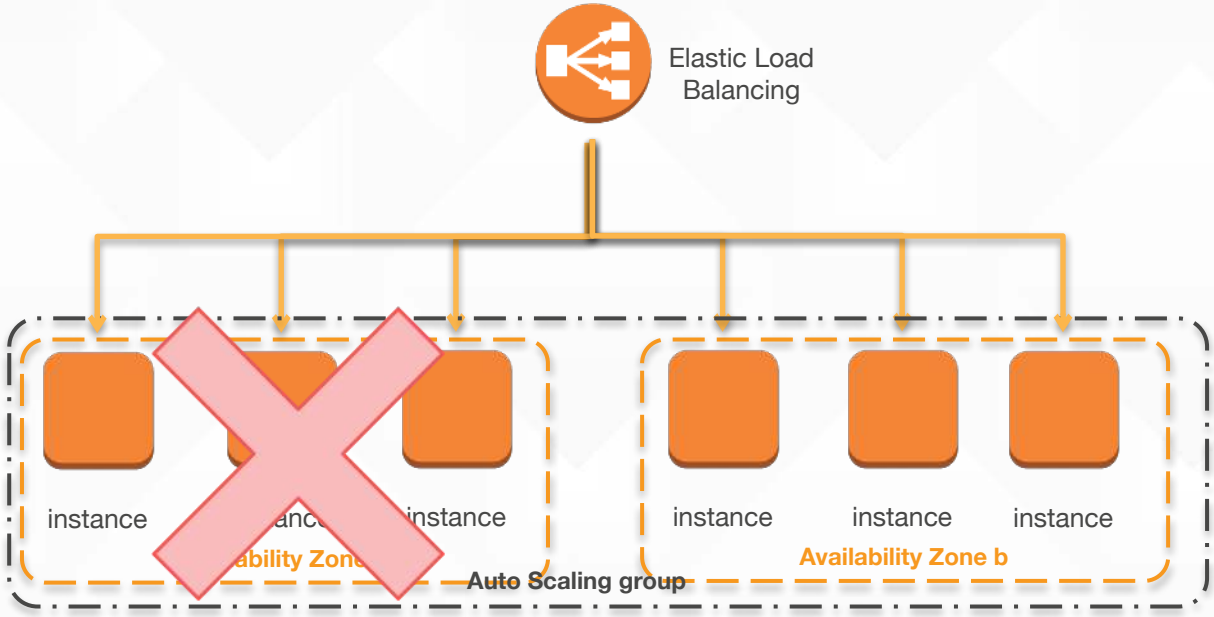
Minimum = 2

Maximum = 10

Desired # of instances = 6

amazon web services

Unhealthy Instances Get Replaced…



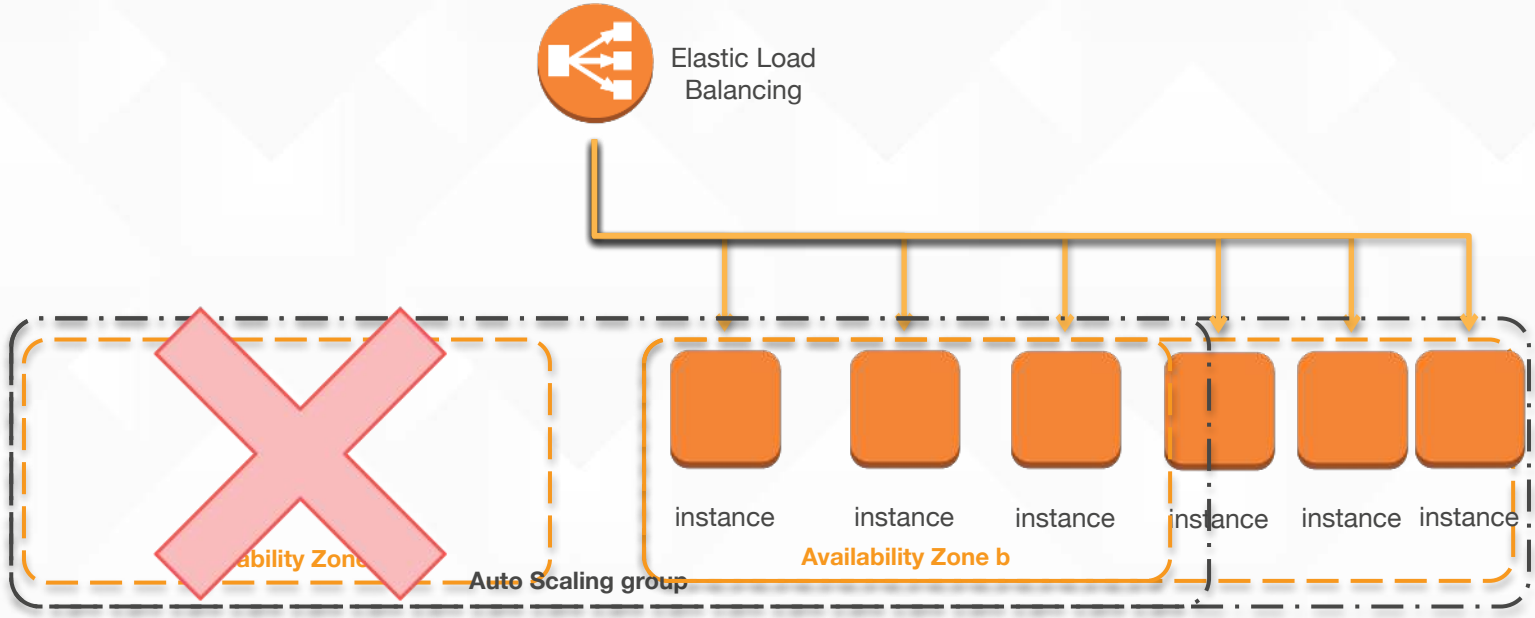
Minimum = 2

Maximum = 10

Desired # of instances = 6

amazon web services

…In a Different AZ if Necessary



Minimum = 2

Maximum = 10

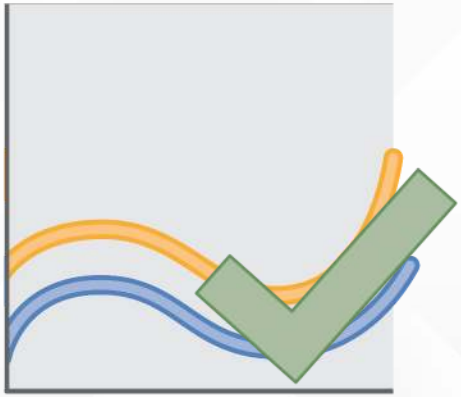
Desired # of instances = 6

amazon web services

Benefits of Auto Scaling

Elastic:

Automatically adapt capacity to demand



Reliable:

Counteract failures of instances or AZs

amazon web services

Scaling Policies

amazon web services

Scaling Policies

Can change capacity of the group in 3 different ways:

- Set fixed capacity, e.g., desired capacity = 4 instances
- Add / remove fixed number of instances, e.g., + 2
