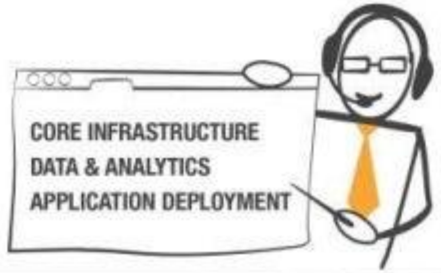
AWS APAC WEBINAR WEEK



Autoscaling on AWS

Dhruv Parpia
Solutions Architect, ASEAN

amazon web services

Topics We’ll Cover

- Amazon EC2
- Types of Scaling
- Auto Scaling Introduction
- Auto Scaling Terminology
- Best Practices
- Console Demo

Amazon Elastic Compute Cloud (EC2)

Amazon EC2

Basic unit of compute capacity

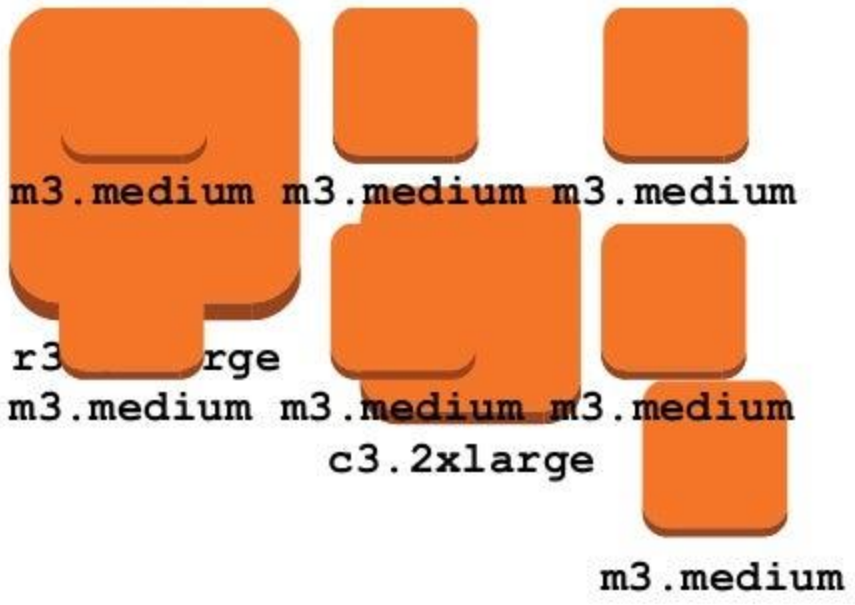
Range of CPU, memory & local disk options

Feature	Details
Flexible	Run windows or linux distributions
Scalable	Wide range of instance types from micro to cluster compute
Machine Images	Configurations can be saved as machine images (AMIs) from which new instances can be created
Full control	Full root or administrator rights
Secure	Full firewall control via Security Groups
Monitoring	Publishes metrics to Cloud Watch
Inexpensive	On-demand, Reserved and Spot instance types
VM Import/Export	Import and export VM images to transfer configurations in and out of EC2

amazon webservices

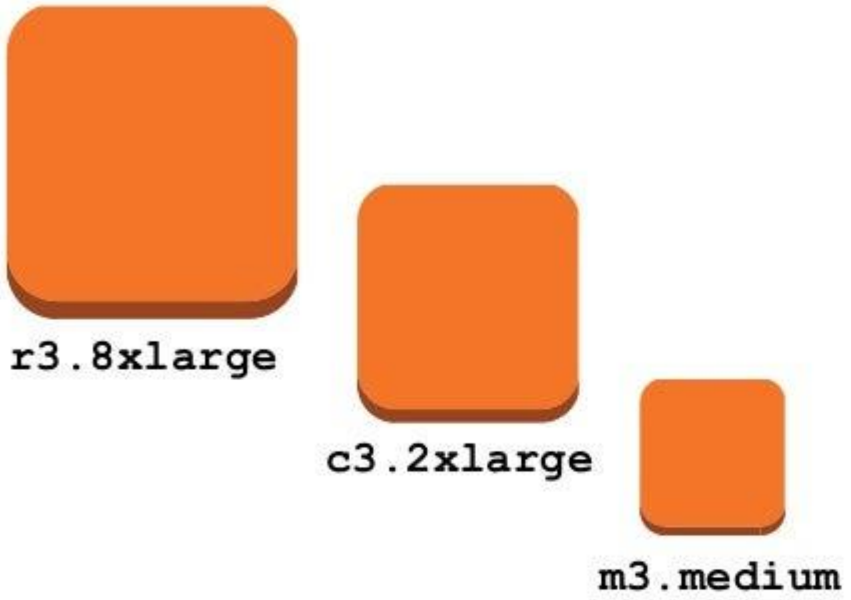
Types of Scaling

- Vertical Scaling
  - Changing instance size
  - Increasing EBS Capacity
- Horizontal Scaling
  - Adding / removing instances
  - ELB



