

# AWS Intro

Build solid foundation of main AWS concepts and services



Azat Mardan @azat\_co



# Module 1: Cloud and AWS Basics

# Why cloud?

Yell out loud the benefits! 

# Overview of cloud computing

» IaaS

» PaaS

» FaaS

» SaaS

Main concepts:  
Regions and AZs

# Main AWS Services

- >> EC2
- >> S3
- >> Glacier
- >> EBS
- >> VPC
- >> CloudWatch
- >> Alerts

# More, much more other services

- >> AI
- >> Big Data
- >> Notifications
- >> Deploy, CI/CD
- >> Mobile, IoT

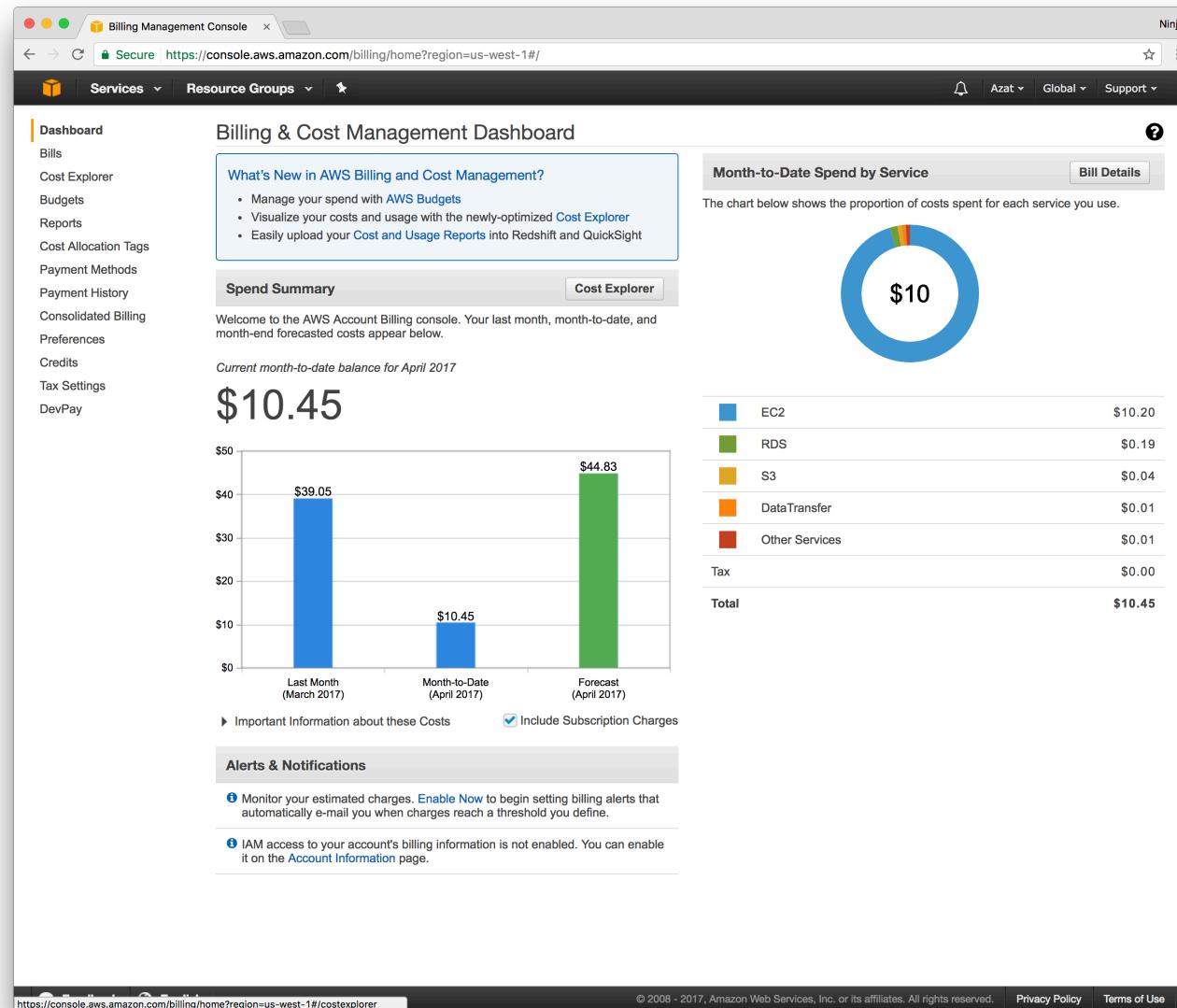
# Move Exabyte-Scale Data Sets with AWS Snowmobile



# Billing and calculator

- >> SIMPLE MONTHLY CALCULATOR
- >> Amazon EC2 Pricing
- >> Amazon S3 Pricing

# Billing Management console



# What you need or lab 0

# Slides&code

Repo: <https://github.com/azat-co/aws-intro>

Git clone (you can fork first too):

```
git clone https://github.com/azat-co/aws-intro.git
```

Download with CURL and unzip (create a new folder):

```
curl https://codeload.github.com/azat-co/aws-intro/zip/master | tar -xv
```

# AWS account: Free tier

- >> Sign up for free tier with an email you have access to
- >> Random verification (phone call or wait – ask me for access to my account)
- >> Debit/credit Card for verification and paid services

# AWS account: Free tier (cont)

Free tier: <https://aws.amazon.com/free>, examples:

- » EC2: 750 hours of t2.micro (~1 month of 1 EC2) – more than enough for this class and then some more
- » S3: 5Gb
- » RDS: 750 hours
- » Lambda: 1,000,000 requests/mo
- » More products!

# Node and npm

**node --version**

**npm --version**

I develop natively on my dev machine, but you can use another EC2 instance

? Questions? ?