- Add / remove percentage of existing capacity, e.g. +20%

Can be either manual, scheduled or dynamic

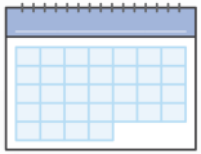
amazon web services

Scheduled Policies

cron-like syntax for recurring scaling events

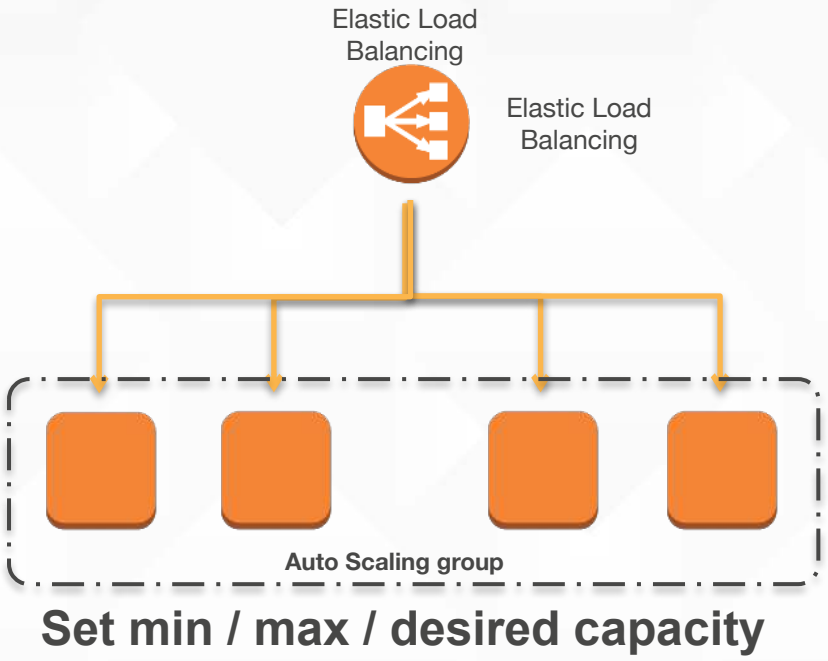
Scaling event

Schedule individual events



Scaling event

(up to 125 events per month per group)



amazon web services

Dynamic Scaling Policies

Trigger scaling events based on demand:

- Demand is measured based on metrics
- Changes in metrics can be mapped to scaling policies

amazon web services

Amazon CloudWatch

Amazon **CloudWatch**: A web service that enables you to monitor and manage various metrics, and configure alarm actions based on data from those metrics.

