

# Welcome to Cloud Foundations

# 3 Week Agenda...

# Week 1

01

- **Accounts, security concepts/policies, building, how the cloud works common services, Docker and Kubernetes**
- **Shared responsibility model (operating model changes)**  
**Infrastructure Fundamentals:**
- **Compute (ECS, EKS, Serverless)**
- **Persistence (S3, RDS vs Aurora, Dynamo, etc.)**
- **Networking considerations**
- **Connectivity (availability zones, regions, resiliency)**
- **Security (IAM, SCPs)**



## Daily Breakdown – week 1

- Day 1:
  - Public vs Private vs Hybrid Cloud
  - IaaS vs PaaS vs SaaS vs FaaS
  - AWS intro/overview of common services
  - AWS regions and Availability Zones
  - Intro to IAM - Users, Roles, Policies
  
- Day 2:
  - IAM - permission boundaries
  - Shared Responsibility Model in AWS
  - (Brief) Overview of EC2
  - AWS Persistence - RDS, DynamoDB, S3 + Labs



# Daily Breakdown – week 1

- Day 3:
  - AWS Persistence continued...
  - Docker - intro
  - Docker - images, Dockerfiles, networks, Docker Compose
  - Docker multi-stage builds
  - Docker labs/hands-on
- Day 4:
  - Kubernetes - intro
  - K8s fundamentals - pods, services, replicaset, deployments, probes
  - K8s labs/hands - on
  - Intro to ECS/EKS - containerized solutions in AWS
- Day 5:
  - Intro to Fargate
  - ECS/EKS demos + labs/hands -on

# Week 2

02

## **Serverless Application Fundamentals (simple CRUD)**

- Designing for Cloud (design for failure, cost considerations)
- API Gateway
- Lambdas
- Aurora Postgres or S3
- Multi-Availability Zones and/or Multi-Region



# Daily Breakdown – week 2

- Day 1:
  - API Gateway
  - AWS Lambda Intro + Lab
  - AWS Encryption - KMS overview + Lab
- Day 2:
  - Build Serverless web application - hands on
  - Walk through/Demo of solution
- Day 3:
  - Discussion of IaC - Infra as Code
  - Terraform Intro
  - Terraform Hands on/walk-throughs
- Day 4:
  - Kubernetes - intro
  - K8s fundamentals - pods, services, replicaset, deployments, probes
  - K8s labs/hands - on
  - Intro to ECS/EKS - containerized solutions in AWS
- Day 5:
  - Intro to Fargate
  - ECS/EKS demos + labs/hands -on

# Week 3

03

## **DevSecOps Concepts**

- Code Build
- Code Deploy
- Provisioning Fundamentals
- Capstone project
- Architecting for high availability and disaster recovery



## Daily Breakdown – week 3

- Day 1:
  - DevOps + DevSecOps Overview
  - AWS CodeBuild + Lab
  - AWS CodePipeline
  
- Day 2:
  - AWS CodePipeline
  - Architecting for HADR high availability/disaster recovery
  
- Days 3-5:
  - Capstone Project
  - Capstone presentations

# Class Logistics



8 hours



1 hour lunch + breaks  
every hour

# Class Logistics

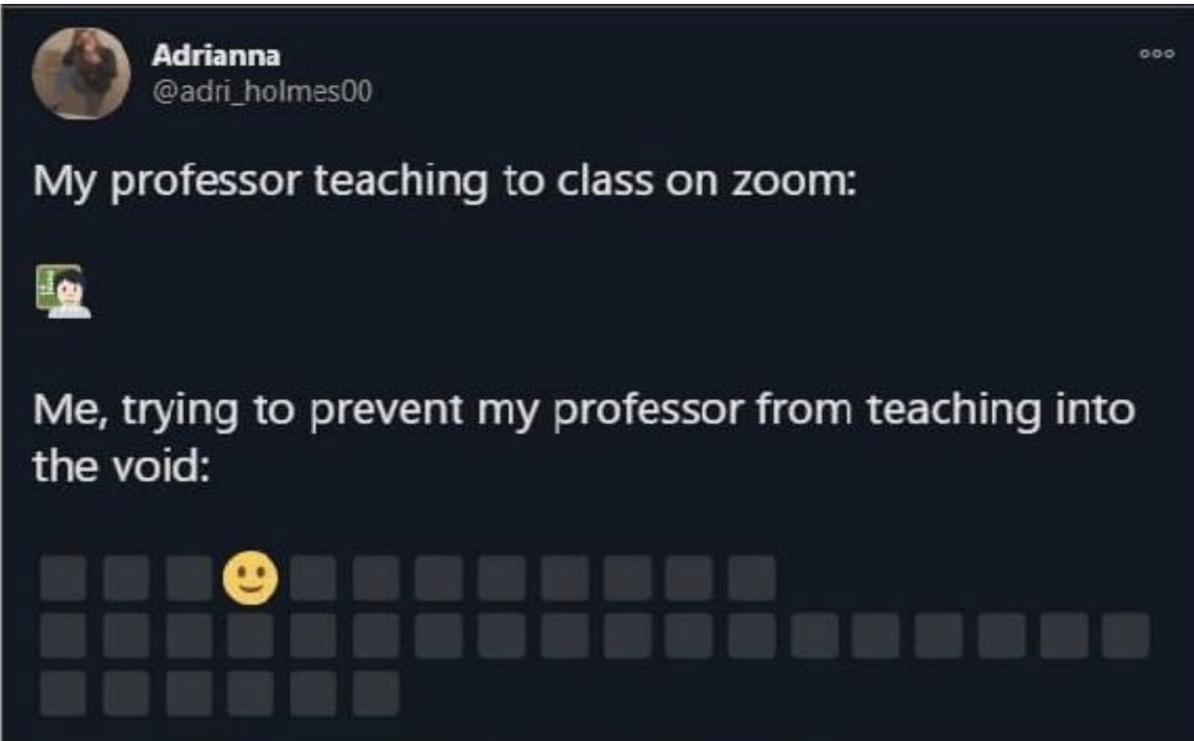


Eliminate  
distractions

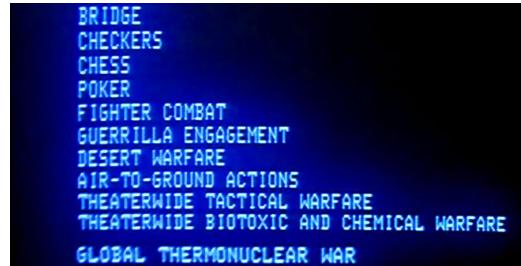


Enable video

# Class Logistics



# Would you like to play a game?



BRIDGE  
CHECKERS  
CHESS  
POKER  
FIGHTER COMBAT  
GUERRILLA ENGAGEMENT  
DESERT WARFARE  
AIR-TO-GROUND ACTIONS  
THEATERWIDE TACTICAL WARFARE  
THEATERWIDE BIOTOXIC AND CHEMICAL WARFARE  
GLOBAL THERMONUCLEAR WAR

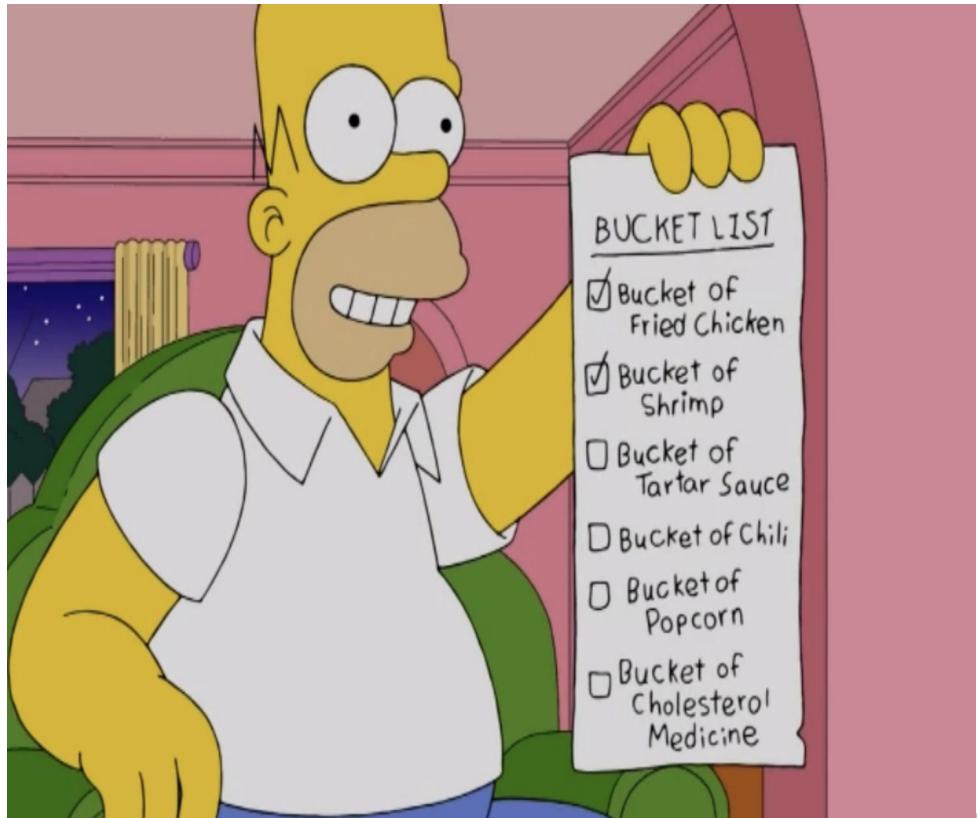
XR

- To make it interesting, let's add a competition factor to our time together ☺
- For questions posed to the larger group, 1 point will be awarded for each answer provided by a participant
- 2 points will be awarded for acting as a spokesperson for a breakout room during one of our group discussions
- There will be a rubric provided for the capstone (targeted for week 4) which will define a points breakdown for the implemented solution

# Objectives/Goals

After this class you should be able to:

- ❑ Describe cloud computing, its benefits, and its economics
- ❑ Explain AWS global infrastructure and understand considerations for resiliency
- ❑ Build serverless apps with common services (IaC Terraform, API Gateway, Lambdas, S3, Postgres)
- ❑ Leverage DevSecOps concepts and how it affects the building, testing, and release of software
- ❑ Provision cloud resources



# Introductions



- Please turn on your camera
- When called on, please share:
- Name, location
- Experience with AWS/other cloud providers
- What motivated you to take the class?
- One fun fact about yourself (like hobbies!)

# Notes

- We will be doing most of our work in virtual machines
- Plenty of hands on – roughly 70% of the class
- You will need to sign up for a personal Github account if you don't have one (free of course).

# Hosting Applications in the Cloud

# Infrastructure Options

## Before we get into AWS

- Let's ensure that we are on the same page cloud terminology wise
- And that we understand all the tradeoffs between public/private and IaaS/PaaS, etc..



## Infrastructure Options



Infrastructure is the hardware & software that run our IT workloads and that provide our business users and customers a way to interface with the applications required to complete their daily jobs

## What Are the Options?



On-Premise (in a Data Center)



Public Cloud



At the Edge



Hybrid Cloud

# What do they all mean?

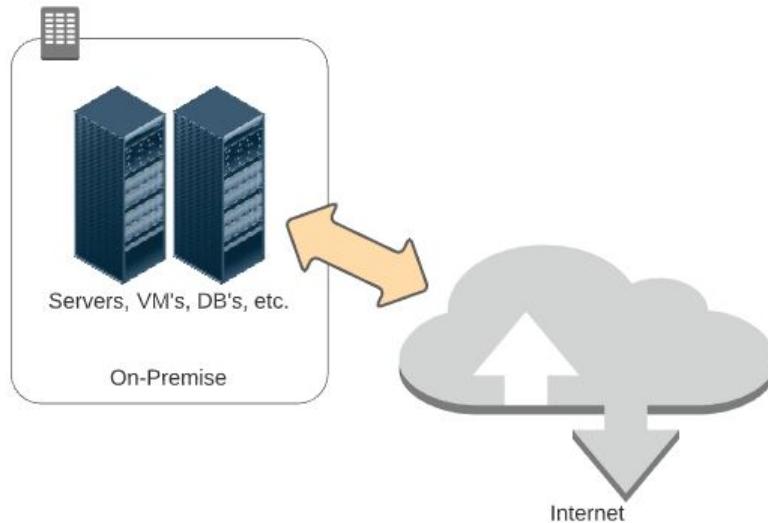
## On-Premise

Can mean a few different things:

- In a wholly-owned Data Center
- In a COLO (or co-location Data Center)
- Sometimes called a “private cloud”



## On-Premise





# On-Premise

Why and What?