# Checklist

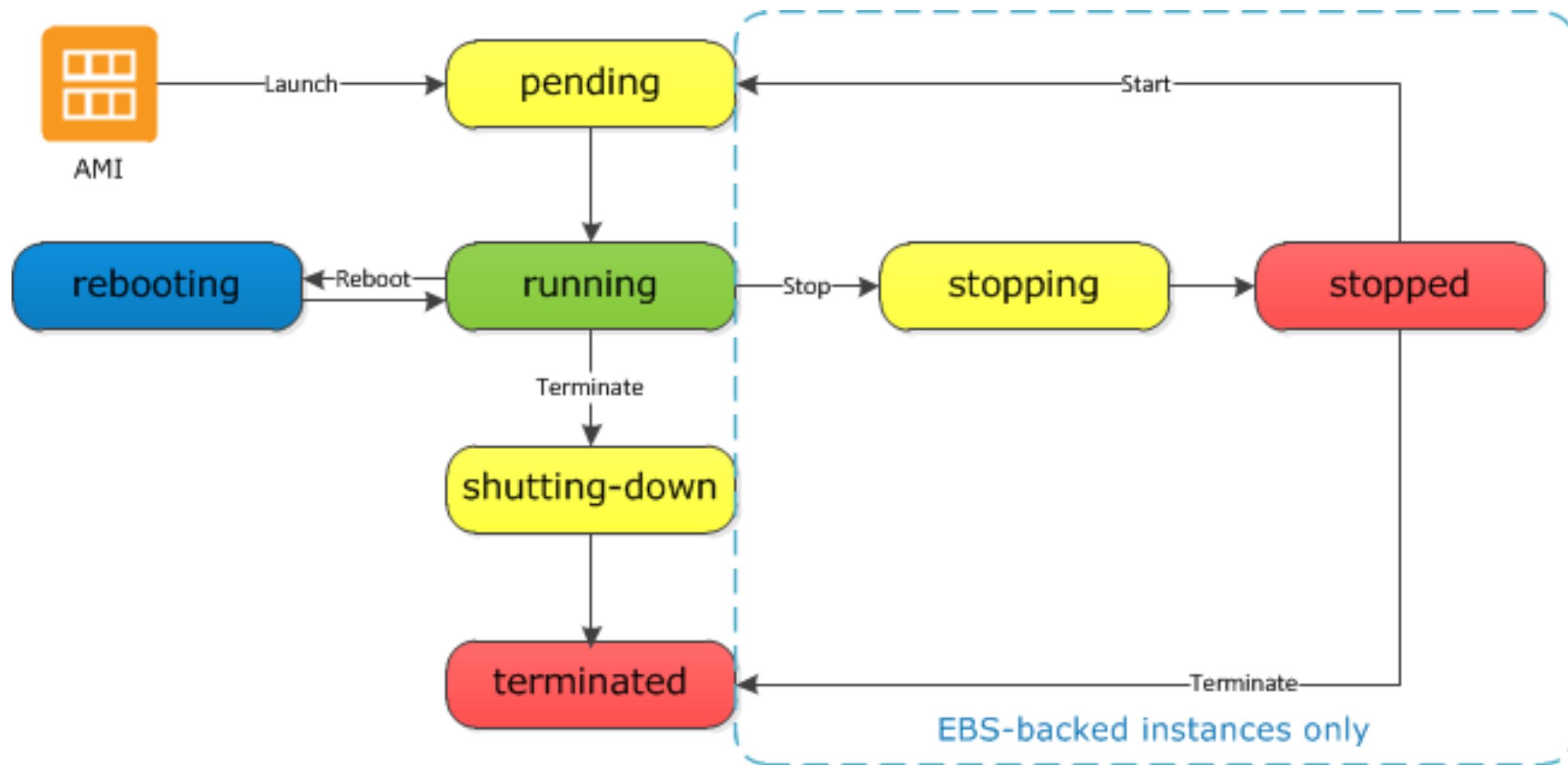
- >> Slides, labs and code <https://github.com/azat-co/aws-intro>
- >> AWS account
- >> Node and npm

15 minutes to download and install

? Questions? ?

# Module 2: Main EC2 Concepts

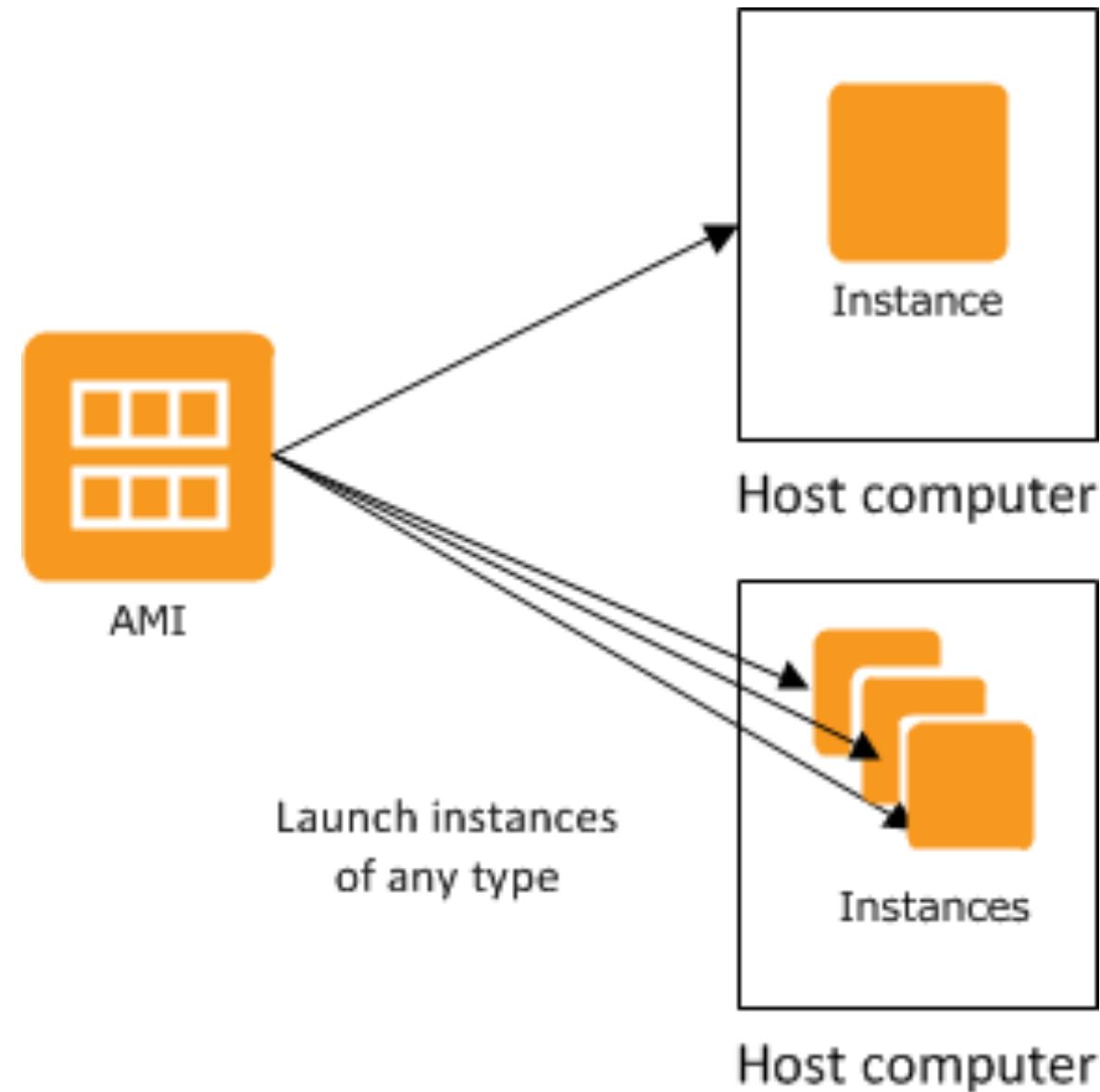
# EC2 Lifecycle



# "Stopped" to "running"

- » Rounded to an hour for billing (no charges for "stopped")
- » A new private IPv4 address unless in VPC or IPv6
- » A new public IPv4
- » Only EBS instances (more on storage later)
- » More *Differences Between Reboot, Stop, and Terminate*

# Images



# Images

- >> AMI can convert to/from other image types
- >> Marketplace
- >> Can create your own images
- >> images IDs for the same OS differ from region to region
- >> AMIs can be backed by EBS or instance store

# Images (cont)

>> OS, e.g., Amazon Linux

>> HVM

>> bit, e.g., 64 bit

>> Any software?