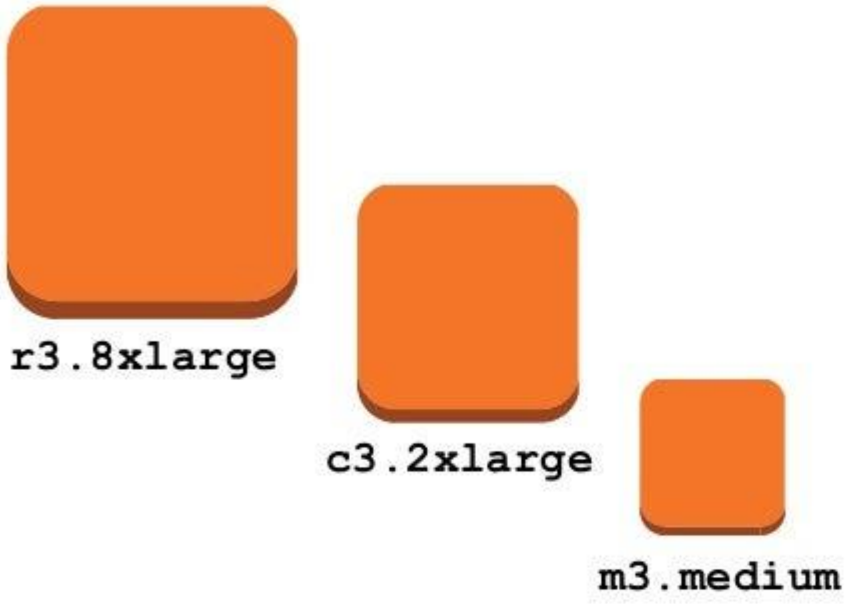
“We’re gonna need a bigger box”

- Different EC2 instance type
  - High memory instances
  - High CPU instances
  - High I/O instances
  - High storage instances
- Can now leverage PIOPs
- Easy to change instance sizes
- Will hit an endpoint eventually

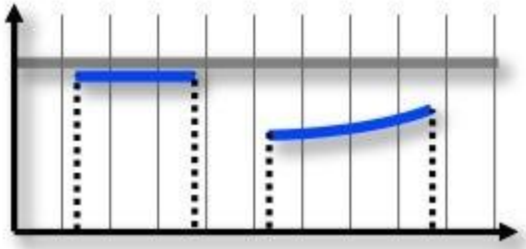


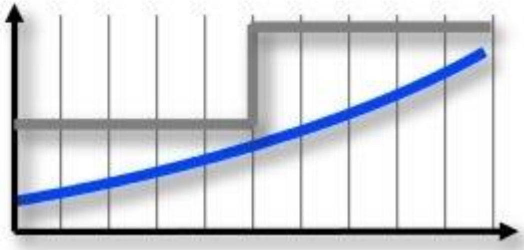
“We’re gonna need a bigger box”

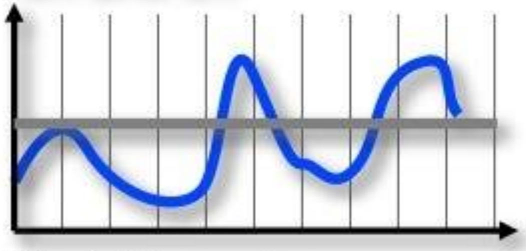
- Different EC2 instance type
  - High memory instances
  - High CPU instances
  - High I/O instances
  - High storage instances
- Can now leverage PIOPs
- Easy to change instance sizes
- Will hit an endpoint eventually