- How infrastructure has traditionally been done
- With this model, companies try and estimate current & future hardware capacity needed to support business operations





# On-Premise

Why and What?

- Stakeholders plan out expected levels of consumption for the next 3 – 5 years (capacity to handle current volumes as well as expected growth)
- Some critical workloads may not be suitable for anything but a physical and directly-managed implementation (e.g., mainframe)



## On-Premise – Discussion



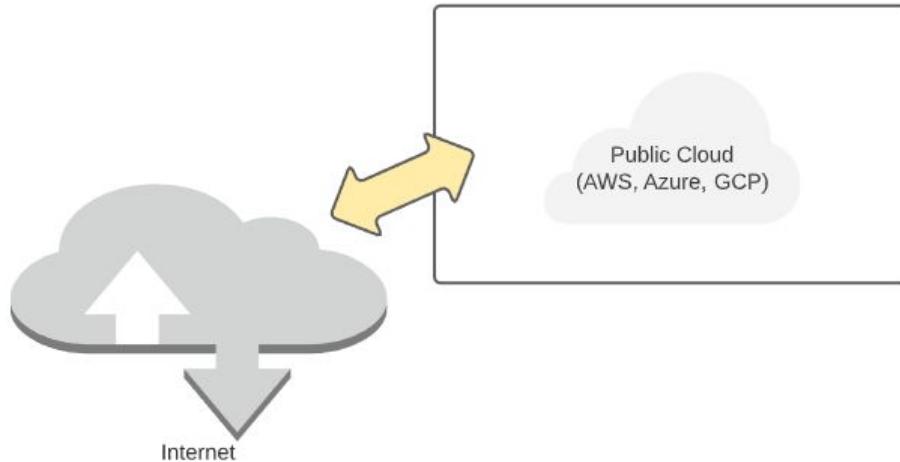
### Pros?

- Discrete capacity planning (even if that planning was off)
- Some workloads (e.g., mainframes and certain legacy systems) are tailor-made for a physical data center
- With a move to COLO's, companies could begin to share expenditure

### Cons?

- Sometimes difficult to know what is needed and when it is needed – if the plan was off (or unexpected spikes in demand occurred), difficult to adjust quickly
- Some workloads are just as effective (if not more so) in a virtual vs. physical implementation
- Harder to control costs and plan for costs – CAPEX vs. OPEX

# Public Cloud





# Public Cloud

Why and What?

- Platform using the standard “Cloud computing model” to provide infrastructure and application services
- Accessed and integrated via the Internet
- May provide a few different types of services – IaaS, PaaS, etc.





# Public Cloud

Why and What?

- Usually supports a subscription or “pay as you go” (on-demand) pricing model
- Largest players in this space include Azure, AWS and GCP



## Public Cloud - Discussion



### Pros?

- Flexibility and elasticity in capacity planning – enables automated schedule-based or metrics-based adjustments to capacity when required
- In some cases, managed services can be leveraged reducing operations overhead
- Because services are PAYG (pay as you go), you're only charged for what you use, and those expenses are OPEX

### Cons?

- Requires enough historical data for schedule-based planning or the right configuration for metrics-based planning
- With managed services you lose some levels of granular control
- Because of the flexibility/elasticity, it can be difficult to budget and, if Cloud services are not managed/monitored, costs can be high



# At the Edge

Why and What?

- It's about bringing the power of Cloud computing to you
- Enables additional processing closer to the sources of data while still supporting the offload of higher order processing to the Cloud
- Often involves setting up “Cloud-in-a-box” facilities on-premise





# At the Edge

Why and What?

- IoT (Internet of Things) is a good example – devices in a facility reading massive amounts of data can incorporate processing at the edge to improve overall efficiency
- Helps inject lower latency, increased security and improved bandwidth into systems used to aggregate critical data for an enterprise



## At the Edge - Discussion



### Pros?

- Allows distribution of processing power across a larger surface area
- Can be used to bring critical latency, security and bandwidth improvements to specific types of business workflows
- Efficiencies gained “at the edge” can help with managing the cost of processing data

### Cons?

- Requires more infrastructure and more configuration to support that distribution
- Increased distribution of processing power and activity can expand attack surface and requires the right configuration to ensure optimal interaction between system components (i.e., increased complexity)
- More components “at the edge” can lead to increased infrastructure costs



# Hybrid Cloud

Why and What?

- In many ways, an amalgamation of the other options
- Supports distribution of system processing across on-premise infrastructure and the public Cloud





# Hybrid Cloud

Why and What?

- Allows an enterprise to keep workloads that are best-suited for on-premise running on-premise while allowing migration of components that can move to the public Cloud
- Can help make an enterprise's move to the Cloud more gradual and planful



## Hybrid Cloud - Discussion



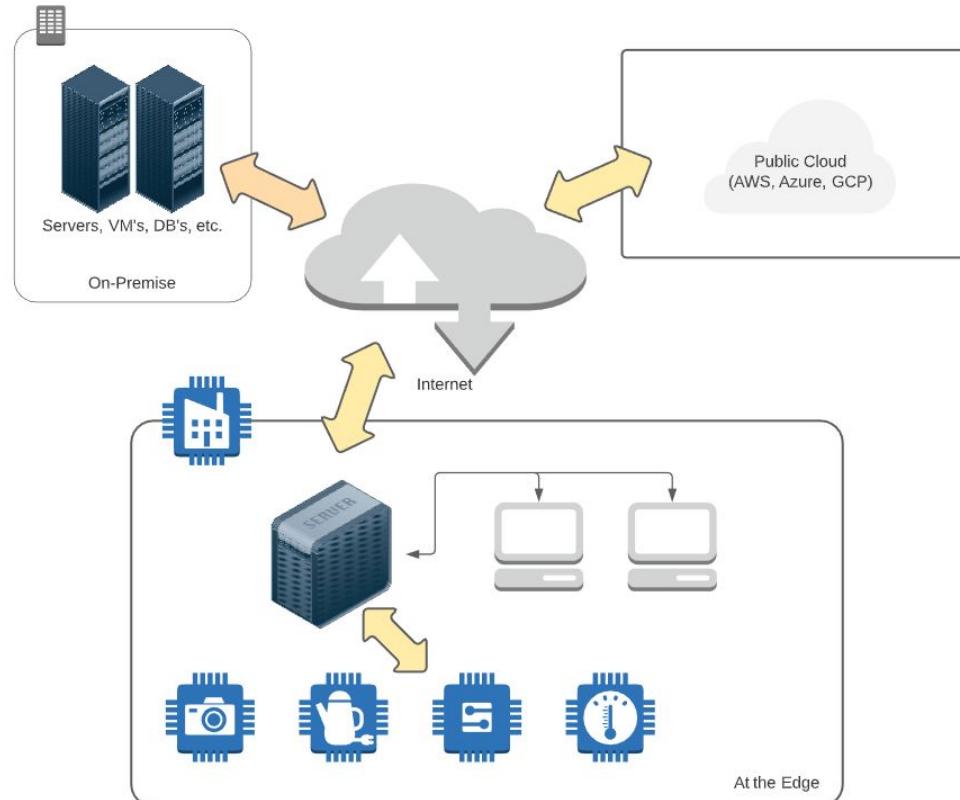
### Pros?

- Allows distribution of processing power across a larger surface area
- Can allow a move to the Cloud to be more gradual and allow an enterprise to target optimal deployment platform while making the move
- The ability to support a gradual move enables an enterprise to assess and understand Cloud costs over time

### Cons?

- Requires more infrastructure and more configuration to support that distribution
- As with Edge, can lead to increased complexity, often including required setup and maintenance of dedicated, secure connectivity between a data center and the Cloud
- If not managed optimally, costs can be higher due to need to pay for Cloud usage and data center (CAPEX + OPEX)

# Hybrid Cloud



## LAB:

### Infrastructure Options

Scenario: Your company (a global leader in FinTech) is currently hosting all infrastructure used to power the business in an on-premise Data Center. This includes a mainframe for primary business functions (customer management, account management, accounts payable, accounts receivable), several Web Apps (for customer interaction), several Web APIs providing backend data and functionality to the UIs, and a system used to manage data feeds from several security cameras used at corporate offices for observation and security.

As a member of the technical staff, you have been asked to provide thoughts and recommendations on moving from the Data Center to the Cloud.

In your assigned breakout room, discuss as a group and be prepared to provide the following: 1) Potential options for infrastructure in the Cloud for the different types of workload, and 2) considerations that the company should keep in mind as they make the move to ensure awareness and proactive planning.

Nominate someone (or volunteer) to share your group's ideas.

## Knowledge Check



With this infrastructure option, stakeholders try to estimate hardware & software needs for the next 3 – 5 years:

- A. Public Cloud
- B. At the Edge
- C. Hybrid
- D. On-Premise

## Knowledge Check



This infrastructure option focuses on bringing additional power closer to the data sources and application users:

- A. Public Cloud
- B. At the Edge
- C. Hybrid
- D. On-Premise

## Knowledge Check



This infrastructure option seeks to build a solution by combining the other options into a larger solution:

- A. Public Cloud
- B. At the Edge
- C. Hybrid
- D. On-Premise

# Application Hosting

## Application Hosting

By Application Hosting, we mean the target infrastructure and runtime platform used for deployment and execution of an application or system; can include compute (CPU and server resources), storage, network, data and operating system

## Application Hosting – An “Interesting” Example?

Here's an example of someone thinking “outside-of-the-box” when it comes to application hosting!

<https://mashable.com/article/pregnancy-test-doom/>

## What Are the Hosting Options with Cloud?

- IaaS
- PaaS
- Serverless / FaaS
- SaaS
- Containers



What do they all mean?



# Infrastructure-as-a-Service (IaaS)

- Involves the building out (and management) of virtual instances of:
  - Compute
  - Network
  - Storage
- Akin to spinning up a server (physical or virtual) in your location or data center complete with disks and required network connectivity





## Infrastructure-as-a-Service (IaaS)

- The difference is in the where – instead of in your data center, it is created in a data center managed by one of the public Cloud providers
- Your organization is responsible for patching the OS, ensuring all appropriate security updates are applied and that the right controls are in place to govern interaction between this set of components and other infrastructure





# Platform-as-a-Service (PaaS)

- Involves leveraging managed services from a public Cloud provider
- With this model, an enterprise can focus on management of their application and data vs. focusing on management of the underlying infrastructure
- Patching and security of the infrastructure used to back the managed services falls to the CSP (Cloud Service Provider)





# Platform-as-a-Service (PaaS)

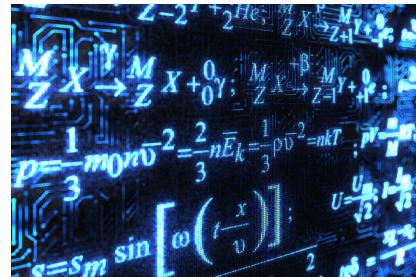
- Many managed services support automatic scale up or down depending on demand to help ensure sufficient capacity is in place
- Can be considered synonymous with the term “Cloud native”





# Serverless / Functions-as-a-Service (FaaS)

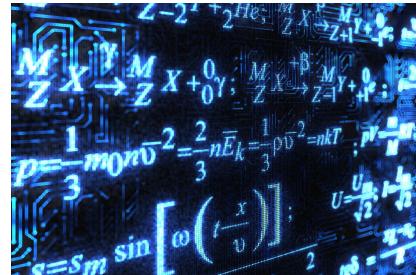
- Also represents a type of managed service provided by the CSP
- Cost structure is usually consumption-based (i.e., you only pay for what you use)
- Supports many different coding paradigms (C#/.NET, NodeJS, Python, etc.)