# Root Device: EBS vs. Instance store

- » EBS is faster to boot
- » More EBS can be added later
- » EBS allows to stop and restart
- » Persistent — EBS data is saved
- » Comparison Storage for the Root Device

# Types

>> Category(t)

>> Generation(2)

>> Size(micro)

t2.micro

All types: <https://aws.amazon.com/ec2/instance-types>

# IAM

# User Data

# Storage

- >> SSD
- >> Provisioned
- >> Magnetic

# Tags

# Spot vs. Reserved

# Key pairs

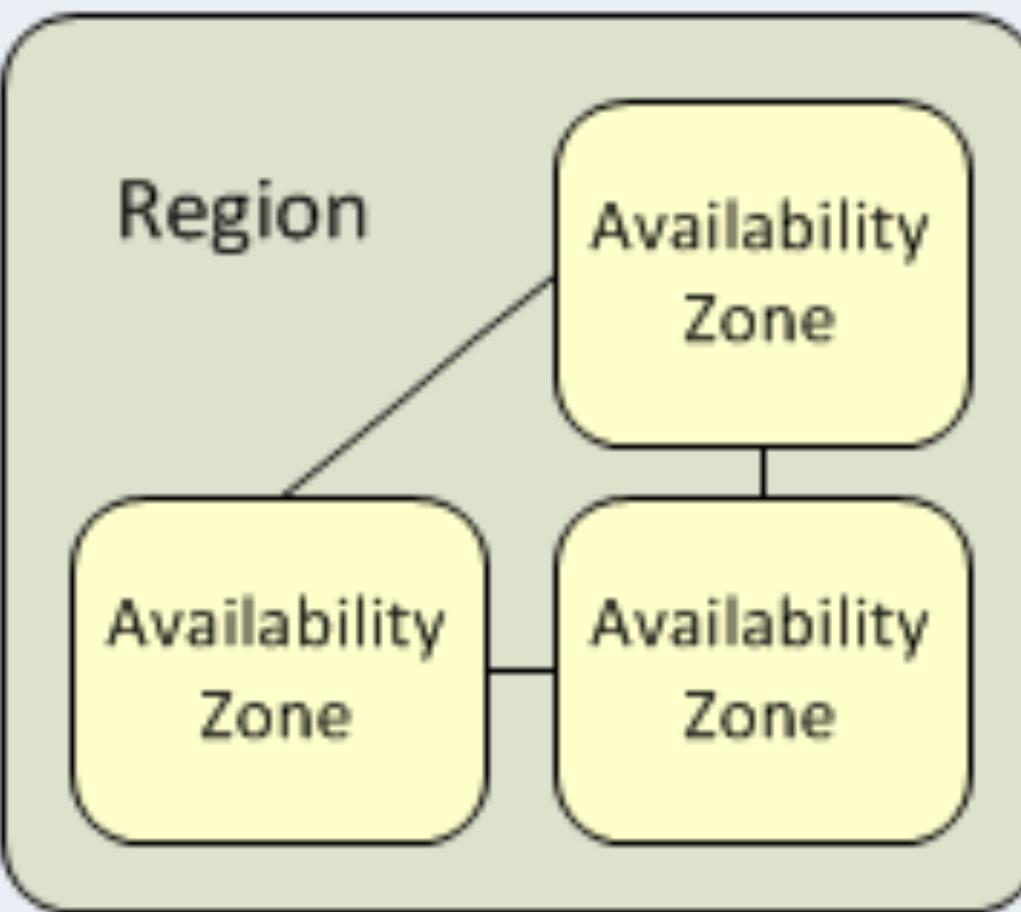
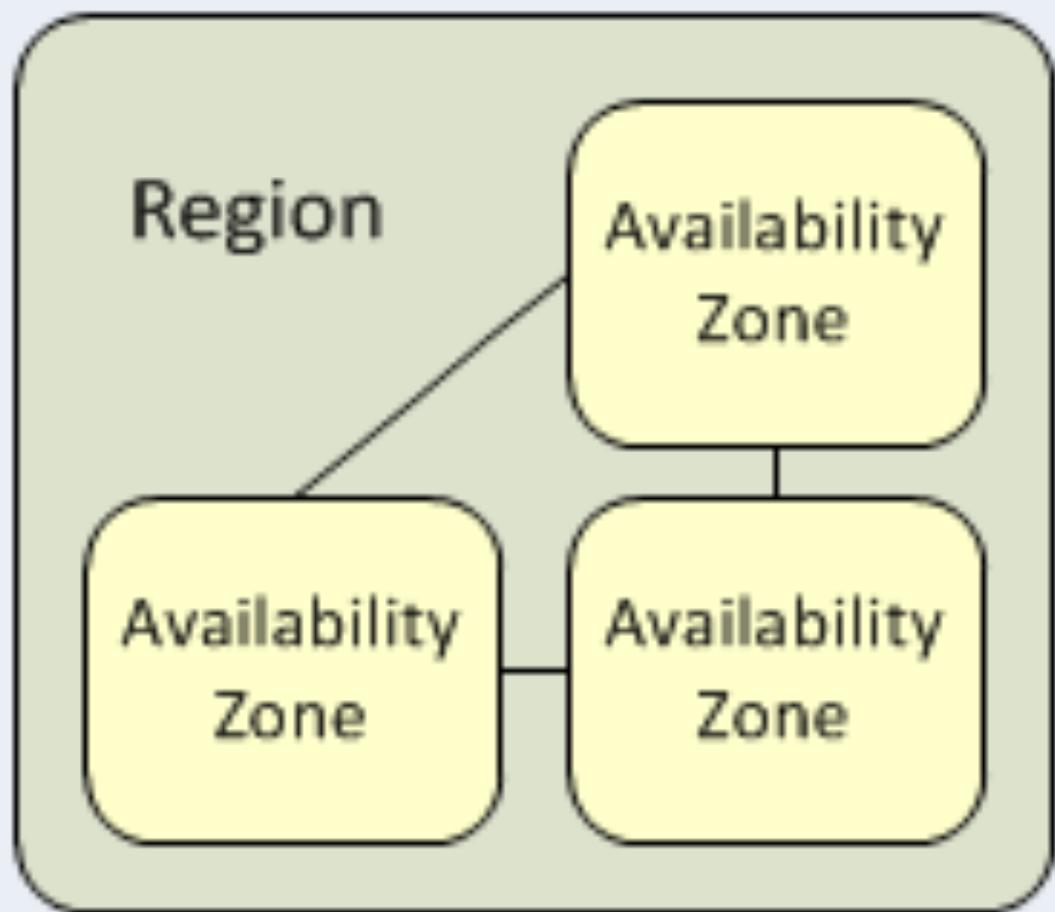
# Security groups