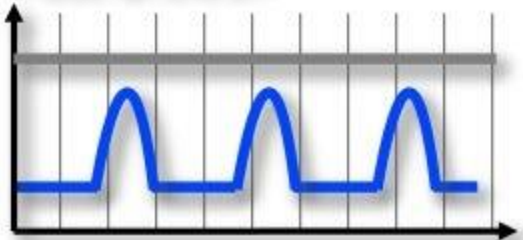


Traditional IT Usage Patterns

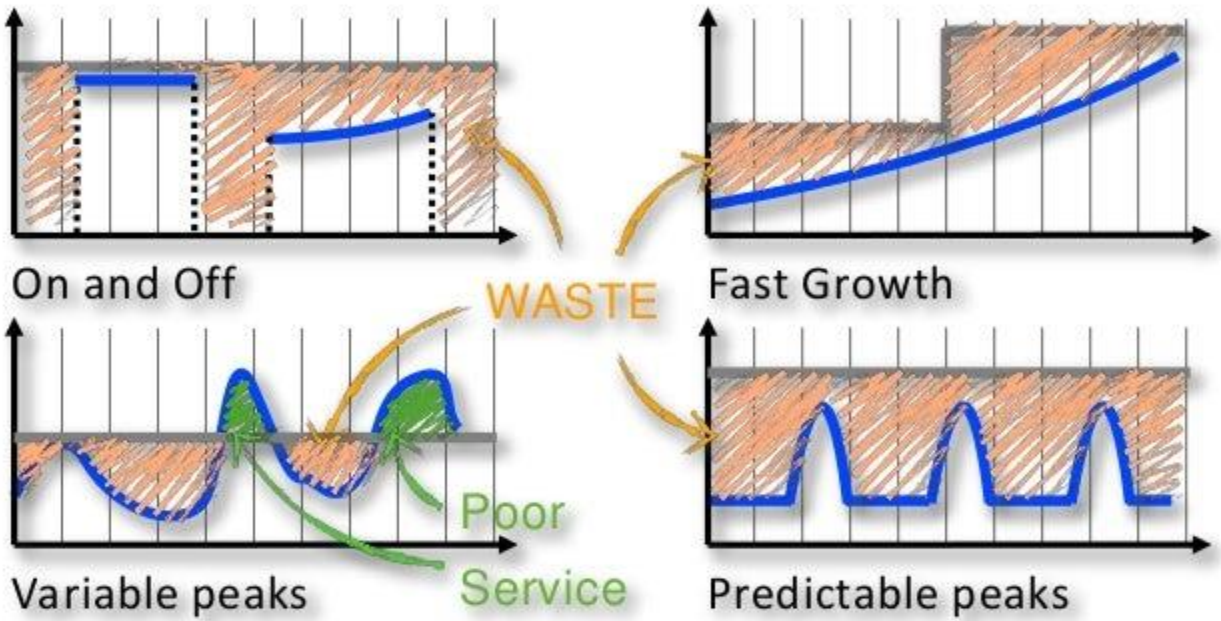

On and Off


Fast Growth


Variable peaks

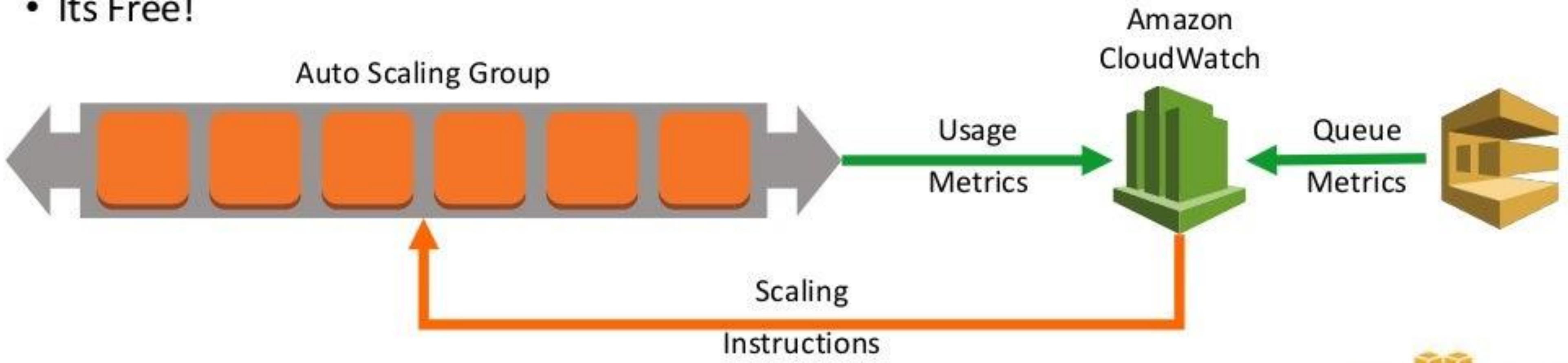

Predictable peaks

Traditional IT Usage Patterns

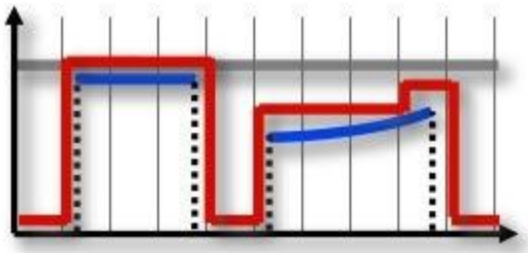


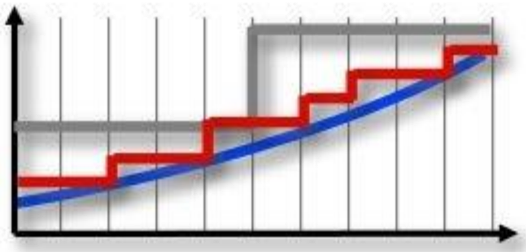
Auto Scaling

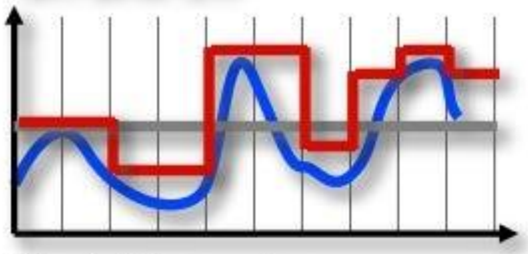
- Automatic resizing of compute clusters based on demand
- Define minimum and maximum number of instances
- Define when scaling out and in occurs
- Use metrics collected in Amazon CloudWatch to drive scaling
- Run Auto Scaling for On-Demand and Spot instance types
- Its Free!

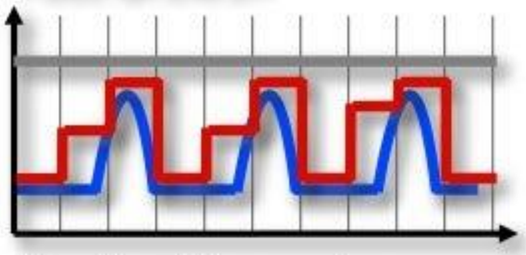


Cloud IT Usage Patterns (Auto Scaling)


On and Off


Fast Growth


Variable peaks


Predictable peaks

Terminology for Auto Scaling

- Auto Scaling Group
- Launch Configuration
- Scaling Plan
- Amazon CloudWatch Alarm
- Amazon SNS Notification
- Elastic Load Balancer
- Instance Lifecycle
- Instance Metadata / Userdata

Auto Scaling and AWS

Auto Scaling Groups

- EC2 instances are categorized into 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.
- Create the launch configuration by providing information about the image you want Auto Scaling to use to launch EC2 instances

Scaling Plan

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

CloudWatch

Retrieving Resource Metrics

- CloudWatch: A web service that enables you to monitor and manage various metrics, and configure alarm actions based on data from those metrics.
- A CloudWatch *alarm* is an object that monitors a single metric over a specific period.
- A metric is a variable that you want to monitor. *eg: CPU usage, or incoming network traffic*.
- 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.