# Serverless / Functions-as-a-Service (FaaS)

- Typically, with Serverless (and PaaS), the consumer is only concerned with the application code and data – elements of the CSP's "backbone" used to support are managed by the CSP
- Includes more sophisticated automated scaling capabilities – built for Internet scale





# Software-as-a-Service (SaaS)

- Subscription-based application services
- Licensed for utilization over the Internet / online rather than for download and install on a server or client machine
- Fully-hosted and fully-managed by a 3<sup>rd</sup> party

```
position: absolute; z-index: 999; top: -5px; left: -5px; width: 10px; height: 10px; border-radius: 50%; background-color: #ccc; display: block; position: absolute; opacity: 1; top: -2px; left: -5px; width: 4px; height: 4px; border-radius: 50%; background-color: #ccc; display: inline-block; font-size: 10px; vertical-align: middle; margin-right: 5px; cursor: pointer; display: block; text-decoration: none; color: inherit; outline: none; border: none; font-family: inherit; font-weight: inherit; font-style: inherit; font-size: inherit; line-height: 27px; padding: 0; margin: 0; position: relative; z-index: 1000; } .gbts { *display: inline-block; padding-right: 9px; } .gbz { .gbz { background: url(../img/icon-saas.png) no-repeat center center; width: 16px; height: 16px; }
```



# Software-as-a-Service (SaaS)

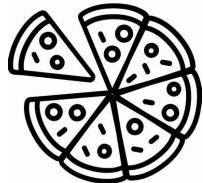
- Of those discussed, often the cheapest option for service consumers
- However, also offers minimal (or no) control, outside of exposed configuration capabilities

```
position: absolute; z-index: 999; top: -5px; left: -5px; width: 10px; height: 10px; border-radius: 50%; background-color: #ccc; display: block; position: absolute; opacity: 1; top: -2px; left: -5px; width: 4px; height: 4px; border-radius: 50%; background-color: #ccc; display: inline-block; font-size: 10px; vertical-align: middle; margin-right: 5px; cursor: pointer; display: block; text-decoration: none; color: inherit; outline: none; border: none; font-family: inherit; font-weight: inherit; font-style: inherit; font-size: inherit; line-height: 27px; padding: 0; margin: 0; position: relative; z-index: 1000; } .gbts { *display: inline-block; padding-right: 9px; } .gbz { background: url(../img/icon-saas.png) center no-repeat; width: 16px; height: 16px; }
```

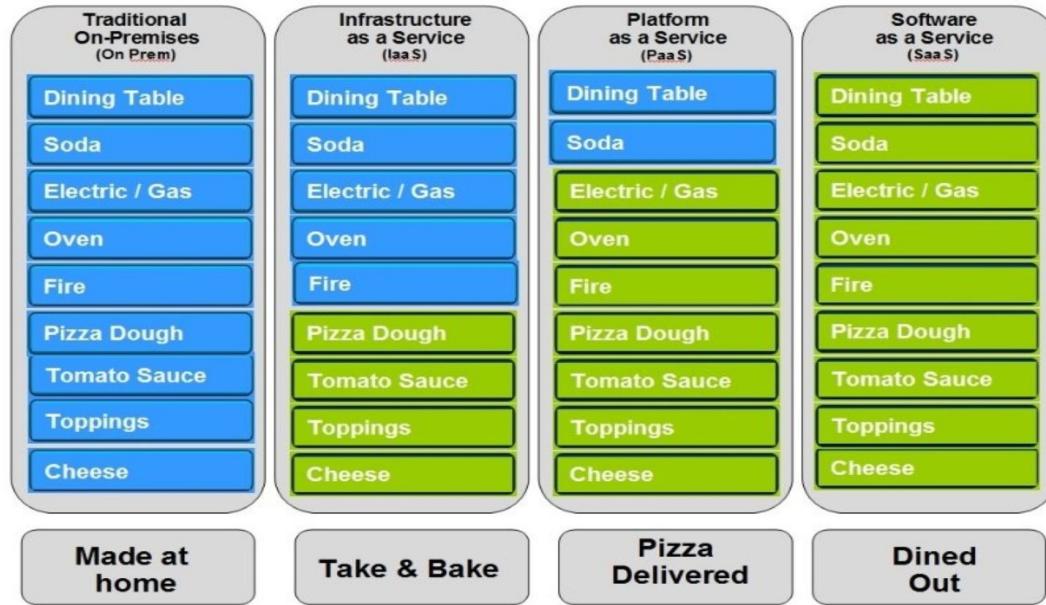
# Pizza-as-a-Service

From a LinkedIn post by Albert Barron from IBM

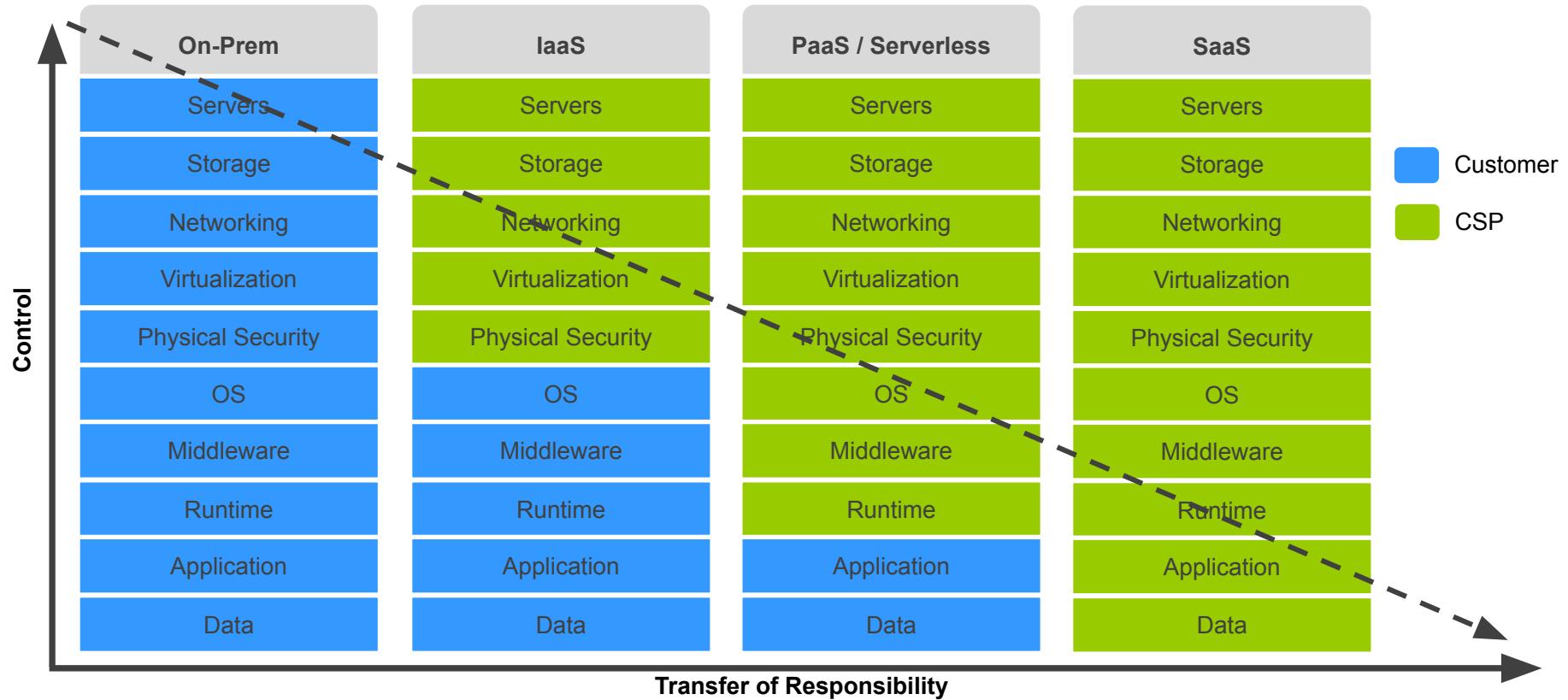
(<https://www.linkedin.com/pulse/20140730172610-9679881-pizza-as-a-service/>)



## Pizza as a Service



# Side-by-Side Comparison



# Containerization in the Cloud

- One option includes standing up VM's (IaaS) and installing / managing a Kubernetes cluster on those machines or
- Another option includes leveraging a managed service (PaaS) provided by the CSP
- Options in AWS include Elastic Container Service (ECS), Elastic Container Registry (ECR), and EKS (Elastic Kubernetes Service)



## Which One is Better?

- The answer is “it depends”
- It depends on the type of application
- It depends on the enterprise





## Which One is Better?

- It depends on the skillset and expertise within the organization
- It depends on whether you have budget and opportunity to modernize an application environment (in some cases)
- The best option might be a combination of multiple approaches – right tool for the right job



## LAB:

### Application Hosting

Scenario: Your company (a global leader in FinTech) is currently hosting all infrastructure used to power the business in an on-premise Data Center. This includes a mainframe for primary business functions (customer management, account management, accounts payable, accounts receivable), several Web Apps (for customer interaction), several Web APIs providing backend data and functionality to the UIs, and a system used to manage data feeds from several security cameras used at corporate offices for observation and security.

As a member of the technical staff, you have been asked to provide thoughts and recommendations on moving from the Data Center to the Cloud.

In your assigned breakout room, discuss as a group and be prepared to provide the following: 1) Recommendations for the types of hosting that could be used for the various application components, and 2) the reasoning behind your recommendations – i.e., what are some benefits you would expect the company to receive by implementing your recommendations.

Nominate someone (or volunteer) to share your group's ideas.

## Knowledge Check



With this application hosting option, you primarily only pay for what you actually consume:

- A. IaaS
- B. PaaS
- C. Serverless/FaaS
- D. SaaS

## Knowledge Check



This application hosting option is used to license certain “commodity” functionality like email, CRM, etc. over the Internet:

- A. IaaS
- B. PaaS
- C. Serverless/FaaS
- D. SaaS

## Knowledge Check



Of the application hosting options outlined below, which of them provides the combination of greater control but less transfer of responsibility:

- A. IaaS
- B. PaaS
- C. Serverless/FaaS
- D. SaaS

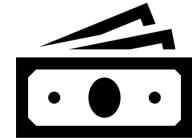
# **AWS – the basics... history, regions, and AZ's**

# AWS – brief history



- 2006 – AWS first public launch with S3 – Simple Storage Solution and shortly followed by EC2 – Elastic Cloud Compute
- GCP (Google Cloud Platform) launched in 2008
- 2010 - AWS launches IAM (Identity Access Management) and CloudFormation. Microsoft Azure also launched this year

# AWS – revenue



- 2019 Revenue - \$35 Billion
- 2020 Revenue - \$45 Billion
- 2021 Revenue - \$62 Billion
- You can see where this is going – growth of the cloud is tremendous!

# AWS – cloud revenue share



- ❑ AWS controlled about **39%** of the cloud infrastructure market in 2021, down from 41% in 2020
- ❑ Microsoft Azure is second

# AWS – more profitable than amazon.com



- ❑ Amazon overall generated \$24.8 billion in operating profits in 2021, and AWS was responsible for \$18.5 billion (or 74%) of it.
- ❑ Basically, a business segment that contributes 14% of overall revenue is generating roughly three-quarters of Amazon's total operating profits.