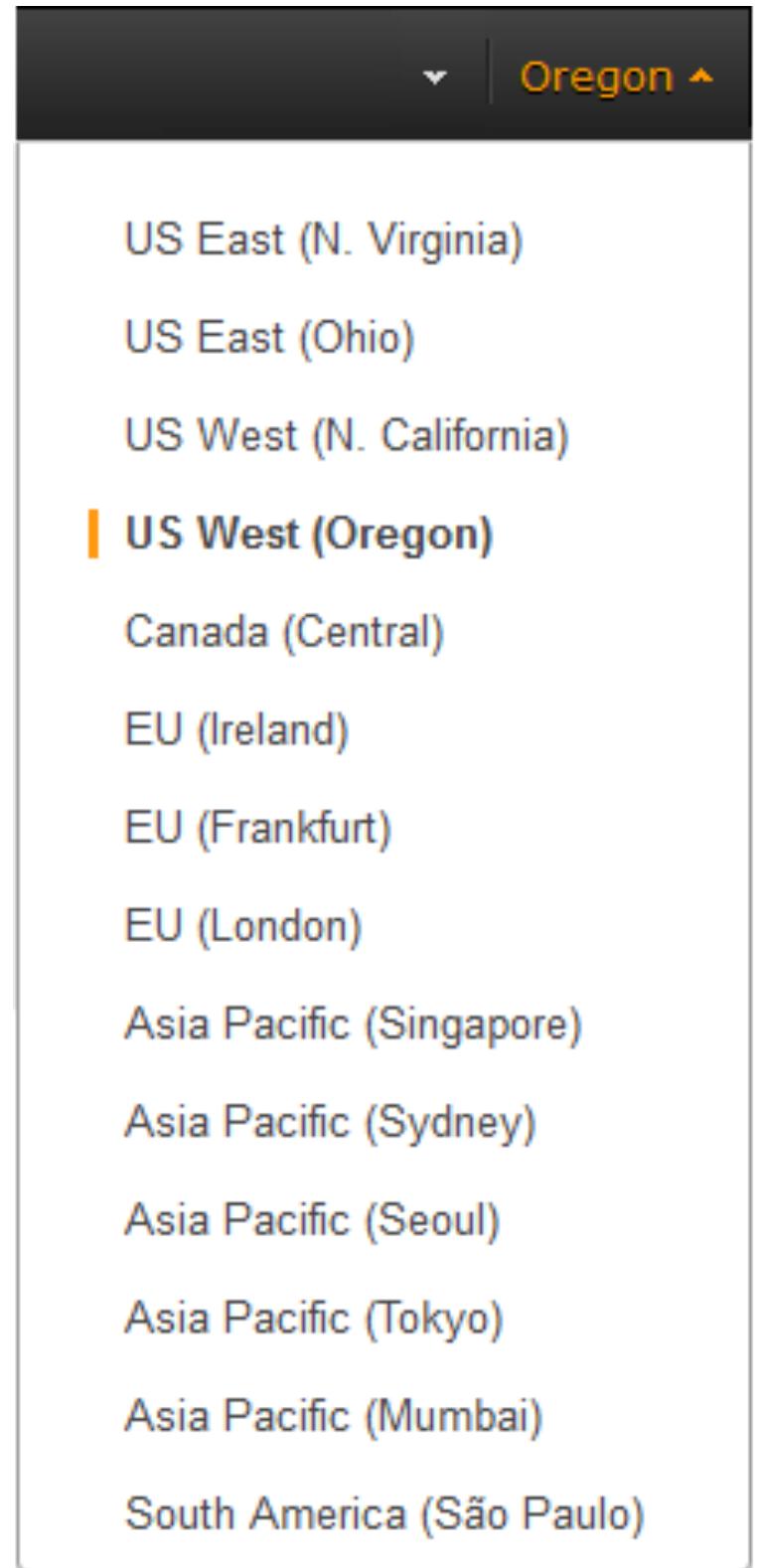
# ELB

# Elastic IP

# VPC, subnets and AZs

# Amazon Web Services





Code	Name
us-east-1	US East (N. Virginia)
us-east-2	US East (Ohio)
<b>us-west-1</b>	<b>US West (N. California)</b> ➡ USE THIS FOR LABS
us-west-2	US West (Oregon)
ca-central-1	Canada (Central)
eu-west-1	EU (Ireland)
eu-central-1	EU (Frankfurt)
eu-west-2	EU (London)
ap-northeast-1	Asia Pacific (Tokyo)
ap-northeast-2	Asia Pacific (Seoul)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)
ap-south-1	Asia Pacific (Mumbai)
sa-east-1	South America (São Paulo)

# Stop vs terminate

? Questions? ?

# Demo Time



- >> Demo: Launching EC2
  - >> Launching EC2: images (Amazon Linux), types (t2.micro), storage (8Gb), tags (role=aws-course)
  - >> Creating security groups (ICMP for ping, SSH and HTTP are open)
  - >> Working with SSH key pairs
  - >> Connecting to EC2 via SSH

? Questions? ?

# Lab 1: WP is King

Task: Create an instance with WordPress, make the website visible in the browser *publicly*

Detailed instruction in labs/lab-01.md

Time to finish: 10 min

# Module 3:

# Environment and App

# Deploy Automation

SSH-ing into each new machine  
manually is not optimal... what to  
do?

# User Data

2nd step, advanced settings

EC2 Management Console Ninja

Secure | <https://us-west-1.console.aws.amazon.com/ec2/v2/home?region=us-west-1#LaunchInstanceWizard>

Services | Resource Groups | Azat | N. California | Support

1. Choose AMI | 2. Choose Instance Type | **3. Configure Instance** | 4. Add Storage | 5. Add Tags | 6. Configure Security Group | 7. Review

### Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

**Number of instances**  Launch into Auto Scaling Group [\(i\)](#)

**Purchasing option**  Request Spot instances

**Network**  [\(i\)](#) [Create new VPC](#)

**Subnet**  [\(i\)](#) [Create new subnet](#)

**Auto-assign Public IP**  [\(i\)](#)

**IAM role**  [\(i\)](#) [Create new IAM role](#)

**Shutdown behavior**  [\(i\)](#)

**Enable termination protection**  Protect against accidental termination

**Monitoring**  Enable CloudWatch detailed monitoring  
Additional charges apply.

**Tenancy**  [\(i\)](#)  
Additional charges will apply for dedicated tenancy.

**Advanced Details**

**User data**  As text  As file  Input is already base64 encoded

(Optional)

1 2

[Cancel](#) [Previous](#) **Review and Launch** [Next: Add Storage](#)

When you specify user data, note the following:

- » User data is treated as opaque data (AWS won't check it, it's up to instance to interpret and execute)
- » Executed only at launch
- » Limited to 16 KB. (raw data, not base64-encoded)
- » User data must be base64-encoded before being submitted to the API. The AWS CLI and the Amazon EC2 web console do encoding for you. For more information about base64 encoding, see <http://tools.ietf.org/html/rfc4648>.
- » Must start with `#!/bin/bash` or similar for shell

Note: There are many other solutions for environment and app built, test and deploy automation. We will cover some of them in *AWS Intermediate*.