Planning your Auto Scaling Group

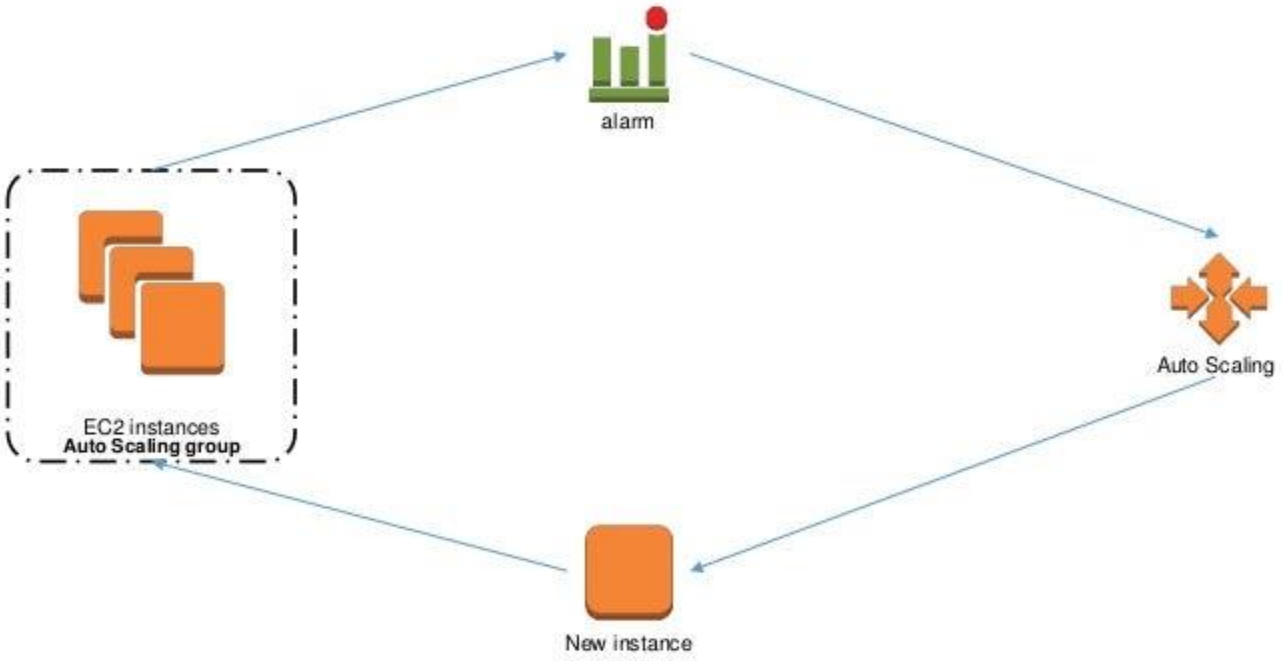
- How long it takes to launch and configure a server
- What metrics have the most relevance to your application's performance
- What existing resources (such as EC2 instances or AMIs) you might want to use as part of your Auto Scaling group
- How many Availability Zones you want to the Auto Scaling group to span
- The role you want Auto Scaling to play in your application.

Load Balance your Auto Scaling Group

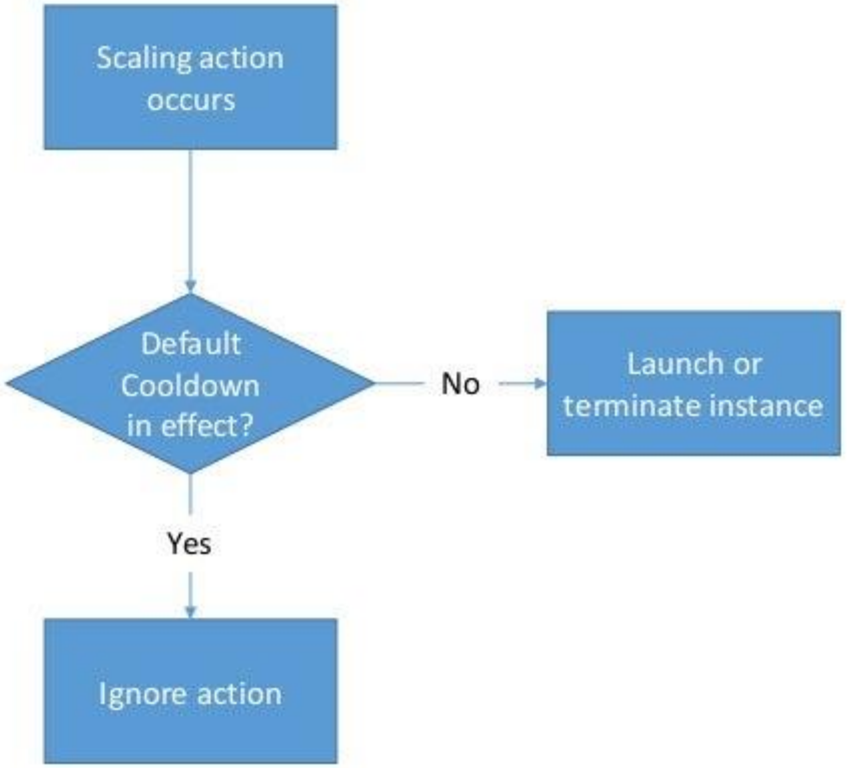
Elastic Load Balancing

- 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.