# AWS Regions

- Where do AWS servers physically live?
- In AWS regions!
- Region == *cluster of data centers*
- These are different geographic locations where AWS has its data centers – like Ohio, N. California, etc. are sample regions
- *Exact* location of regions is secret - to avoid potential attack (AWS runs a lot of the internet, one data center going down could cause severe damage)



# AWS Regions

- These are some sample regions from the AWS console
- Currently 30 regions worldwide (as of early 2023) w/ plans to add 5 more



<b>US East (N. Virginia)</b>	<b>us-east-1</b>
<b>US East (Ohio)</b>	<b>us-east-2</b>
<b>US West (N. California)</b>	<b>us-west-1</b>
<b>US West (Oregon)</b>	<b>us-west-2</b>
<hr/>	
<b>Asia Pacific (Mumbai)</b>	<b>ap-south-1</b>
<b>Asia Pacific (Osaka)</b>	<b>ap-northeast-3</b>
<b>Asia Pacific (Seoul)</b>	<b>ap-northeast-2</b>
<b>Asia Pacific (Singapore)</b>	<b>ap-southeast-1</b>
<b>Asia Pacific (Sydney)</b>	<b>ap-southeast-2</b>
<b>Asia Pacific (Tokyo)</b>	<b>ap-northeast-1</b>



# What regions do we use at CG?

- us-east-1 and us-west-2



# How do we choose an AWS region?

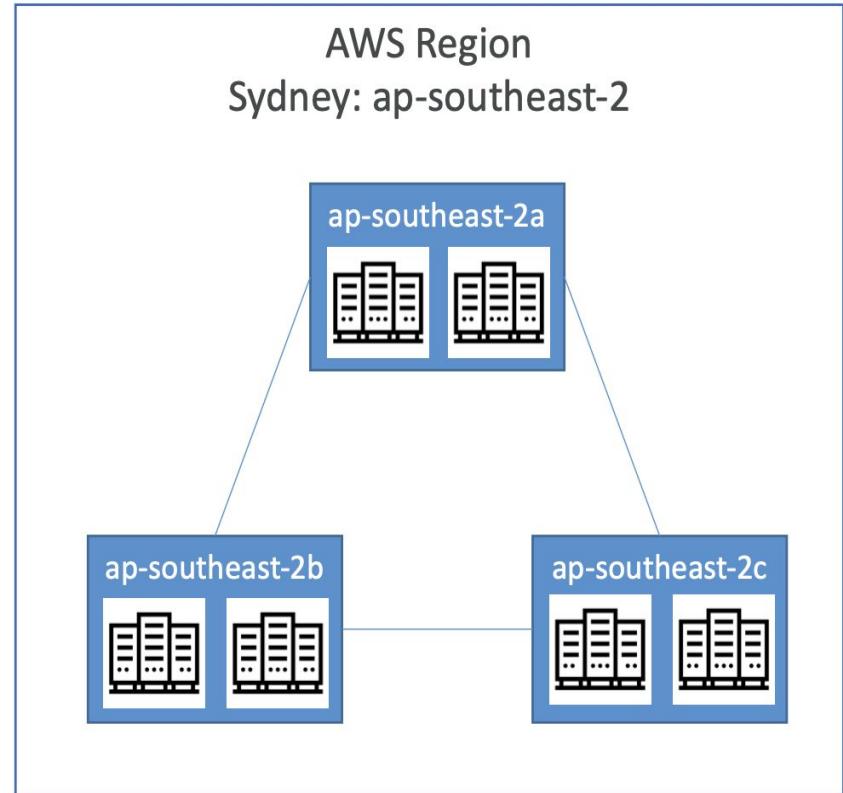
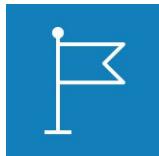


- When we launch an application in AWS, we have to choose a region – so how do we choose? There's a few considerations:
- **Location/Proximity:** Typically you want your application to live on servers that are physically closest to your end users
- **Availability within a region:** Some AWS services are not available within certain regions
- **Keeping Compliance:** some governments (like Canada) have strict requirements saying that their citizens data must be stored on servers in the same country



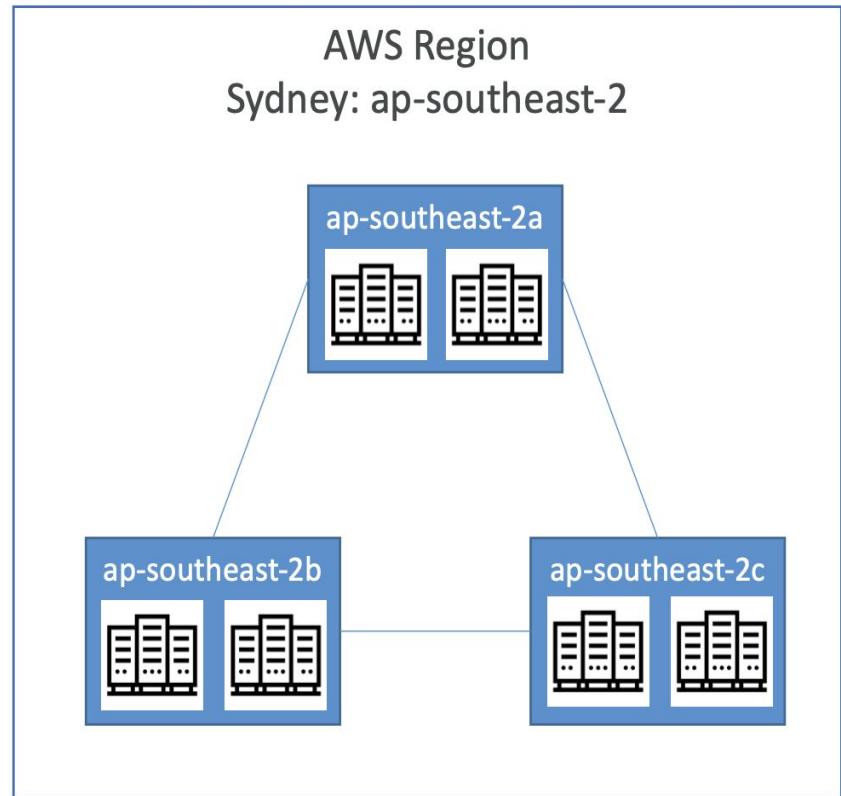
# Availability Zones (AZ's)

- ❑ AWS has multiple AZs (3-6) within a given region
- ❑ On the right you can see the Sydney Australia region of ap-southeast-2 – and within that region there are 3 AZ's
- ❑ AZ's distinguished from one another by the trailing letter, ie the "a" in ap-southeast-2a
- ❑ Each AZ is a physically separate data center, but they are not completely independent of each other..



# Availability Zones (AZ's) - redundancy

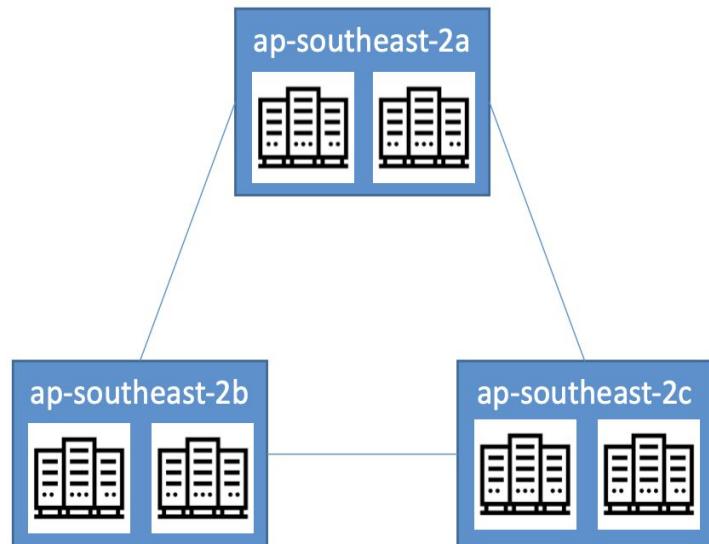
- Although AZ's are separate they are inter-connected via a low-latency network...
- This way if one AZ goes down (maybe a tornado hits), if your workloads reside on multiple AZ's you should not experience downtime



# Availability Zones (AZ's) - redundancy

- AZ's are physically separate by many kilometers, although they are all within 100 km of each other
- Key point: AZ's give ***additional redundancy in a region***
- Another example: us-east-1 has 6 AZ's, us-west-1 has 3 AZ's...it varies but most regions have 3 AZ's (for now)
- So, if one AZ goes down in us-east-1 then there's still 5 others. The entire region does not go down

AWS Region  
Sydney: ap-southeast-2





# Availability Zones (AZ's)

- ❑ Again, here we can see that AZ's are distinguished from one another with a trailing letter in the region name
- ❑ So if our region is **us-east-1**, the AZs in that region are **us-east-1a**, **us-east-1b**, **us-east-1c**, **us-east-1d**, **us-east-1e**, and **us-east-1f**

subnet-0ed4a0219eb649f9b

VPC: vpc-072af642d197ce683 Owner: 356042463280 Availability Zone: us-east-1e  
IP addresses available: 4091 CIDR: 172.31.48.0/20)

subnet-04645e429be0c8ebe

VPC: vpc-072af642d197ce683 Owner: 356042463280 Availability Zone: us-east-1c  
IP addresses available: 4091 CIDR: 172.31.16.0/20)

subnet-0b62d9bfbafd5c8b5

VPC: vpc-072af642d197ce683 Owner: 356042463280 Availability Zone: us-east-1a  
IP addresses available: 4091 CIDR: 172.31.0.0/20)

subnet-0a941e9351418997d

VPC: vpc-072af642d197ce683 Owner: 356042463280 Availability Zone: us-east-1d



# AWS – global and region scoped services



- ❑ To build apps on the cloud we need a lot of different services
- ❑ AWS has some services that are *global*, meaning they are not limited by region – like IAM, Route 53 (DNS), CloudFront
- ❑ But most AWS services are *region-scoped* – like Lambda, EC2, and more!
- ❑ Also see:  
<https://aws.amazon.com/about-aws/global-infrastructure/regions-product-services/>

# IAM - Identity Access Management

# IAM – why?



- IAM helps us to define who can do what
- Some people (users) in your organization will have more permissions than others
- Some users need admin rights and some will need lesser privileges inside of AWS

# IAM – more on why?



- ❑ AWS costs real money - easy to forget with cloud resources
- ❑ So we don't want everyone to have unlimited access b/c mistakes can be made
- ❑ Especially if people don't have much AWS experience!

# IAM – global service



- IAM is a *global* service
- Since users, unlike cloud resources (VM's, databases, storage and such), are not tied to a particular region/data center so this makes sense
- Quick demo of this



# IAM – root account



- ❑ The root account allows full access to your AWS resources
- ❑ If your credentials are stolen then anyone can create resources under your account and incur costs

Even for admin tasks, you can create admin users through IAM to do everything you need – avoid using root

# How many AWS accounts do you need?

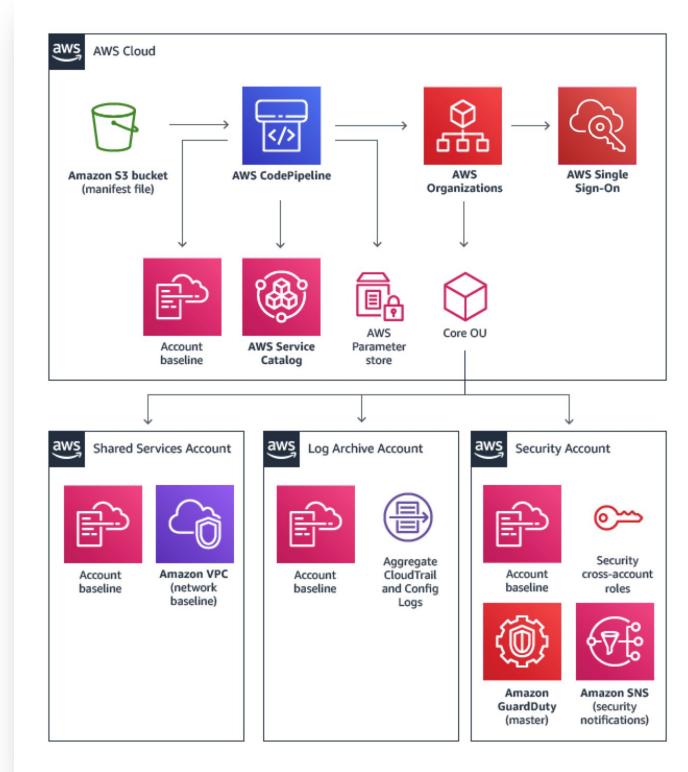


Typically more than one account is always required.

First account is the Master Account (or Management Account). Bills of all the accounts get consolidated here. You get tiered discount as well.