# Demo: User Data for Apache httpd and HTML page

```
#!/bin/bash  
yum install -y httpd  
service httpd start  
echo "Hello World!" > /var/www/index.html
```

# Demo: User Data for Apache httpd, PHP and PHP code

```
#!/bin/bash
yum update -y
yum install -y httpd24 php56 mysql55-server php56-mysqld
service httpd start
chkconfig httpd on
groupadd www
usermod -a -G www ec2-user
chown -R root:www /var/www
chmod 2775 /var/www
find /var/www -type d -exec chmod 2775 {} +
find /var/www -type f -exec chmod 0664 {} +
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
echo "<?php echo 'Hello World!' ?>" > /var/www/html/index.php
```

# Testing Apache httpd, PHP and Hello World

[http://PUBLIC\\_URL/](http://PUBLIC_URL/) or [http://PUBLIC\\_URL/index.php](http://PUBLIC_URL/index.php)  
[http://PUBLIC\\_URL/phpinfo.php](http://PUBLIC_URL/phpinfo.php)

Note: Never leave phpinfo in production because attackers can use the system info against the system!

# Cloud Init

Same Apache httpd, PHP and phpinfo config in User Data but with cloud-init instead of bash:

```
#cloud-config

repo_update: true
repo_upgrade: all

packages:
  - httpd24
  - php56
  - mysql55-server
  - php56-mysqlnd
```

# Cloud Init (cont)

Same Apache httpd, PHP and phpinfo config in User Data but with cloud-init instead of bash:

## **runcmd:**

- **service httpd start**
- **chkconfig httpd on**
- **groupadd www**
- [ **sh, -c, "usermod -a -G www ec2-user"** ]
- [ **sh, -c, "chown -R root:www /var/www"** ]
- **chmod 2775 /var/www**
- [ **find, /var/www, -type, d, -exec, chmod, 2775, {}, +** ]
- [ **find, /var/www, -type, f, -exec, chmod, 0664, {}, +** ]
- [ **sh, -c, 'echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php'** ]

# Cloud Init (cont)

For trouble shooting, cloud init logs are in `/var/log/cloud-init-output.log`.

More docs: <http://cloudinit.readthedocs.io/en/latest/index.html>

? Questions? ?

# Demo: User Data for a Node app with pm2 (restart on reboot)

hello-world-server.js:

```
const port = 3000
require('http')
.createServer((req, res) => {
  console.log('url:', req.url)
  res.end('hello world')
})
.listen(port, (error)=>{
  console.log(`server is running on ${port}`)
})
```

# User Data Bash for Node app with pm2 (Amazon Linux Example)

```
#!/bin/bash -ex

exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1
curl --silent --location https://rpm.nodesource.com/setup_6.x | bash -
yum -y install node.js
npm i -g pm2@2.4.3
echo "const port = 3000
...
" >> /home/ec2-user/hello-world-server.js
```

# How to Restart the app on reboot?

>> Crontab @reboot

>> /etc/rc.d/rc.local

>> chkconfig

>> Upstart

# Restarting with crontab

```
...  
crontab -l | { cat; \  
    echo "@reboot pm2 start /home/ec2-user/hello-world-server.js \  
        -i 0 --name \"node-app\""; } \  
    | crontab -  
sudo reboot
```

# Restarting with rc.d/rc.local

```
#!/bin/bash

...
sudo sed -i '$i export PATH=$PATH:.nvm/versions/node/v6.7.0/bin' \
/home/ec2-user/.bashrc
sudo sed -i '$i ./.nvm/nvm.sh' /etc/rc.d/rc.local
sudo sed -i '$i pm2 start /home/ec2-user/hello-world-server.js -i \
0 --name \"node-app\"' /etc/rc.d/rc.local
sudo reboot
```

Another way to install Node (via nvm):

```
#!/bin/bash

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.1/install.sh | bash
. /nvm/nvm.sh

sudo sed -i '$i export NVM_DIR="$HOME/.nvm"' /home/ec2-user/.bashrc
sudo sed -i '$i [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" ' /home/ec2-user/.bashrc
nvm install 6.7
```

# Pulling code from S3 (Amazon Linux Example)

```
#!/bin/bash  
yum install -y aws-cli  
cd /home/ec2-user/  
aws s3 cp 's3://my-private-bucket/{app-folder}/lastest.zip' /var/app \  
--region {bucket_region} # requires IAM role to access S3  
# unzip and launch
```

Note: AWS CLI covered in the AWS Intermediate course

# Pulling code from private GitHub repository (Amazon Linux Example)

```
#!/bin/bash  
yum install -y git-all  
git clone https://username:password@github.com/{username}/{private_repository}.git
```

Note: You can also use SSH (private key on EC2 and public on GitHub)

? Questions? ?

# Lab 2: Hello world Baby

Task: Launch an instance with "Hello World" HTTP server (Node.js), make the website visible in the browser *publicly*. Do not SSH!

Detailed instructions in labs/lab-02.md, Node server code is in code/index.js

Time: 20 min

? Questions? ?

# Module 4: Networking

Problem: EC2 Instances are  
publicly accessible... but most of  
the time you don't need them...  
what to do?