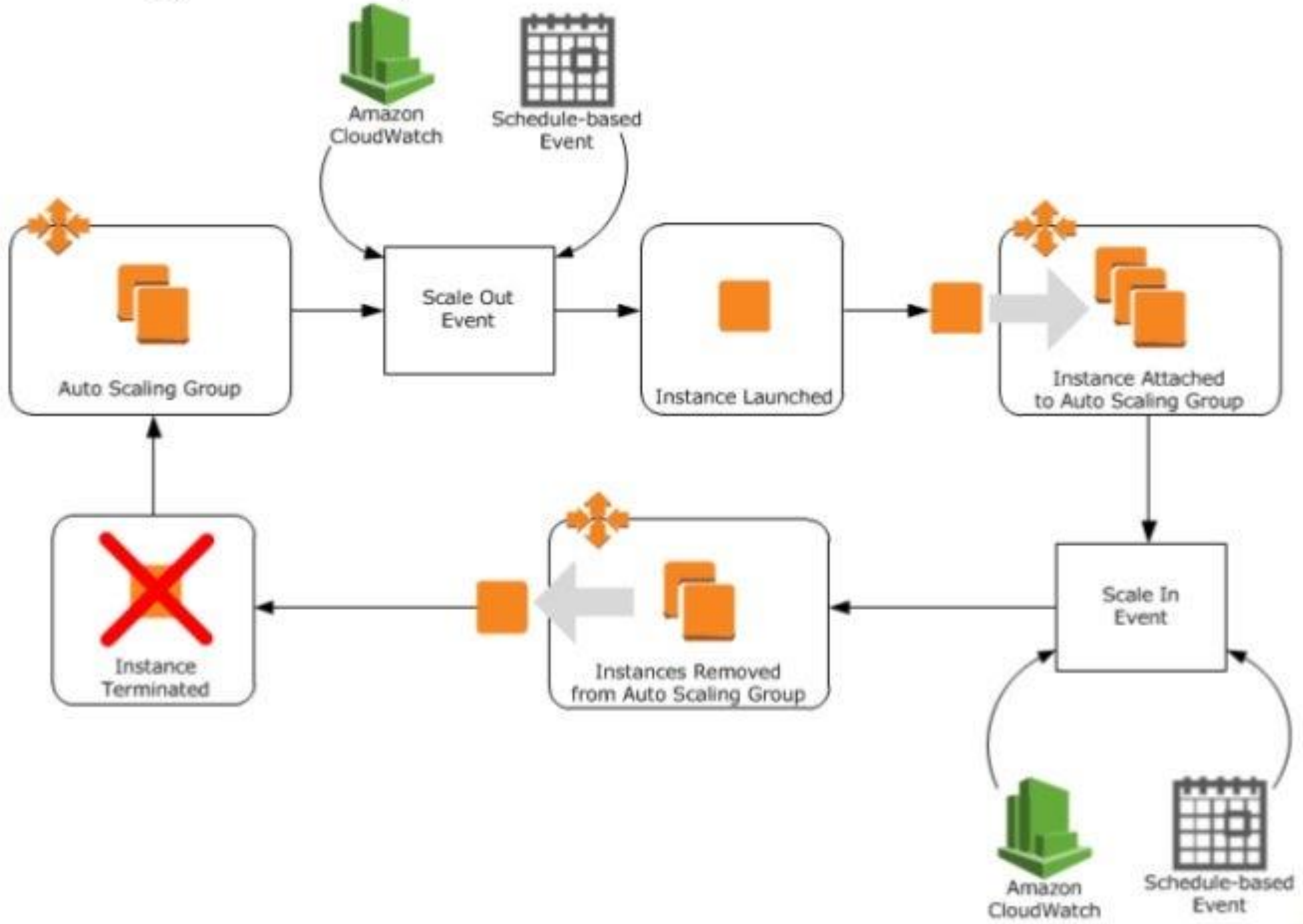
Understanding Auto Scaling Cooldowns



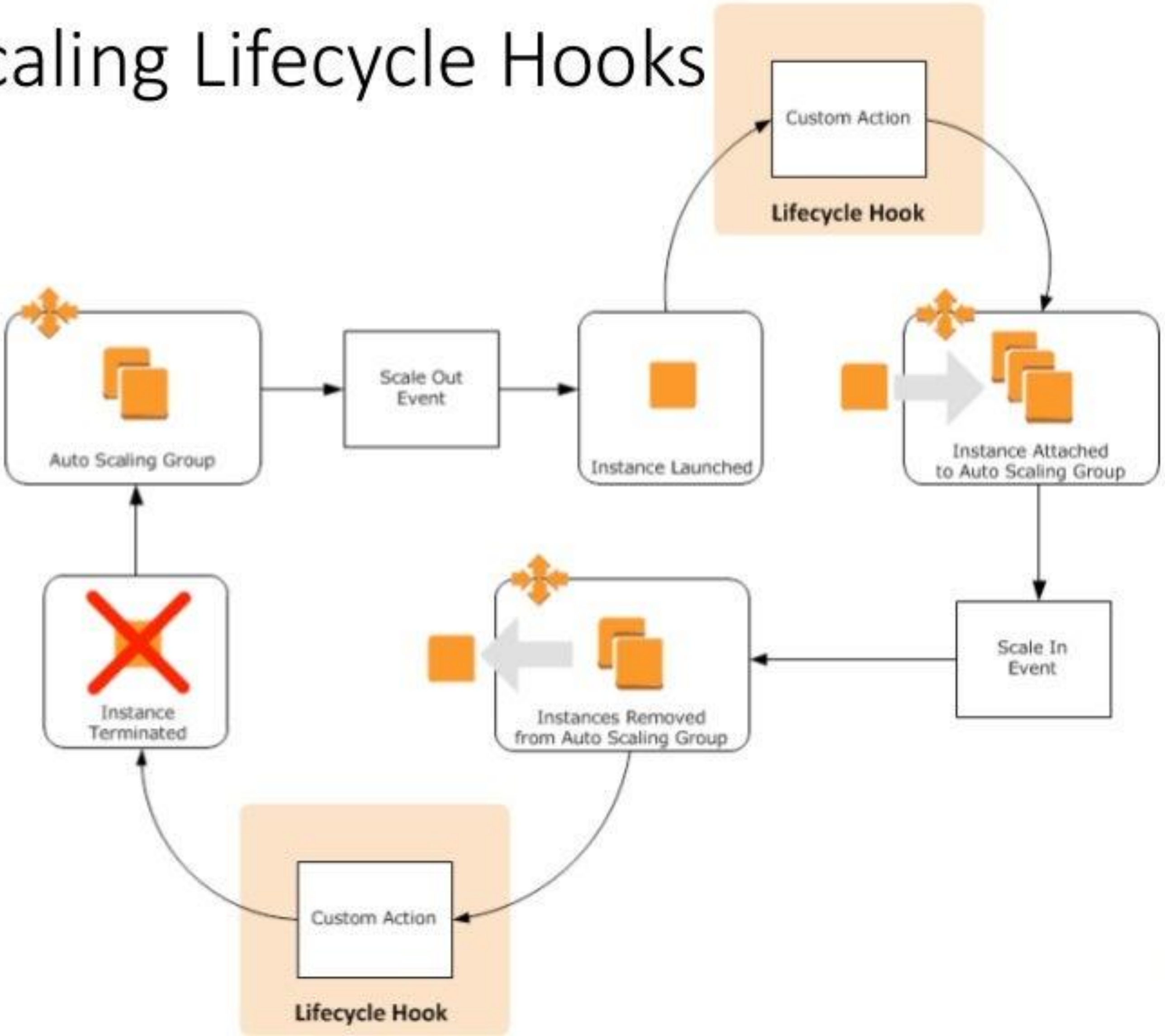
Understanding Auto Scaling Cooldowns



Auto Scaling Lifecycles



Auto Scaling Lifecycle Hooks



Introducing Bootstrapping

Bootstrapping: some examples

- Install latest software
- Copy data from S3
- Register with DNS
- Start services
- Update packages
- Reboot
- Open port 80
- Register with load balancer
- Mount devices

Bootstrapping tools

- **Scripts** on instance (Bash, Powershell)
- **Config Management Tools** (Chef, Puppet)
- **Amazon OpsWorks**

EC2 Metadata and UserData

- Every **EC2 Instance** has access to local instance **metadata** and **userdata service**

- **Metadata**: immutable information about the instance

Accessible from within the instance via HTTP at

- **http://169.254.169.254/latest/meta-data/**

EC2 Metadata and UserData

EC2 Metadata and UserData

Script(s) on instance may retrieve useful information about the instance, such as:

- **Host name**
- **AMI ID**
- **Instance ID**
- **Public/Private DNS**
- **Availability Zone**

EC2 Metadata and UserData

EC2 Metadata and **UserData**

- **Pass up to 16KB of text to an instance on launch**
- Text can be parsed by script on instance and used to configure the machine

UserData and CloudInit

- CloudInit executes UserData on first boot if UserData begins with:
  - **#! (Linux)**
  - **<script> (Windows; technically, EC2Config, not CloudInit, does this)**

UserData and CloudInit

- CloudInit executes UserData on first boot if UserData begins with:
  - **#! (Linux)**
  - **<script> (Windows; technically, EC2Config, not CloudInit, does this)**
- CloudInit is installed on Amazon Linux, Ubuntu, and RHEL AMIs
- EC2Config is installed on Windows Server AMIs
- Both may be installed on other distributions via a package repo or source