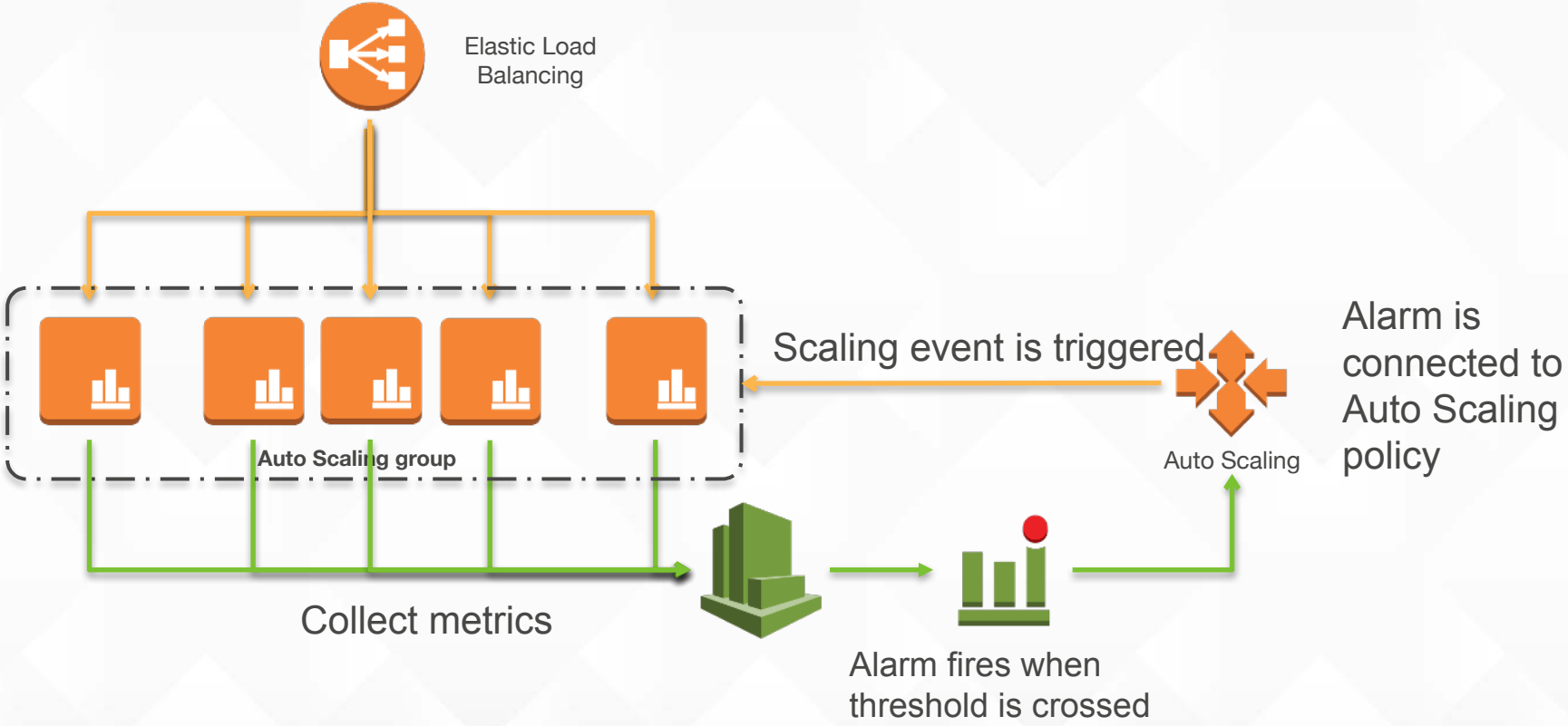
A **metric** is a variable that you want to monitor, e.g., CPU usage or incoming network traffic.

A **CloudWatch *alarm*** is an object that monitors a single metric over a specific period.

The alarm changes its **state** when the value of the metric breaches a defined range and maintains the change for a specified number of periods.