# VPC

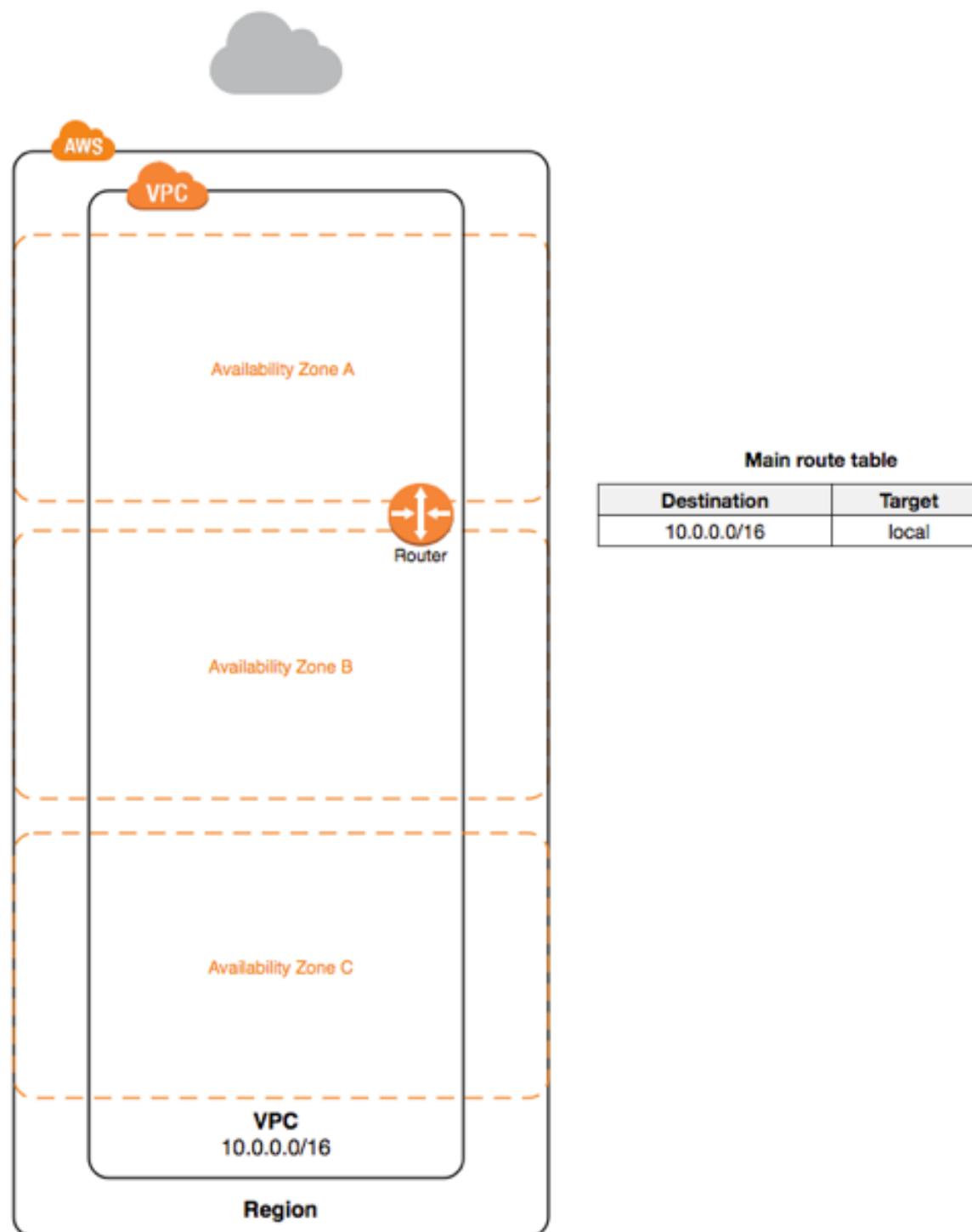
# What is VPC

VPC is a virtual network which is logically isolated from other virtual networks and dedicated to your AWS account.

# Benefits of VPC

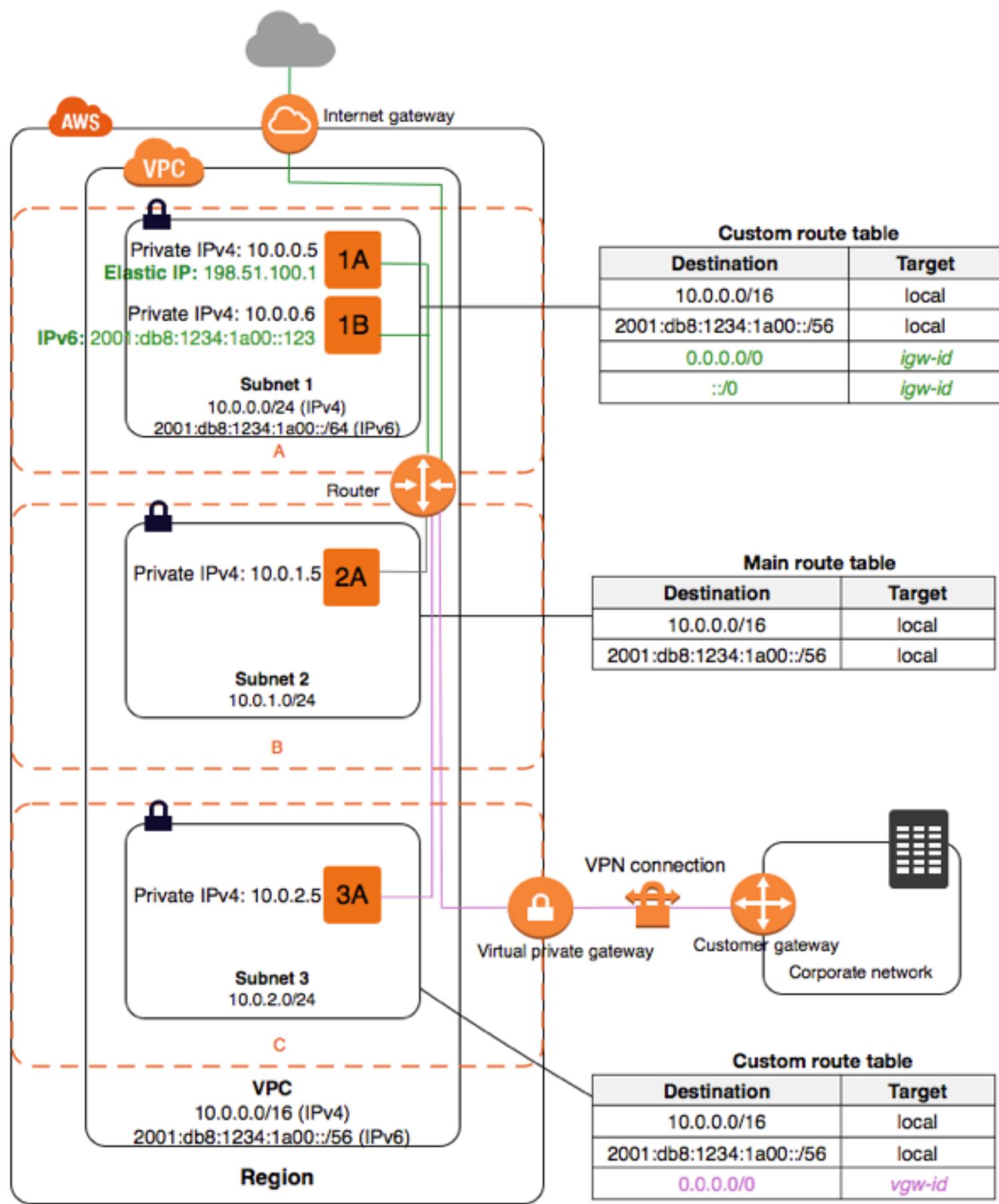
- » Security by controlling traffic to/from
- » Application performance
- » Bandwidth efficiency
- » Keep your private IP for EC2 on restart

# What VPC can do



- » Spans all AZ in the region
- » Connect to corporate or home networks
- » Create subnets which are limited to an AZ