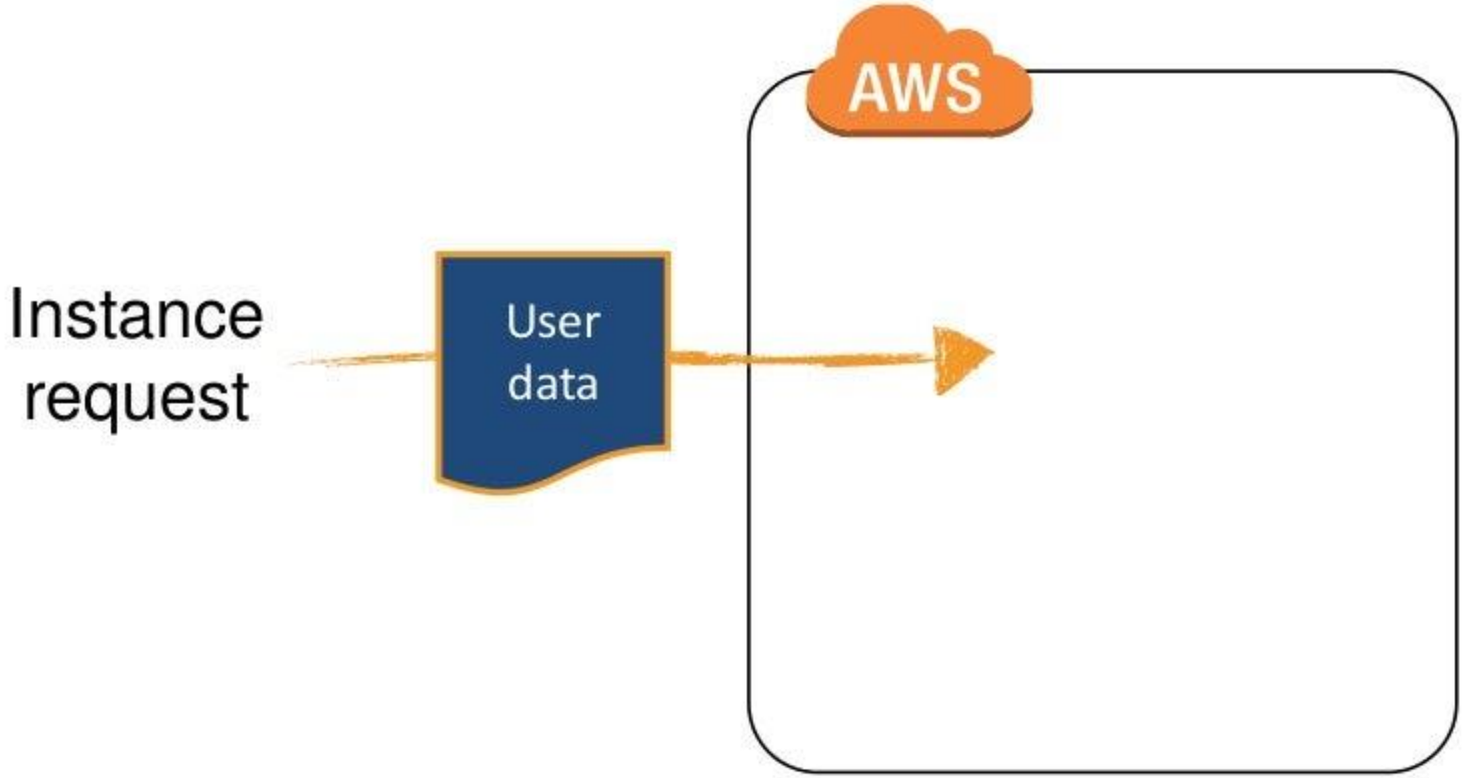
UserData and CloudInit

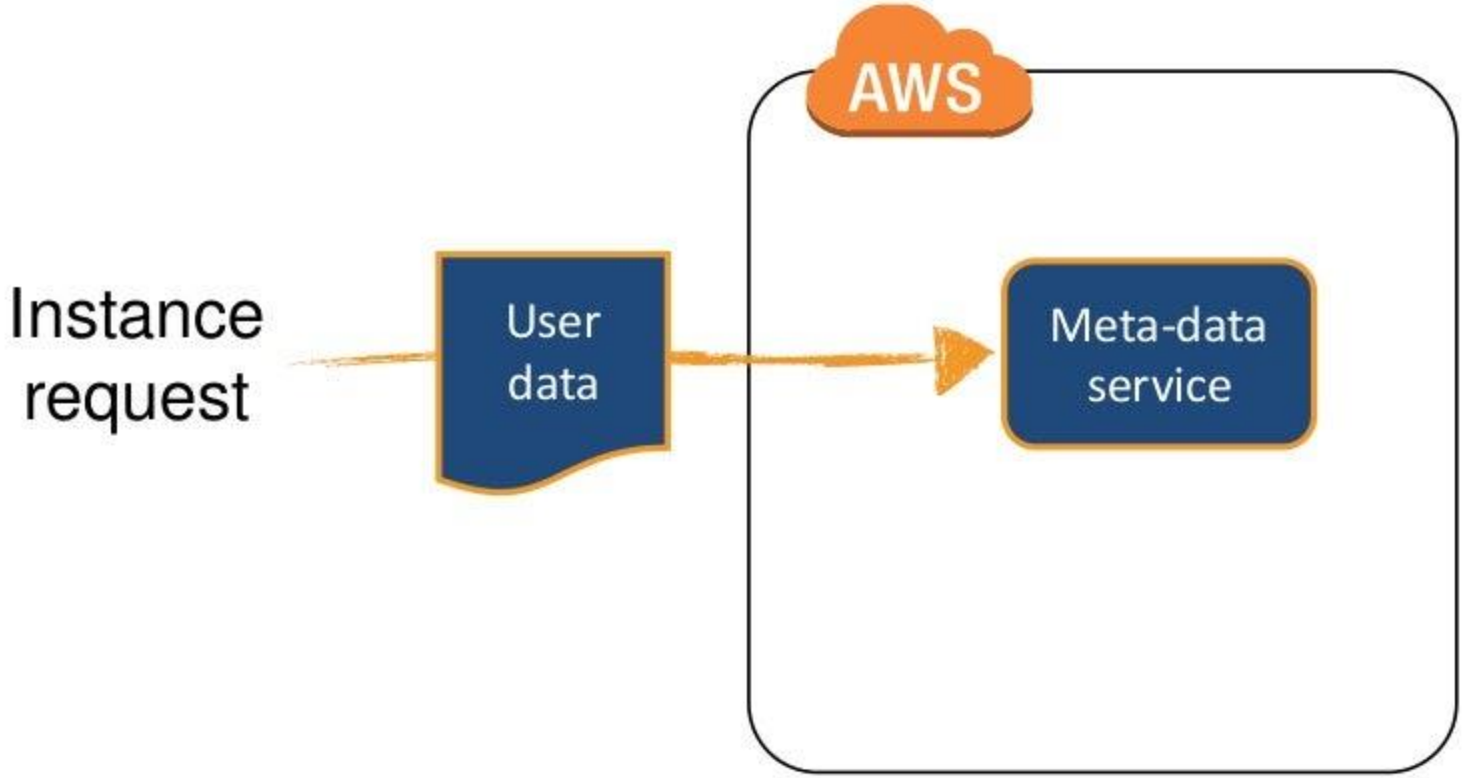
- UserData to **install Apache** and **MySQL on boot,** and **attach an EIP:**