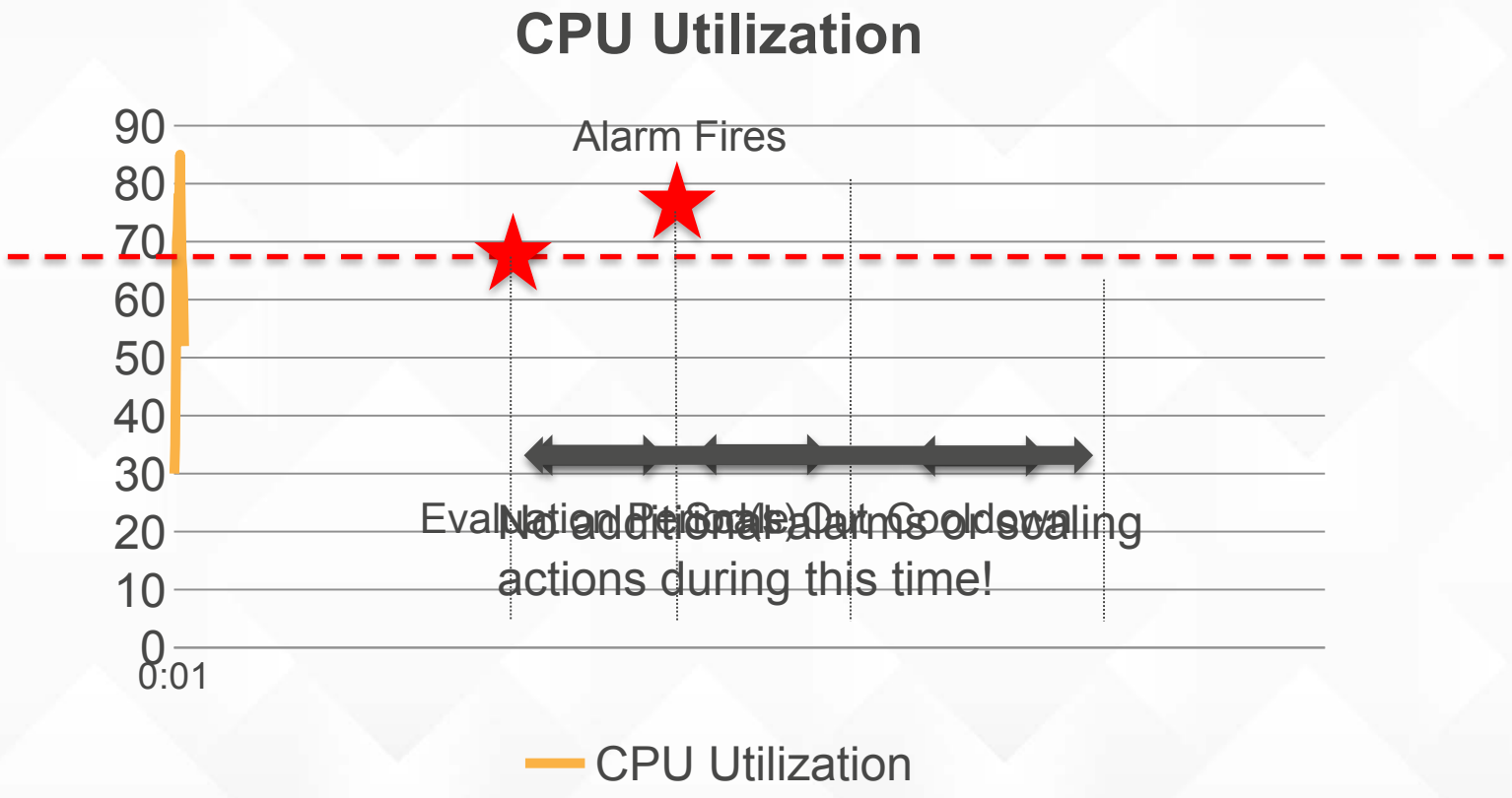
amazon web services

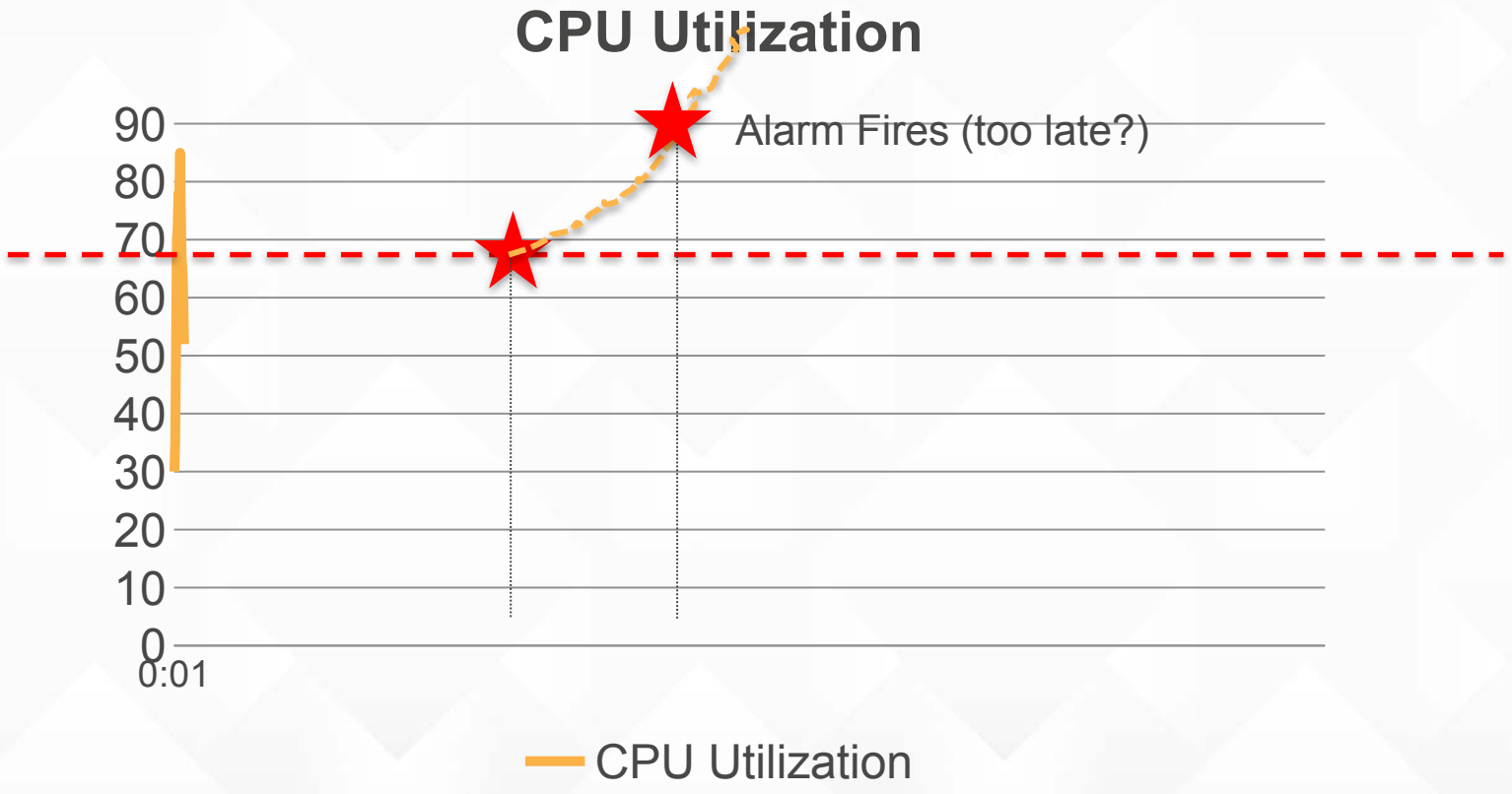
How Alarms Work



Amazon CloudWatch can aggregate metrics across pre-defined dimensions, e.g., aggregate average CPU utilization of all EC2 instances in an Auto Scaling group

amazon web services


CPU Utilization
90
80
70
60
50
40
30
20
10
0
0:01
Alarm Fires
Evaluation Period