# Subnets



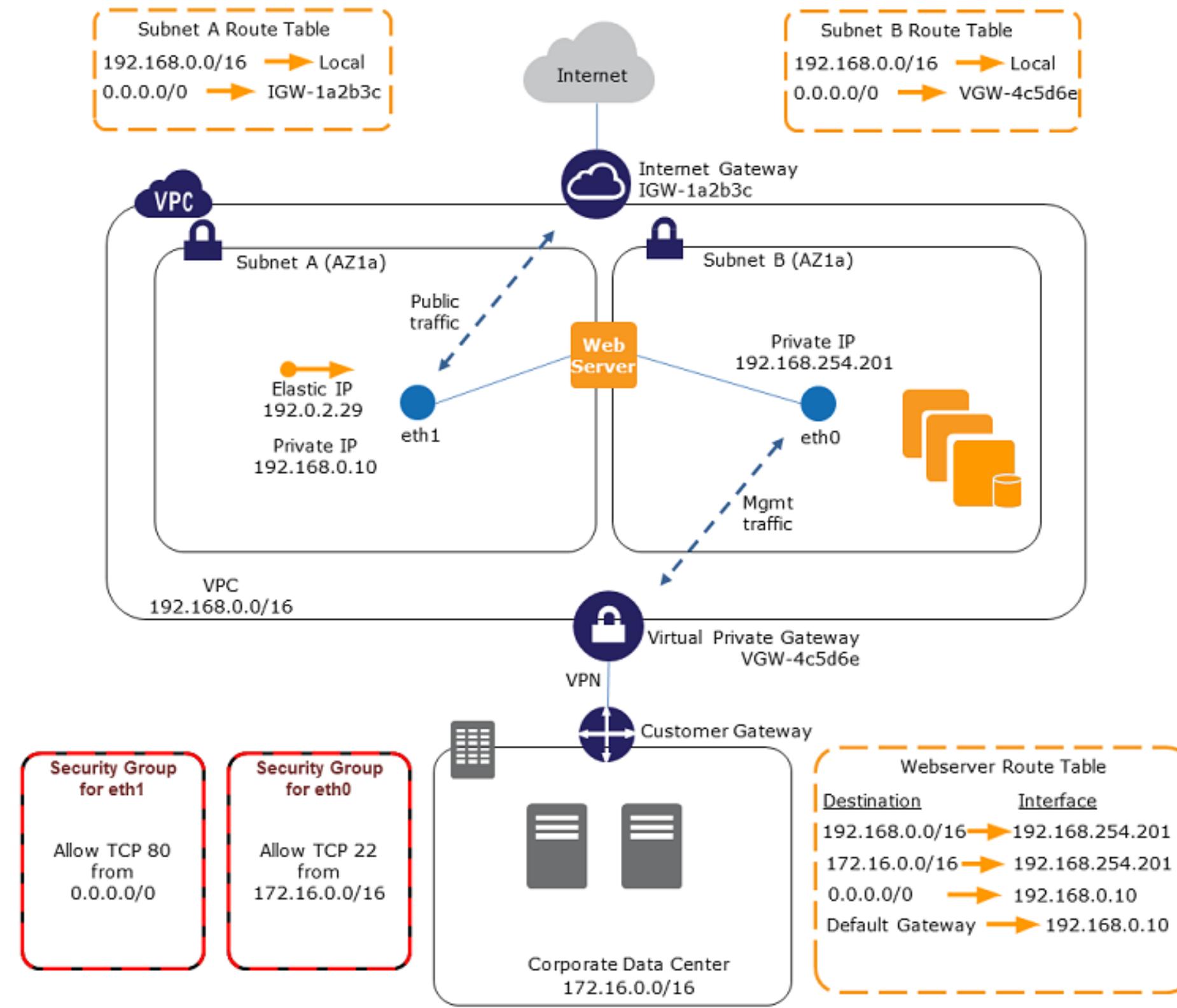
- >> 1A, 1B, 2A, and 3A are instances in the VPC.
- >> Subnet 1 is a public subnet (route to the Internet gateway)
- >> Subnet 2 is a private subnet (no route to the Internet gateway)
- >> Subnet 3 is a VPN-only subnet (route to a virtual private gateway)
- >> An Internet gateway enables communication over the Internet
- >> A virtual private network (VPN) connection enables communication with your corporate network

# Interface

An elastic network interface (or network interface or just interface) is a virtual network interface that you can attach to an instance in a VPC.<sup>1</sup> You can associate multiple instances with one interface.

---

<sup>1</sup> <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html>



# ELB

# Benefits

- >> Monitors health
- >> Routes to multiple AZs (fault tolerance)
- >> Sticky sessions (cookies)
- >> CloudWatch metrics
- >> More [here](#)

# Classic Load Balancer vs. Application Load Balancer

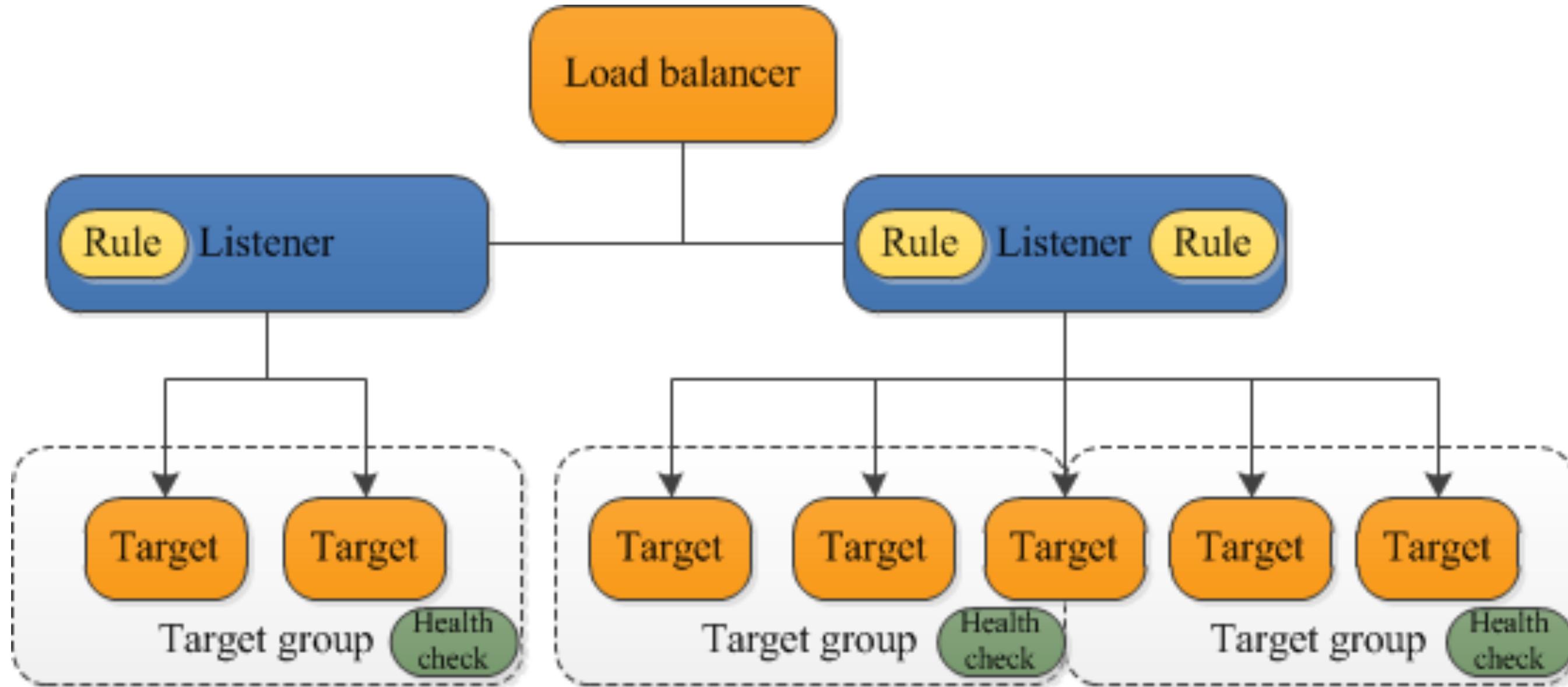
Classic Load Balancer — devs register **instances** with the load balancer.

Application Load Balancer — devs register the instances as **targets in a target group**, and routes traffic to a target group.