Other accounts (as per the best practices):

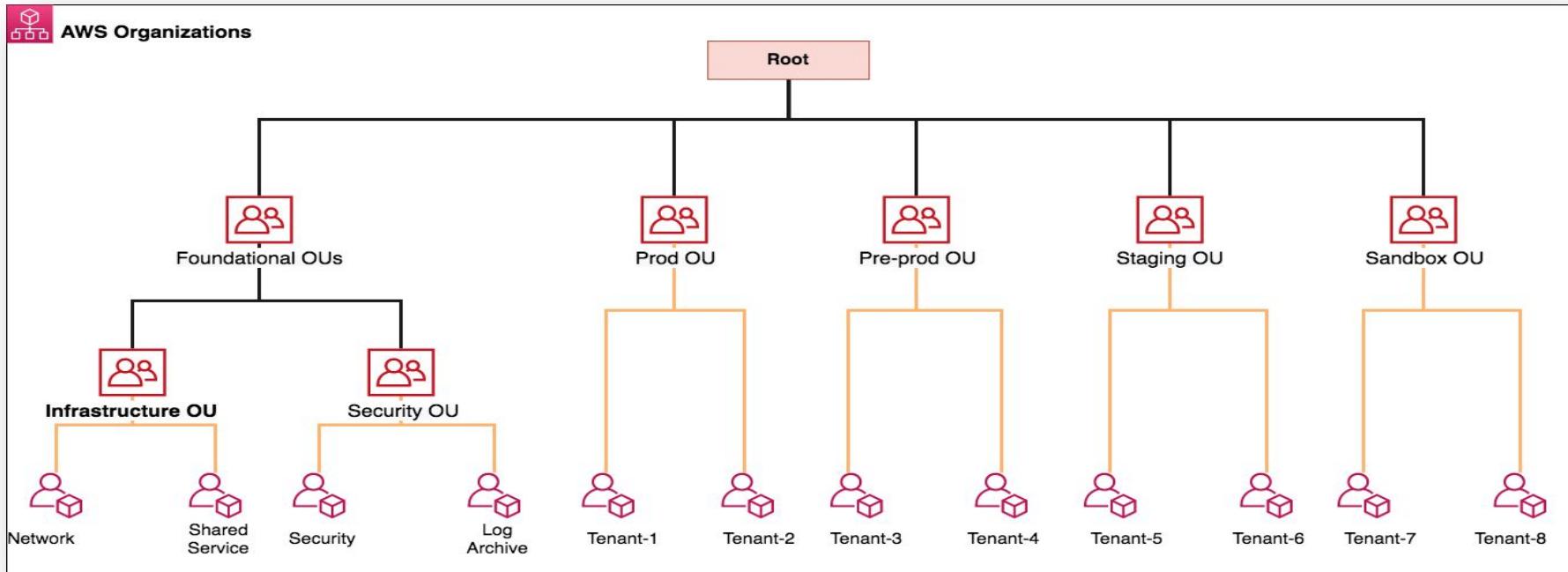
Log Archive Account



# Managing multiple accounts – AWS Organizations



- SCPs (Service Control Policy) can be applied at any node.
- **Master/Management Account** remains **unaffected** by **SCPs**.



# Creating a New AWS Account



- ❑ New AWS account creation process
  - ❑ Independent account creation
  - ❑ Under AWS Organizations
- ❑ Free Tier usage
- ❑ Why are Limits/Quota important?

# IAM users and groups



- Users are tied to actual people in your organization – Bob, Lisa, Raj, etc..
- Groups allow us to give the same permissions to multiple users

# IAM users and groups



- Groups can not contain other groups – only users
- Users can be part of multiple groups, and do not have to be part of a group

# IAM Policies – define permissions



- ❑ Policies are how we define who can do what in AWS
- ❑ Example: Bob can read and delete EC2 instances
- ❑ Policies are JSON documents – see on right
- ❑ Policies can be applied to users and/or groups...

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "ec2:Describe*",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": "elasticloadbalancing:Describe*",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch>ListMetrics",  
                "cloudwatch:GetMetricStatistics",  
                "cloudwatch:Describe"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

A red rectangular box containing three items: two green checkmarks and one red X, likely indicating a comparison between the shown policy and a baseline or best practice.

# IAM Policies – least privilege



- ❑ Generally, in AWS and other cloud providers you should apply the least privilege principle
- ❑ This means that you do not give more permissions than a user actually needs...

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "ec2:Describe*",  
      "Resource": "*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": "elasticloadbalancing:Describe*",  
      "Resource": "*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "cloudwatch>ListMetrics",  
        "cloudwatch:GetMetricStatistics",  
        "cloudwatch:Describe"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```

# IAM Policies – inline and managed



- ❑ There are 2 types of policies: inline and managed
- ❑ Inline Policies – these are written specific to a particular IAM entity.
- ❑ Managed Policies – these can be attached to multiple IAM entities.

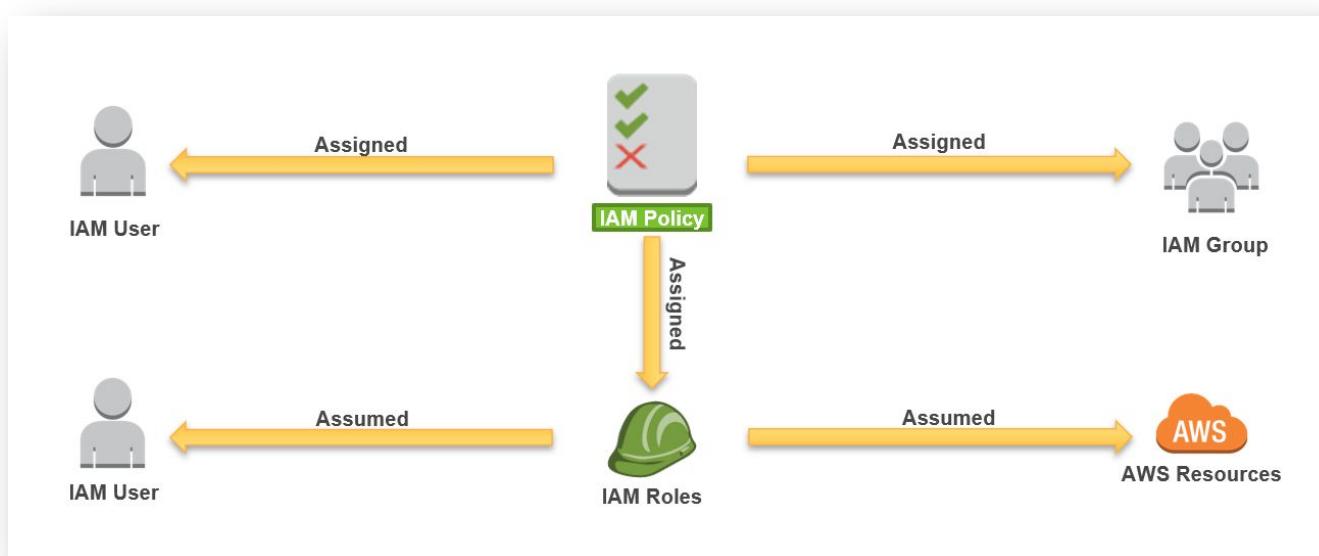
The 2 types:

- ❑ Amazon Managed Policies
- ❑ Customer Managed Policies

# IAM Policies – inline and managed



When two conflicting permissions come together,  
DENY always WINS instead of ALLOW



# IAM Roles and services



- ❑ Why do we need roles? AWS services will often need to perform some actions on your behalf
- ❑ Example: perhaps our EC2 instance needs to list the IAM users. By default, it does not have permissions for this
- ❑ But we can assign our EC2 instance a role and then it will be able to list IAM users – let's see this in a lab!

# IAM Roles and services Lab/Hands On



- ❑ LAB: perhaps an EC2 instance needs to list the iam users. By default, it does not have permissions for this
- ❑ But we can assign an EC2 instance a role and then it will be able to list iam users – let's see this in a lab!

# IAM Roles - more



- Roles are similar to users b/c they are IAM identities with permission *policies*
- *Think of it as "assuming a role"*
- Common use-cases for roles: Lambda instance roles, EC2 instance roles, CloudFormation roles

# IAM Best Practices



- Never** use Root User's access keys. Delete them.
- Create individual IAM users.
- Use groups to assign permissions to IAM users.
- Grant **least privilege** always.
- Configure a **strong** password policy.
- Enable MFA** for all the IAM users.
- Use **roles** for applications that run on Amazon EC2 instances.
- Delegate by using roles** instead of by sharing credentials.
- Rotate** credentials regularly.
- Remove unnecessary/inactive users and credentials.
- Use policy conditions for extra security.

# IAM Labs



- Execute this lab:

**[https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial\\_cross-account-with-roles.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_cross-account-with-roles.html)**

- Execute this lab:

- [https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial\\_managed-policies.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/tutorial_managed-policies.html)**

# IAM Permission Boundaries

# IAM Permission Boundaries – why?



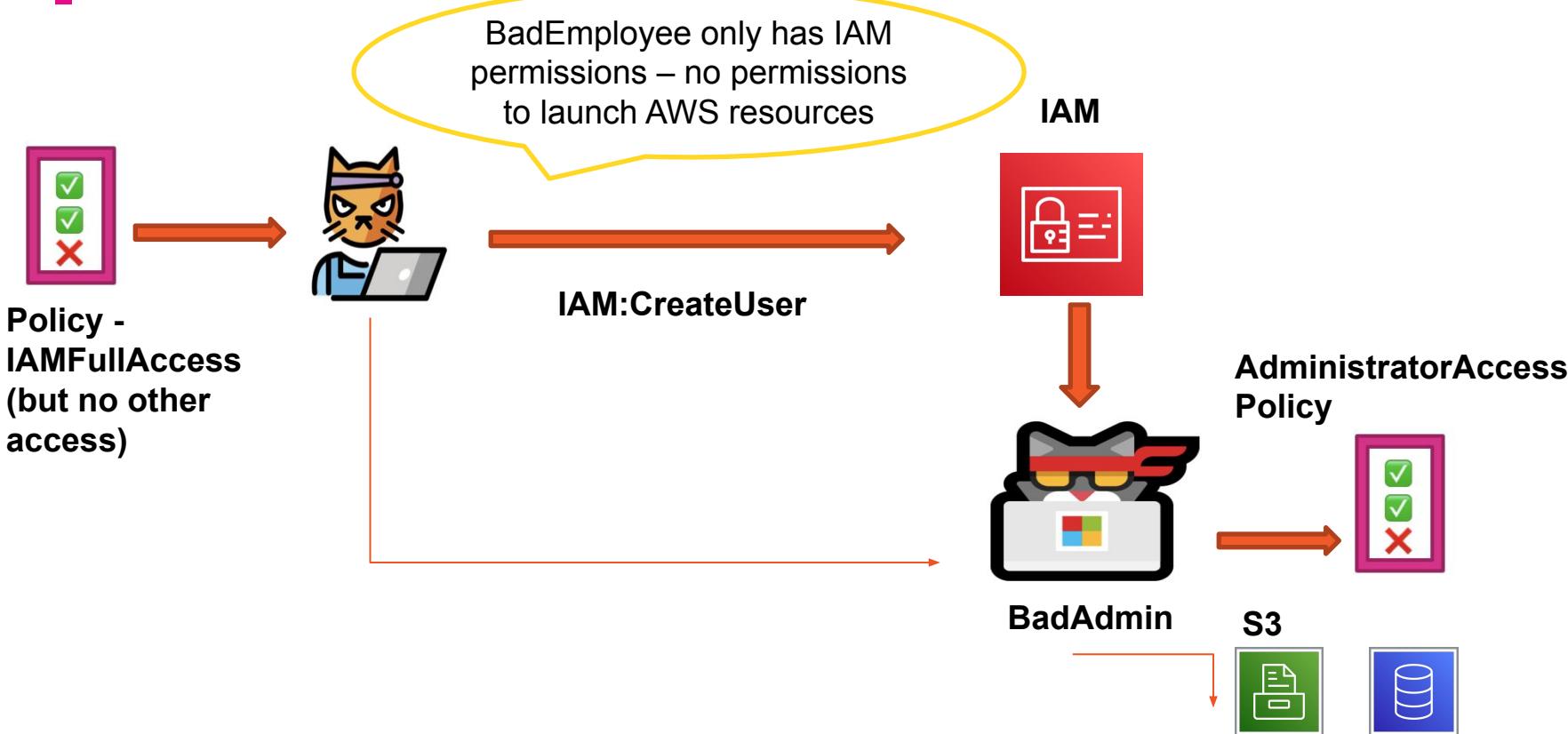
- ❑ To best understand permission boundaries, it helps to first go through a mock scenario
- ❑ This scenario will help us understand why we need them..
- ❑ The scenario is called “Privilege Escalation”

# Privilege Escalation



- ❑ Privilege escalation is a cybersecurity concept – and it's not limited to AWS
- ❑ It's *malicious* – it's when a user tries to give themselves *more* access than they are supposed to have
- ❑ Why? So the user can get access to sensitive resources, do some damage, etc..
- ❑ Note that privilege escalation can even be performed by employees of an org like CG – either knowingly or unknowingly

# Privilege Escalation Example



# Permission Boundary to the rescue!



- So, how do we fix that mock scenario?
- With a permission boundary...
- We use these at CG
- Let's take a look at how that would work...