Scale Out Cooldown
No additional alarms or scaling actions during this time!
CPU Utilization



amazon web services

Step Scaling Policies

Define **multiple steps** in the same policy.

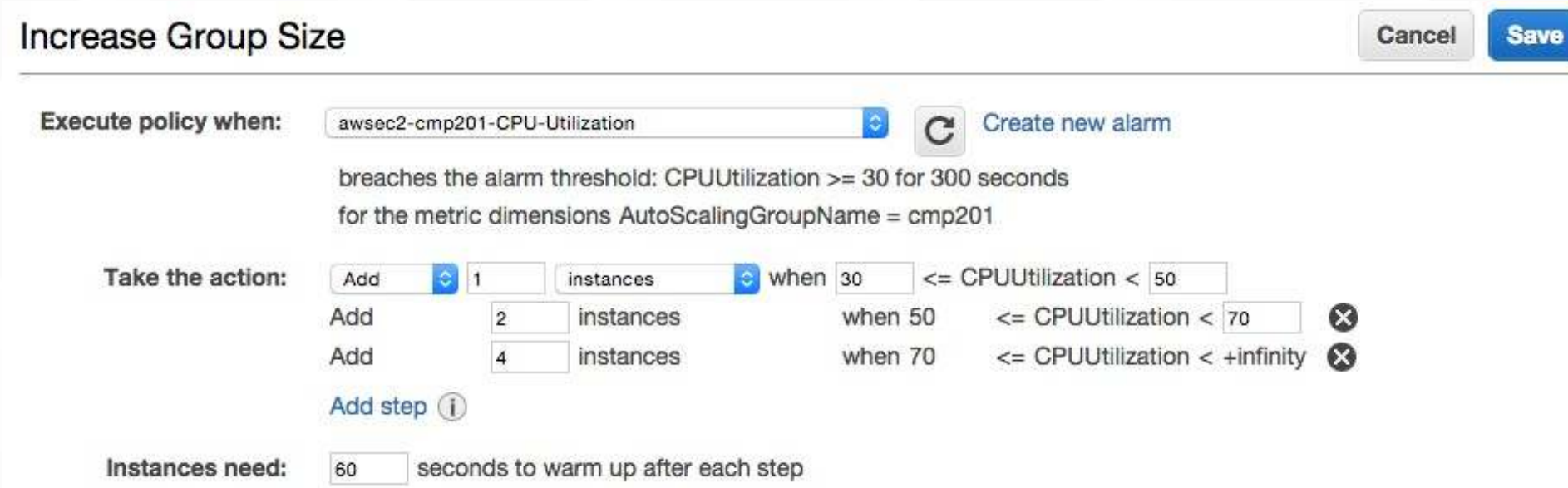
The appropriate scaling step is selected based on the value of the **metric** that triggered the alarm, i.e., based on the magnitude of the breach.