# Application Load Balancer over Classic LB

- » Cross-zone load balancing always enabled
- » Host-based routing and Path-based routing (microservices)
- » Route to multiple ports on a single instance (containers)
- » HTTP/2 support (security)
- » Websockets support
- » Load balancer deletion protection

# Application LB



# LB Scheme

- >> Internal
- >> Internet-facing

Note: *Internet-facing and internal load balancers route requests to your instances using private IP addresses. Hence, instances do not need public IP addresses to receive traffic from load balancers.*

# How ELB Works: Two Points of Elasticity

1. Client looks up DNS (azat.co) which by CNAME resolves to xxxx.us-west-1.elb.amazonaws.com
2. AWS gives ELB's IP – point 1
3. Client connects to ELB's IP
4. ELB passes traffic to one of the instances – point 2

# Elastic IP

# Benefits of EIP

- » Static *public* IPv4 (currently no IPv6) for DNS and sending emails (anti-spam)
- » Can associate with instance or network interface (many instances)
- » The ability to remap the address to another instance in the case of instance failure

# EIP

- » Limited to 5 per account per region
- » When we associate instance with EIP, it loses its previous public IP (if it had one)
- » Elastic IP address remains allocated until it is explicitly released

# Elastic IP vs ELB

ELB - more scalable and reliable (no single IP - see 2 points of elasticity); health checks; uses DNS (CNAME record), limited to 20 per region; cross-AZ.

EIP - static IP; uses IP (A record); specific region only, limited to 5 per region; one AZ.

? Questions? ?

# Demo

Create ELB and 2 EC2 instances with Apache httpd, make one fail  
- see the other running

? Questions? ?

# Lab 3: ELB for Save!

Task: Create an app ELB with 2 targets which are Apache httpd servers in different AZs, make one fail and observe

Detailed instruction in labs/03-elb.md

Time to finish: 15 min

? Questions? ?

# Module 5: S3

# Buckets

# HTTP access

# IAM Roles

# Versioning and multi-region

# Demo



Upload an object to S3 bucket via web console

? Questions? ?

# Lab 4: Static Website Rule!

Task: Serve an HTML page from S3

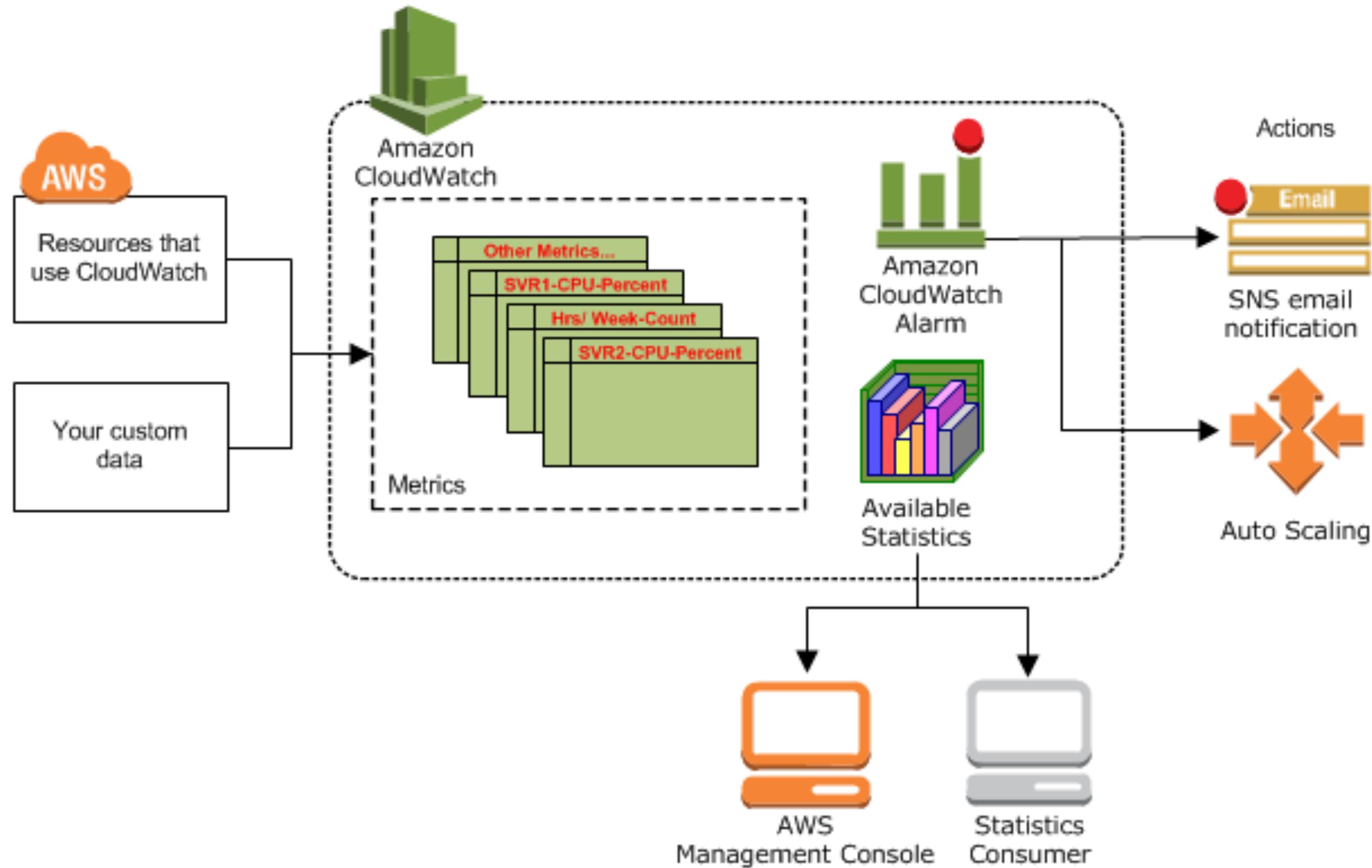
Detailed instruction in labs/04-static-rules.md

Time to finish: 10 min

? Questions? ?

# Module 6: Auto Scaling

# CloudWatch



# CloudWatch Use Cases

- >> Get an alert when your bill exceeds \$200/mo
- >> Shutdown underused machines
- >> Auto scale (up) machines under heavy load
- >> Log API calls (CloudTail)
- >> When EBS exceeds 100Mb through, put threshold

# Alerts

- » Watch a single metric from 1,068 CloudWatch metrics available in the US West (N. California), as an example
- » Send actions to SNS or Auto Scaling
- » Periods of 1min, 5min, 15min, etc.
- » You can create up to 5000 alarms per region per AWS account.

# Alert Types

- >> ALARM: Data for the metric is outside the threshold
- >> INSUFFICIENT\_DATA: No data or not enough or too early
- >> OK: Data for the metric is within the threshold

? Questions? ?

# Demo



Create a launch configuration with 1 initial Node app, autoscaling group, and a policy to increase instances, and then see if autoscaling works

? Questions? ?

# Lab 5: Pager Duty

Task: Deploy 2 Node apps under ELB, then load test it to see if autoscaling works

Detailed instruction in labs/03-pager-duty.md

Time to finish: 20 min

? Questions? ?

# End of Modules