# Privilege Escalation Prevention



BadEmployee only has IAM permissions – no permissions to launch AWS resources

IAM

Policy -  
IAMFullAccess  
(but no other  
access)



IAM:CreateUser



Permissions Boundary Policy -  
ensures that users created by BadEmployee  
have the same or fewer permissions...



AdministratorAccess  
Policy



BadAdmin

S3



# Permission Boundary Demo/Lab



- ❑ Demo – create a user with the same permissions as our BadEmployee and then login as that user
- ❑ Then, let's create a new user but with AdministratorAccess – this new user can see/access services that they shouldn't be able to
- ❑ Then fix the issue with a Permission Boundary

# Shared Responsibility Model

# Shared responsibility model



- Security and compliance is not just the responsibility of AWS when you (the customer) use their services
- Rather, it's a *shared* responsibility between AWS and you the customer!
- That is what is meant by the shared responsibility model

# Shared responsibility model - example



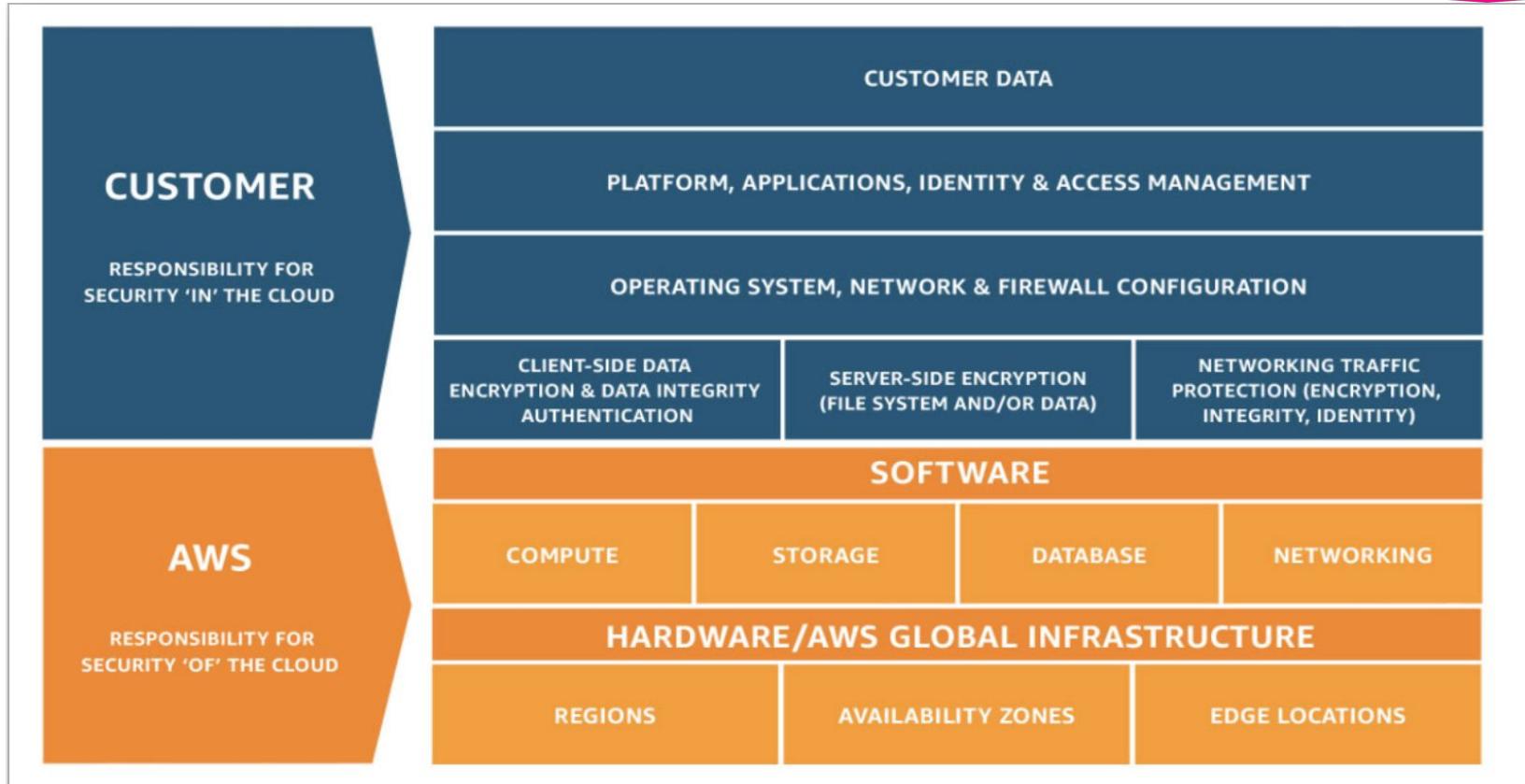
- If you use an EC2 instance you (not AWS) are responsible for installing and managing things like operating system updates + security patches
- EC2 gives you a lot of control, but that comes with more responsibility since it's an IaaS solution

# Shared responsibility model - example



- ❑ Similarly even with other services where AWS has more control and you have less – like S3, a PaaS solution – you still are responsible for a lot
- ❑ Things such as the data, IAM tools to apply permissions, etc..

# Shared responsibility model graphic



# Shared responsibility model – applying it



- Ok great, so what should we do to ensure that we are being responsible customers?
- See here:  
<https://aws.amazon.com/compliance/shared-responsibility-model/>

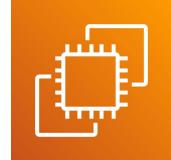
# EC2 – IaaS in AWS

# EC2 Basics – Elastic Cloud Compute



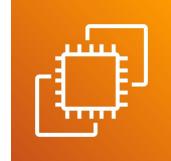
- ❑ Note: we cover EC2 here (briefly) because it is something that you should be aware of – as it's used in a lot of examples within AWS documentation/tutorials
- ❑ But, at *Capital Group*, our trajectory for the cloud is more towards *serverless services* in AWS – so EC2 *is not a solution that you should look to implement*

# EC2 Basics – Elastic Cloud Compute



- ❑ **Operating System (OS):** RHEL, SLES, Ubuntu, Amazon Linux, Windows or Mac OS
- ❑ How much compute power & cores (CPU)
- ❑ How much random-access memory (RAM)
- ❑ How much storage space:
  - ❑ Network-attached (EBS)
  - ❑ Host (Instance Store)
- ❑ **Network card:** speed of the card, Public IP address
- ❑ **Firewall rules:** security group
- ❑ Bootstrap script (runs once during the launch): EC2 User Data

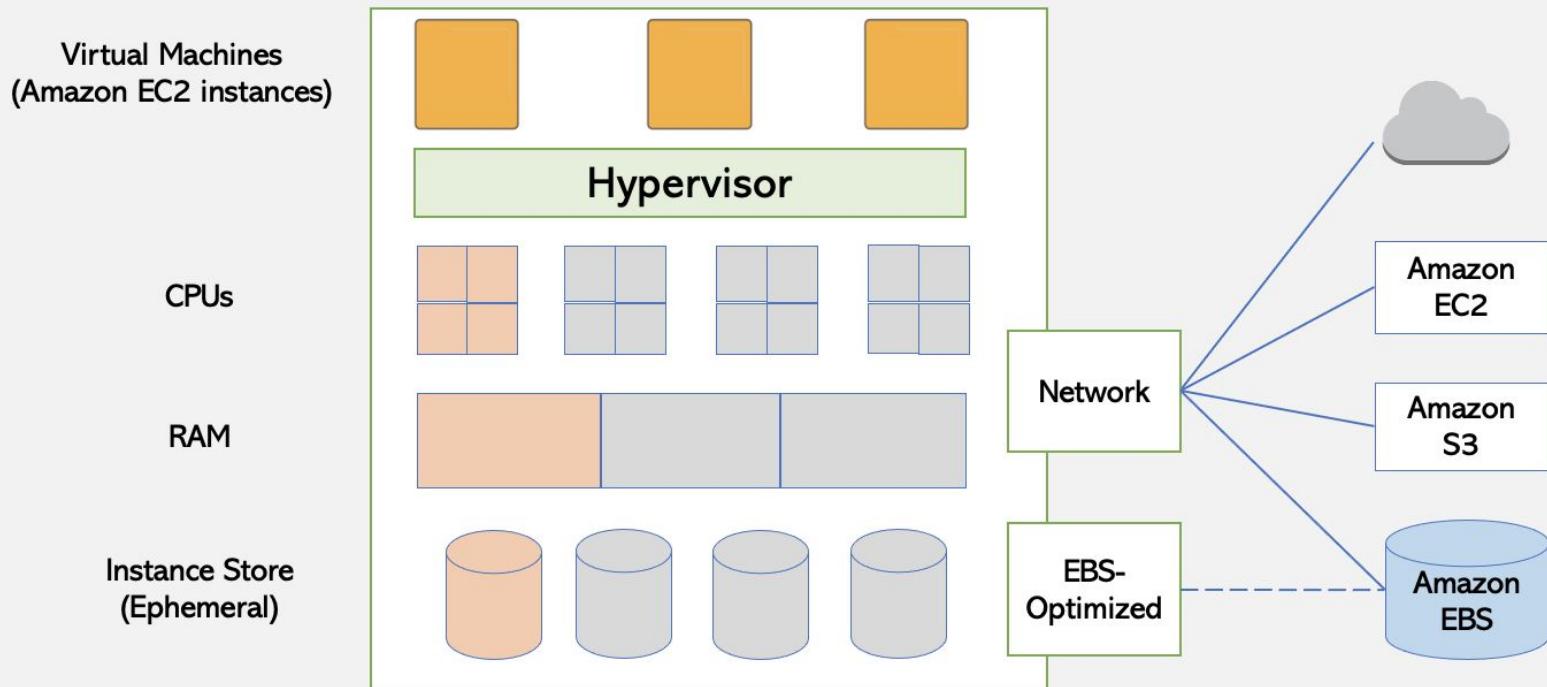
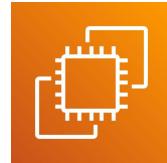
# EC2 Basics