Step scaling policies are **evaluated continuously**.

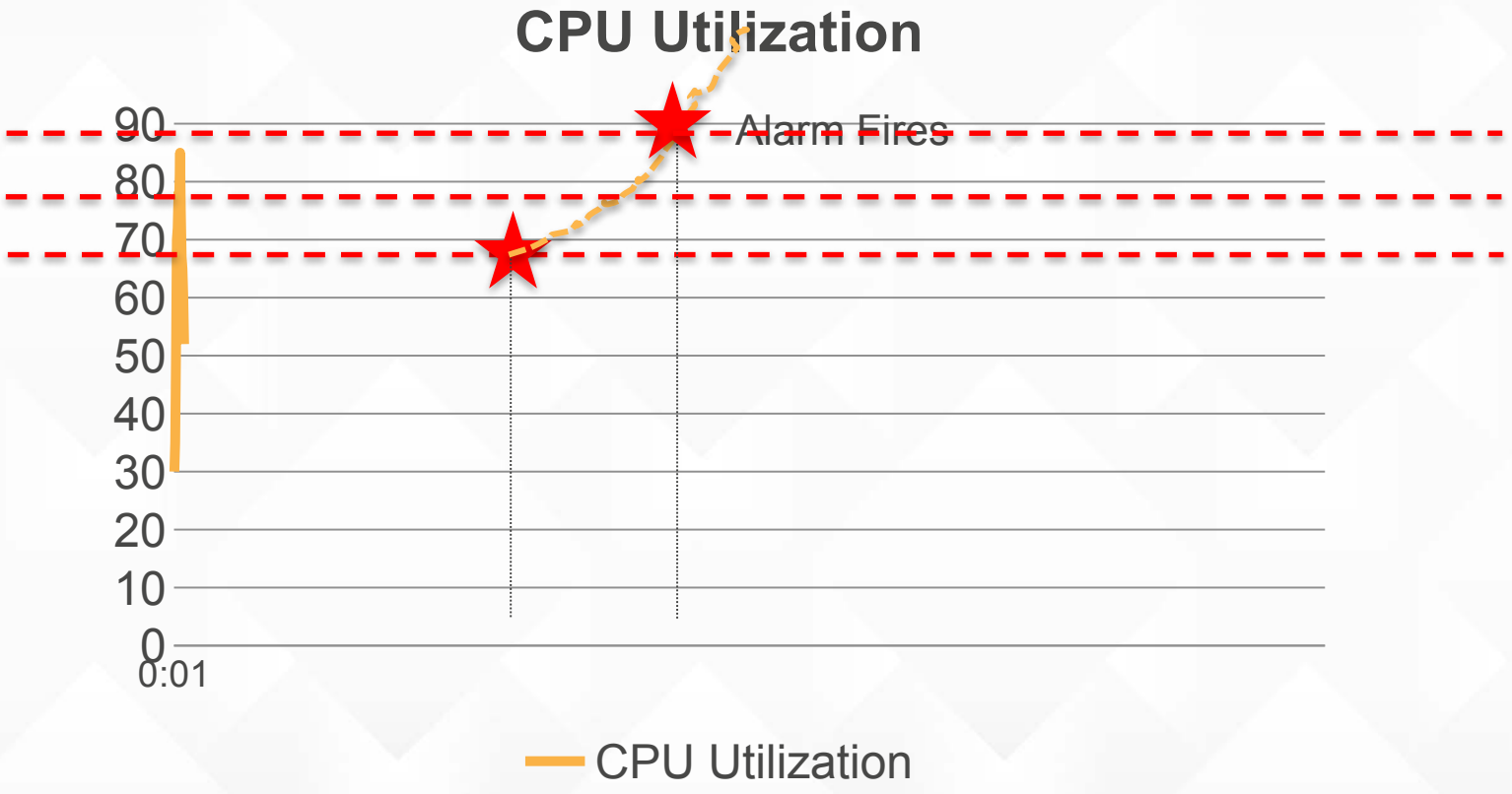
No locking – alarms keep getting evaluated even while scaling.