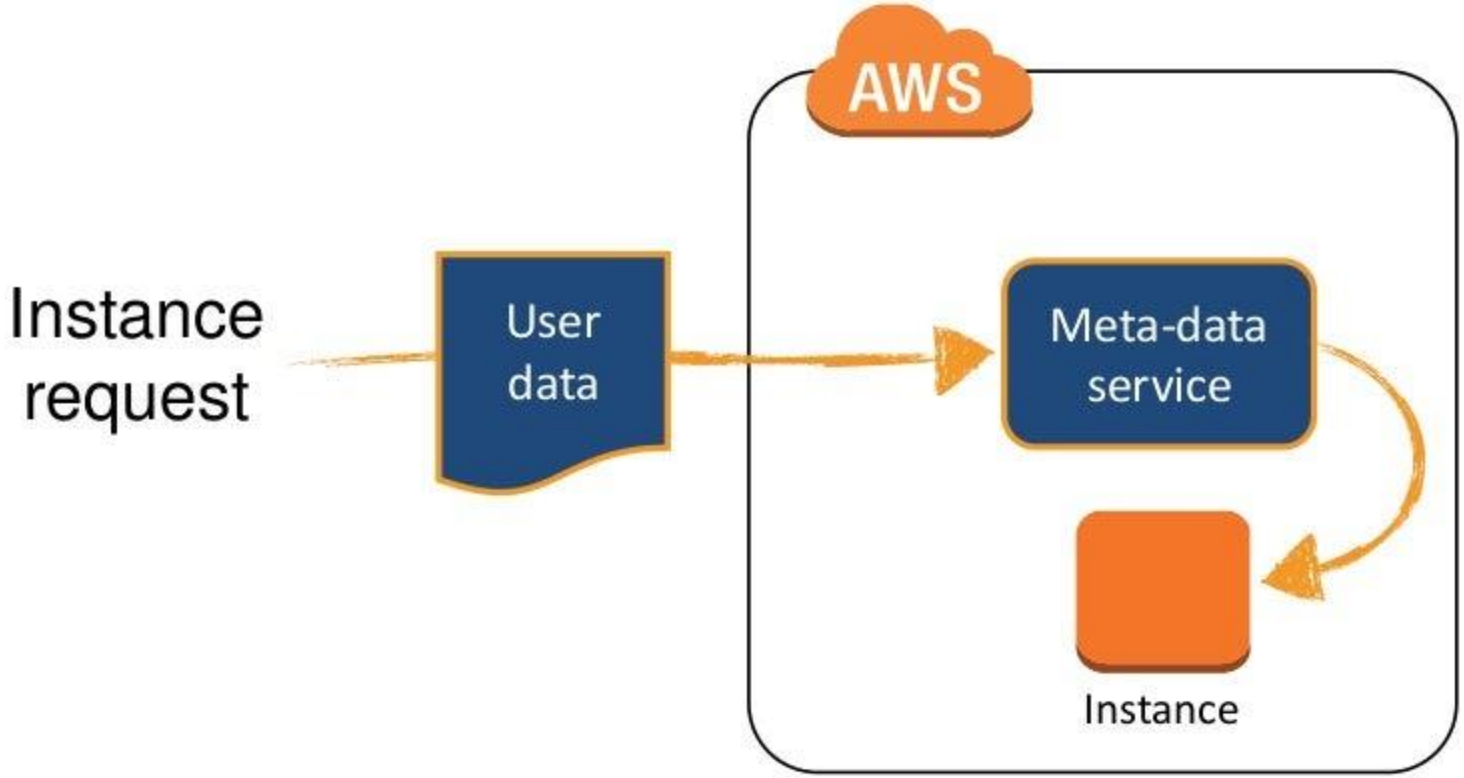
```
#!/bin/bash

# Install Apache, PHP, and MySQL
yum install -y httpd mysql-server

# Attach an Elastic IP to this instance
ec2-associate-address \
      23.34.45.56 \
      -i $(curl http://169.254.169.254/latest/meta-data/instance-id)
```


Instance request
User data
AWS


Instance request
User data
AWS
Meta-data service


Instance request
User data
AWS
Meta-data service
Instance

Amazon Windows EC2Config Service executes user-data on launch:

```
<script>dir > c:\test.log</script>

<powershell>any command that you can run</powershell>
```

AWS Powershell Tools (use IAM roles as before…)

```
<powershell>
     Read-S3Object -BucketName myS3Bucket
     -Key myFolder/myFile.zip
     -File c:\destinationFile.zip
</powershell>
```

Why do this?

Automation
Less fingers, less mistakes

Security
Instances locked down by default

Availability
Drive higher availability with self-healing

Flexible
Shell, Powershell, CloudFormation,Chef, Puppet, OpsWorks

Scale
Manage large scale deployments and drive autoscaling

Efficiency
Audit and manage your estate with less time & effort

Some Do’s and Don’ts

Do

Use IAM roles

Go keyless if you can

Strike a balance between AMI and dynamic bootstrapping

Some Do’s and Don’ts

Do

Use IAM roles

Go keyless if you can

Strike a balance between AMI and dynamic bootstrapping

Don’t

Put your API access keys into code (and then publish to GIT) or bake into AMIs (and share)

☹

Scaling Demo

Autoscaling isn't one size fits all

- Choose the right metrics
  - CPU Usage
  - Queue Depth
  - Number of concurrent users
- Scale too aggressively
  - Overprovisioning: increases costs
  - Bounciness: Add more than you need and have to partially scale back shortly after scaling up, increasing costs.
- Scale too timidly
  - Poor performance
  - Outages due to lack of capacity
- Scale out early / Scale in slowly

What’s new?

- Attach / Detach Instances from Auto Scaling Groups
- Place instances into Standby State to Troubleshoot
- Hold instances in Pending state for installing software / retrieve logs
- Create an Auto Scaling Group / Launch Configuration based on a running instance