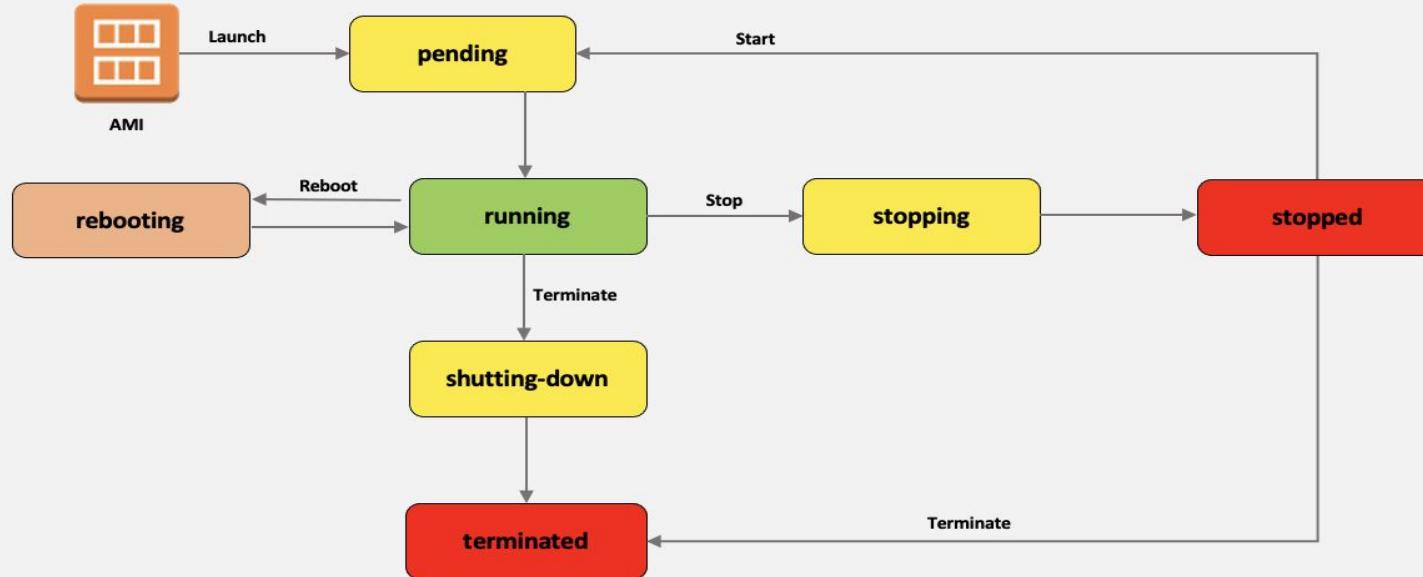
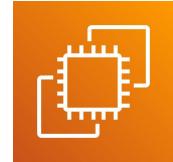
If you use an EC2 instance  
you (not AWS) are  
responsible for installing  
and managing things like  
operating system updates  
+ security patches

EC2 gives you a lot of  
control, but that comes with  
more responsibility since  
it's an IaaS solution

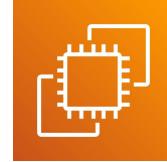
# EC2 Architecture



# EC2 Lifecycle

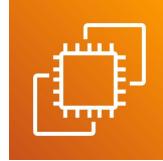


# Reboot vs Stop vs Terminate



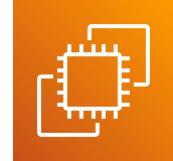
Characteristic	Reboot	Stop/Start (EBS-backed instances only)	Terminate
Host computer	The instance <b>stays on the same host computer.</b>	The instance runs on a <b>new host computer.</b>	
Public IP address	No change	<b>New address assigned</b>	
Elastic IP addresses (EIP)	EIP remains associated with the instance.	EIP remains associated with the instance.	EIP is <b>disassociated</b> from the instance.
Instance store volumes	Preserved	<b>Erased</b>	<b>Erased</b>
EBS volume	Preserved	Preserved	Boot volume is <b>deleted by default.</b>
Billing	Instance billing hour doesn't change.	You <b>stop incurring charges</b> as soon as state is changed to <i>stopping</i> .	You <b>stop incurring charges</b> as soon as state is changed to <i>shutting-down</i> .

# EC2 Pricing



On-Demand	Reserved	Spot Instances
<ul style="list-style-type: none"><li>• No commitment</li><li>• Pay by the hour</li><li>• Any partial hour converted to full</li><li>• A new billing cycle starts whenever an instance changes to “<b>Running</b>” state</li><li>• A billing cycle ends when instance changes to “<b>Stopping</b>” state</li><li>• Billing cycles don’t start at 9am, 10 am etc.</li><li>• Pay per second (supported for few Operating Systems)</li></ul>	<ul style="list-style-type: none"><li>• Two terms available – 1 year or 3 years</li><li>• 3 Payment options:<ul style="list-style-type: none"><li>- Full Upfront</li><li>- Partial Upfront</li><li>- No Upfront (not for 3 years term)</li></ul></li><li>• Lot of saving in comparison to On-Demand</li><li>• Gives you Capacity Guarantee as well</li><li>• You commit the usage for chosen term</li><li>• You can re-sell on AWS if you choose not to use</li><li>• Considered for full term</li></ul>	<ul style="list-style-type: none"><li>• Unused capacity at AWS is given in market for bidding</li><li>• Look at pricing history and decide bid price</li><li>• Instances are terminated with 2 minutes notice when market price goes above bid price</li><li>• If terminated by AWS, last partial hour is free</li><li>• Optionally, use Spot Block option with bid to block the instance (maximum 6 hours)</li></ul>

# EC2 Lab



Execute the lab here:

[https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2\\_GetStarted.html](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html)

<https://github.com/varoonsahgal/cq-cloud-foundations/wiki/EC2-Lab>

# AWS and IP's

# IPv4 vs IPv6

- In networking there are really 2 IPs - IPv4 and IPv6
  - IPv4, most common and what you've probably seen already - 4 #'s separated by 3 dots - 123.152.125.33
    - Allows for 3.7 billion possible addresses in the public space
  - IPv6 is more for IoT - internet of things - it could look like this: 2001:0db8:85a3:0000:0000:8a2e:0370:7334



# Private vs Public IPs

- ❑ No 2 machines can have the same public IP
- ❑ Public IP can be geo-located easily as well...
- ❑ Public IP - machine can be identified on the internet
  - When SSH into EC2 we are using a public IP
- ❑ Private IP:
  - Only exists in a private network
  - ip does have to be private in the private network

# Elastic IP's

- ❑ Remember our EC2 instances? We can easily stop and then start them back up
  - When that happens the EC2 instance can change its public IP address...
- ❑ **Elastic IP:** Perhaps you need a fixed IP address - then that is what's called an elastic IP
  - It's a public IPv4 you own (until you delete it)
  - Generally: avoid using Elastic IP as it reflects poor architectural decisions...
  - Instead, use a random public IP and register a DNS to it
  - With Elastic IP you can mask the failure of an instance by remapping address to another instance in your account
  - BEST OPTION: use a load balancer - that's the best pattern for AWS

# IP address hands on

- ❑ Log into your EC2 instance

# Databases on AWS

# Unmanaged vs Managed Services

## Unmanaged:

*Scaling, fault tolerance, and availability are managed by you.*

Amazon Elastic Compute Cloud (EC2)



## Managed:

*Scaling, fault tolerance, and availability are typically built in to the service.*

Amazon Relational Database Service (RDS)



# Relational and Non-relational Databases

	Relational	Non-Relational
Data Storage	Rows and Columns	Key-Value
Schemas	Fixed	Dynamic
Querying	Using SQL	Focused on collection of documents
Scalability	Vertical	Horizontal

## Relational

ISBN	Title	Author	Format
9182932465265	Cloud Computing Concepts	Wilson, Joe	Paperback
3142536475869	The Database Guru	Gomez, Maria	eBook

## Non-Relational

```
{  
    ISBN: 9182932465265,  
    Title: "Cloud Computing Concepts",  
    Author: "Wilson, Joe",  
    Format: "Paperback"  
}
```

# Different Database types and use cases

## Common data categories and use cases



**Relational**



**Key-value**



**Document**



**In-memory**



**Graph**



**Time-series**



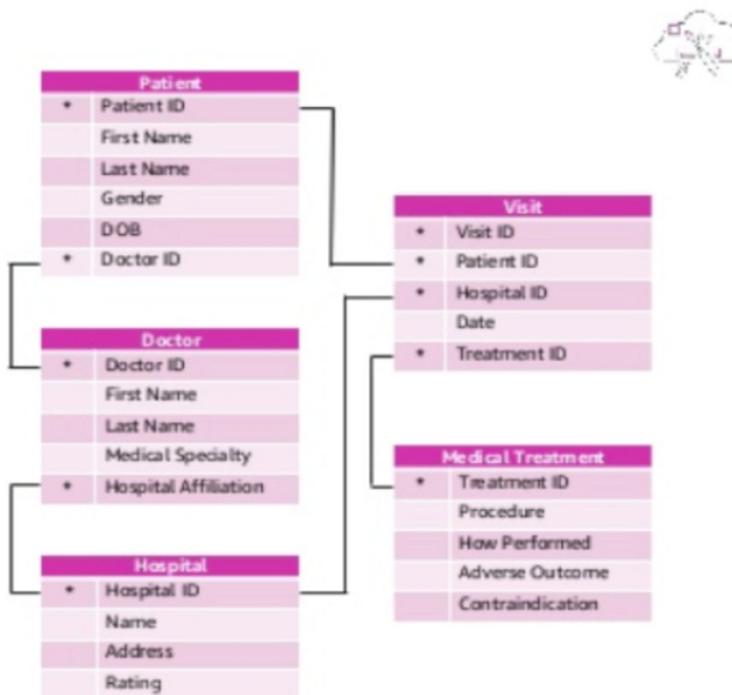
**Ledger**

Referential integrity, ACID transactions, schema-on-write	High throughput, low-latency reads and writes, endless scale	Store documents and quickly access querying on any attribute	Query by key with microsecond latency	Quickly and easily create and navigate relationships between data	Collect, store, and process data sequenced by time	Complete, immutable, and verifiable history of all changes to application data
Lift and shift, ERP, CRM, finance	Real-time bidding, shopping cart, social, product catalog, customer preferences	Content management, personalization, mobile	Leaderboards, real-time analytics, caching	Fraud detection, social networking, recommendation engine	IoT applications, event tracking	Systems of record, supply chain, health care, registrations, financial

# Relational databases

## Relational data

- Divide data among tables
- Highly structured
- Relationships established via keys enforced by the system
- Data accuracy and consistency



# Different Database services

- Amazon RDS, Aurora – Relational
- Amazon DynamoDB – NoSQL
- Amazon ElastiCache – In memory
- AWS Database Migration Service – For migrating data

# Relational databases

- ❑ Self describing - schemas help define relations and constraints between rows and tables in database
- ❑ Amazon DynamoDB – NoSQL
- ❑ Amazon ElastiCache – In memory
- ❑ AWS Database Migration Service – For migrating data

# Different Database services

- ❑ Lets go through some of these one by one...

# AWS RDS

# RDS – Relational Database Service

- ❑ RDS is not a DB of its own – rather it's a service around a database ***that you choose!***
- ❑ RDS gives you 7 databases engines to choose from:
- ❑ Including Amazon Aurora MySQL-Compatible Edition, Amazon Aurora PostgreSQL-Compatible Edition, MySQL, MariaDB, PostgreSQL, Oracle, and Microsoft SQL Server.

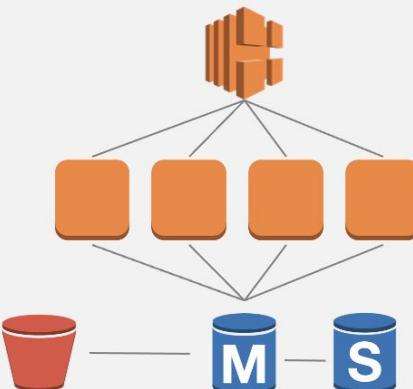
# RDS

- ❑ RDS could be provisioned as Single-AZ or Multi-AZ
- ❑ With Multi-AZ operation, your database is **synchronously** replicated to an instance in another Availability Zone in the same AWS Region.

# RDS

- **Failover** to the standby automatically occurs in case of master database failure. There will be delay for few seconds for sure.
- Planned maintenance is applied first to standby databases.