amazon web services

Step Scaling Policies



amazon web services



amazon web services

Pros & Cons

Simple scaling policies

- Reaction always the same, independent of size of breach
- Locks the Auto Scaling group when a scaling action is run, i.e., single event at a time
- Evaluates metric only when no scaling action ongoing
- Cooldown takes effect after scaling action is complete

Step scaling policies

- Continuously evaluates metrics
- “Instant” reaction
- Does not ‘lock’ the Auto Scaling group while action is evaluated
- Cooldown not supported (use ‘warmup’ period instead)

amazon web services

Lifecycle hooks

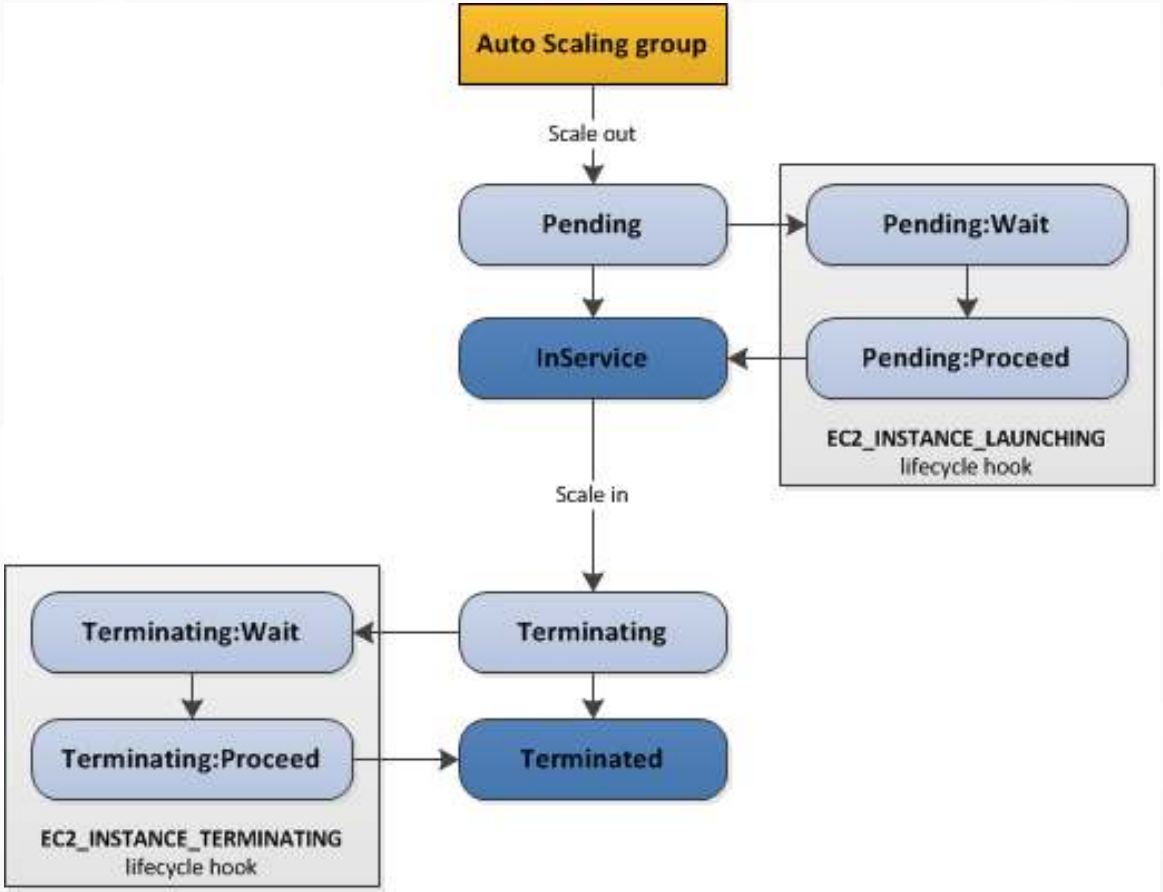
amazon web services

Why? – Common Use Cases

- Assign Elastic IP address on launch
- Register new instances with DNS, message queues,…
- Pull down log files before instance is terminated
- Investigate issues with an instance before terminating it
- …
- Scaling containers on ECS cluster

amazon web services

Instance lifecycle



amazon web services

Lifecycle hook vs notification

Lifecycle hooks

- Executed *before* taking a new instance into service / terminating it
- Put instances into a WAIT state

Auto Scaling SNS notifications

- Notifications get sent *after* an instance has entered “InService” or “Terminated” state, respectively

amazon web services

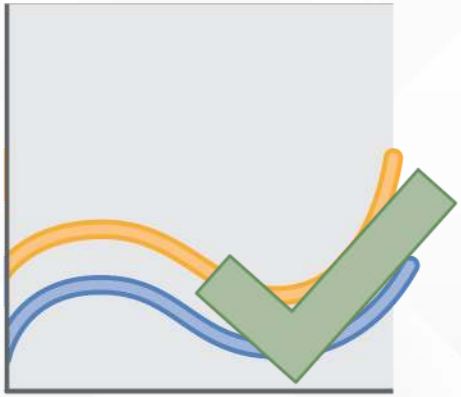
Lifecycle hooks with SQS / SNS

1. Create a notification target– either an Amazon **SQS queue,** an Amazon **SNS topic**
2. Create an **IAM role** that allows Auto Scaling to access the notification queue / topic.
3. Associate the lifecycle hook with the Auto Scaling group, role and notification target.
4. Code the lifecycle hook's action

amazon web services

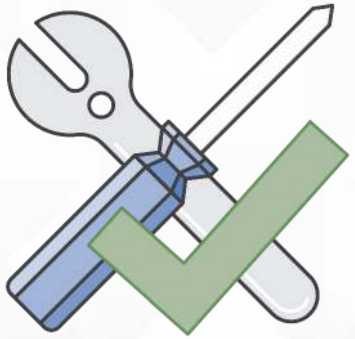
Benefits of Auto Scaling

Elastic:
Automatically adapt capacity to demand



Reliable:
Counteract failures of instances or AZs

Customizable:
With bootstrapping & lifecycle hooks



amazon web services

DEMO

Demo gods, I'm your humble servant,
please be good to me

Let's look at a new way to implement hooks,
launched in January 2016 : **CloudWatch Events**

Write a Lambda function printing lifecycle events
Create a CloudWatch event rule
Connect it to the Lamba function
Trigger Auto Scaling activity
Fetch events from CloudWatch Logs

…and learn some CLI on the way ;)

amazon web services

Lifecycle hooks with CloudWatch Events (1)

Write a Lambda function

```
console.log('Loading function');

exports.handler = function(event, context {
    console.log("AutoScalingEvent");
    console.log("Event data:\n" + JSON.stringify(event, null, 4));
    context.succeed("...");
};
```

amazon web services

Lifecycle hooks with CloudWatch Events (2)

In CloudWatch Events, create a rule to route auto scaling events to your Lambda function

- Select event types
- Select auto scaling groups

Events will be logged to the CloudWatch log group of your Lambda function (created automatically when the function is invoked)

amazon web services

Event rule

```
{
  "source": [
    "aws.autoscaling"
  ],
  "detail-type": [
    "EC2 Instance Launch Successful",
    "EC2 Instance Terminate Successful",
    "EC2 Instance Launch Unsuccessful",
    "EC2 Instance Terminate Unsuccessful",
    "EC2 Instance-launch Lifecycle Action",
    "EC2 Instance-terminate Lifecycle Action"
  ]
}
```

amazon web services

Putting it all together

```
% aws lambda get-function --function-name autoscalingFunction
  --query "Configuration.FunctionArn”
% aws events put-rule --name autoscalingRule --event-pattern file://autoscalingRule.json
  --state ENABLED
% aws events put-targets --rule autoscalingRule --targets Id=1,Arn=FUNCTION_ARN
% aws lambda add-permission --function-name autoscalingFunction --statement-id 1
  --action 'lambda:InvokeFunction' --principal events.amazonaws.com --source-arn RULE_ARN

<Start some Auto Scaling activity>

% aws logs describe-log-streams --log-group-name /aws/lambda/autoscalingFunction
  --query "logStreams[].logStreamName”
% aws logs get-log-events --log-stream-name LOG_STREAM_NAME
  --log-group-name /aws/lambda/autoscalingFunction
```

amazon web services

Thank You

Julien Simon
Principal Technical Evangelist, AWS
julsimon@amazon.fr
@julsimon

amazon web services

Pop-up Loft
TEL AVIV