# Resilient Durable Application Architecture - RDS



Elastic Load Balancing load  
balancer instance

Application, in Amazon  
EC2 instances

Amazon RDS database instances:  
Master and Multi-AZ standby

DB snapshots in  
Amazon S3

# We could just deploy a DB instance on EC2...

- But why would we want to use RDS instead?
- B/C RDS is PaaS (Managed service) and we can offload the database administration. RDS gives us:
  - Dashboards for monitoring
  - OS Patching, Automated provisioning
  - Read replicas

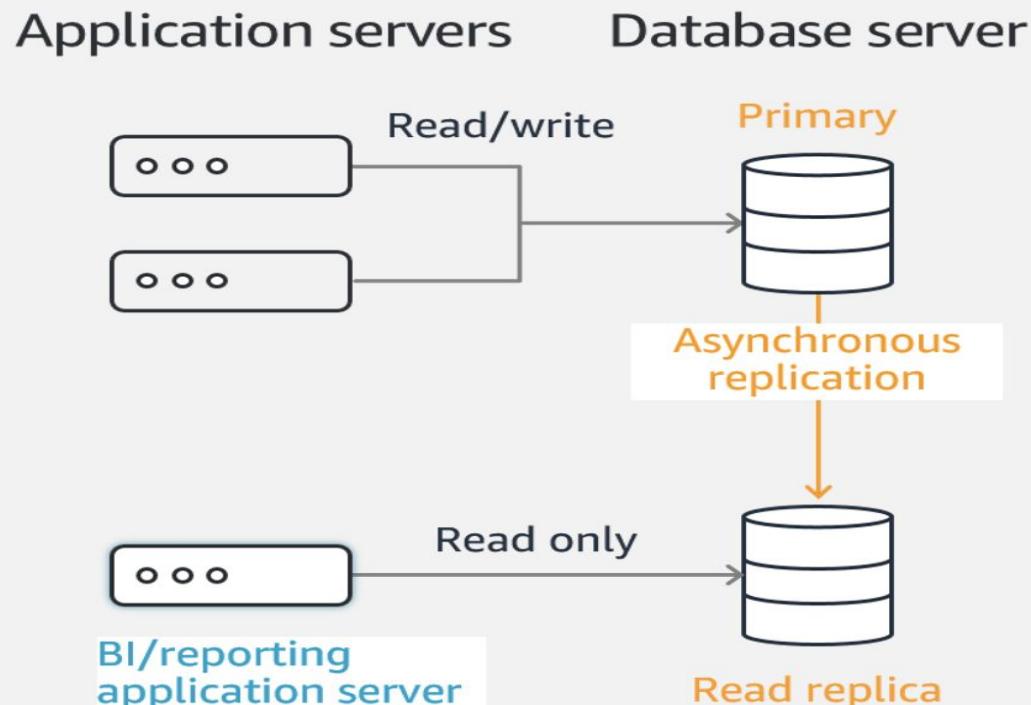
# We could just deploy a DB instance on EC2...

- Disaster recovery via multi-AZ setup
- Scaling – both vertical and horizontal..

# RDS Read Replica

- ❑ Read replica == copy of the primary DB instance
- ❑ Can offload read requests
- ❑ Why use them? b/c they make it easy to scale out and improve DB **read** performance
- ❑ Especially good if your application is read-heavy as opposed to write-heavy

# RDS Read Replica - Diagram



# RDS Read Replica

- ❑ Read replica == copy of the primary DB instance
- ❑ Can offload read requests

# RDS Read Replica – use case

- Why use them? b/c they make it easy to scale out and improve DB **read** performance
- Especially good if your application is read-heavy as opposed to write-heavy
- Great for read-heavy database workloads like reporting applications

# RDS Read Replica == SELECT in SQL

- ❑ Since they are *read* replicas and not write replicas they are used for SELECT kind of statements in SQL
- ❑ As opposed to statements which write to a DB – like DELETE, INSERT, UPDATE, etc..

# RDS Read Replicas – Network Cost

- ❑ When data is transferred from one AZ to another, there is typically a network cost charged by AWS
- ❑ However, when we create RDS Read Replicas within the same region, we don't pay that fee..

# Takeaways

- ❑ RDS is a PaaS offering allowing developers to focus on their applications as opposed to DB administration
- ❑ RDS uses read replicas

# RDS Lab

- [https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT\\_WebAppWithRDS.html](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/TUT_WebAppWithRDS.html)
  
- <https://github.com/varoona-sahgal/cg-cloud-foundations/wiki/RDS-Lab>

# Amazon Aurora

# Aurora walkthrough and demo

- ❑ Demo of this service

# Aurora Labs

- [https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP\\_GettingStartedAurora.CreatingConnecting.Aurora.html](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html)
- [https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP\\_GettingStartedAurora.CreatingConnecting.AuroraPostgreSQL.html](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/CHAP_GettingStartedAurora.CreatingConnecting.AuroraPostgreSQL.html)
- [https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/TUT\\_WebAppWithRDS.html](https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/TUT_WebAppWithRDS.html)

# DynamoDB

# DynamoDB Lab



- ❑ Execute labs here:
- ❑ <https://aws.amazon.com/getting-started/hands-on/create-no-sql-table/>
- ❑ <https://aws.amazon.com/getting-started/hands-on/create-large-nonrelational-database-dynamodb/> (time permitting)
- ❑ <https://github.com/varoonsahgal/cg-cloud-foundations/wiki/Dynamo-DB-Lab>

# DynamoDB



- ❑ Demo + walkthrough of service...

# AWS S3

# AWS S3 – simple storage solution



- ❑ S3 is for storage
- ❑ It's not a database
- ❑ What kind of storage?

# S3 Use Cases



- Backup + Storage
- Static websites
- Disaster Recovery
- Archives

# S3 Case studies



## Customers



NASCAR modernizes multi-PB media archive at speed with Amazon S3 »

[Snap optimizes cost savings while storing 2 exabytes - over 1.5 trillion photos and videos - on Amazon S3 Glacier Instant Retrieval »](#)



Shutterstock transforms IT and saves 60% on storage costs with Amazon S3 »



Runtastic saves €300,000, stays on track for growth using Amazon S3 »

# AWS what is durability?



- ❑ Durability *is not the same as availability*
- ❑ Durability is probability that the object will remain intact and accessible after a period of year
- ❑ 100% durability – no possibility of object being lost
- ❑ 90% - there's a 1 in 10 chance of being lost

# AWS S3 11 9s - durability



- 11 9's is **99.99999999%**
- S3 guarantees 11 9's of durability – so  
**99.99999999%**
- This means that for example, if you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years

# AWS S3 – objects



- In AWS S3, we often use the term *object*
- An object is a file and any metadata that describes that file
- **Buckets** are containers for objects
- To store an object in S3 you create a **bucket** and then upload objects into that bucket..

# AWS S3 – buckets



- *Buckets* are containers for objects
- They must have a globally unique name
- Despite that fact, buckets are defined at the region level
- To store an object in S3 you create a *bucket* and then upload objects into that bucket..

# AWS S3 Bucket Policies



- ❑ S3 buckets have policies as well
- ❑ What could we do with S3 bucket policies?
- ❑ We can: give access to other accounts, give public access to the bucket, and even force objects to be encrypted..

# AWS S3 lab



- ❑ Execute the tutorials available at:
- ❑ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/GetStartedWithS3.html>
- ❑ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/website-hosting-custom-domain-walkthrough.html>
- ❑ <https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html>

# AWS S3 lab takeaways



- ❑ What did we learn from the lab?

# AWS Security: Encryption + KMS

# AWS KMS – Key Management Service



- ❑ What is an encryption key?
- ❑ An encryption key is **a random string of bits created explicitly for scrambling and unscrambling data.**
- ❑ We need these keys to protect sensitive data
- ❑ KMS gives us a place to store these keys and manages our encryption keys for us
- ❑ Analogy – something like a building manager that holds all the keys

# AWS KMS – Key Management Service



- ❑ Often, you will hear the term “encryption” for an AWS service – this is typically a reference to KMS
- ❑ Most AWS services seamlessly integrate with KMS – such as S3, RDS, etc..
- ❑ KMS is fully integrated with IAM
- ❑ We can also use AWS CloudTrail to audit KMS usage (where CloudTrail tracks user activity/API usage)

# AWS KMS – Key Management Service



- ❑ You **never** want to store your secrets in plaintext (human readable) directly in your codebase (typically a github repository)
  - Aside: you never want to store your aws access keys from this class in a github repo - please do not do that!
- ❑ So, what you can do is invoke KMS encryption via API calls instead

# AWS KMS Labs



- ❑ Execute the tutorials here:

<https://docs.aws.amazon.com/kms/latest/developeruide/getting-started.html>

# Serverless

# What is serverless



- ❑ Serverless does not mean that there are no servers
- ❑ Rather, it's a paradigm in which developers no longer have to directly manage servers
- ❑ But servers are still being used in the background – you as a developer just do not see them

# What is serverless



- So what exactly do developers do with serverless?
- Developers deploy code..

# What is serverless



- ❑ Initially serverless just meant AWS Lambda, b/c that's the AWS service that first started using serverless
- ❑ Also, initially Lambdas were == FaaS

# What is serverless



- ❑ But now, serverless encompasses other AWS services...
- ❑ Essentially anything that's a managed service is considered serverless b/c you typically can not log into the service and you never really see the servers...

# AWS - Sample serverless services



- ❑ AWS Lambda
- ❑ Fargate – serverless for ECS where we do not provision our infrastructure to run our Docker containers
- ❑ S3 – storage solution, we don't interact with servers directly
- ❑ More here: <https://aws.amazon.com/serverless/>

# Lambdas – why do we use them?



- ❑ To understand why we use lambdas let's talk about some of the drawbacks of EC2
- ❑ With EC2 == virtual servers in the cloud
- ❑ With EC2 our servers are constantly up and running – they may be incurring costs even if we are not using them
- ❑ The CPU and RAM have limits based on what we defined for the EC2 instances..
- ❑ In order to scale we have to define ASG's, some manual intervention is requisite for this

# Lambdas – why do we use them?



- ❑ Now, let's talk a bit more about lambdas...
- ❑ Lambdas are *virtual functions*, which means that there are no servers to manage...
- ❑ Because lambdas *run on-demand* - only when invoked, this is very different than the EC2 cost model...
- ❑ Lambdas have *short execution* periods – default of 3 seconds, with a max of 15 minutes (you can adjust this value within that range)
- ❑ With Lambdas, *scaling is automated*, if you need more lambda functions, AWS will automatically provision them for you..

# Lambdas case study - Netflix



Statistic	Existing EC2	Enhanced Lambda
Average Time to First Response of New Instance	3 minutes	2006 ms
Average Response Time of Warmed Instance	800 ms	1100 ms
Average Active Instances / Concurrent Executions	~1000 instances	<100 Preprocessor instances <25 Concurrent Executions in Lambda
Average Daily Costs	~\$1000	<\$100

# Lambdas vs EC2 – other considerations



Infrastructure Differences	
EC2	Lambda
<ul style="list-style-type: none"><li>• Developer controls delivery artifacts</li><li>• Traditional instance monitoring tools</li><li>• Os packages can be installed</li><li>• Background processing available</li></ul>	<ul style="list-style-type: none"><li>• Artifacts must be compressed into ZIP file</li><li>• Primarily monitored with AWS Cloudwatch</li><li>• Statically built executables only*</li><li>• Background processing is suspended if request is not being processed</li></ul>

\* AWS Lambda does provide some non configurable preinstalled programs

# Lambdas – pricing model



- ❑ Pricing is easier b/c you pay per request and for the compute time
- ❑ So really 3 factors that determine the cost of an AWS lambda:  
**number of executions** of the lambda, the **memory allocation**,  
and the **average duration/run-time** of the lambda
- ❑ Generally, it's quite cheap to run AWS Lambda so it's very popular for  
that reason
- ❑ See here: <https://dashbird.io/lambda-cost-calculator/>
- ❑ Also see here: <https://aws.amazon.com/lambda/pricing/>

# Lambdas – free tier + GB-seconds?



- Generous **free tier** of one million free requests per month and 400K GB-seconds of compute time per month
- GB second is **the number of seconds your function runs for, multiplied by the amount of RAM memory consumed** – 1 GB-second is 1 GB of memory used for one second.
- If your code uses 1 GB for 1 minute then 2 GB for two minutes, the accumulated memory usage is  $1*60 \text{ seconds} + 2*120 = 300 \text{ GByte seconds}$ .
- If your function uses less RAM – like 128 MB RAM, then you get 3.2 million seconds in the free tier, but you get 400K seconds if function uses 1 GB RAM

# Lambdas – benefits



- ❑ Integration with most AWS services
- ❑ Lambdas can work with many programming languages
- ❑ AWS Lambda natively supports **Java, Go, PowerShell, Node.js, C#, Python, and Ruby** code, and provides a Runtime API which allows you to use any additional programming languages to author your functions.
- ❑ Runtime allows for more programming language support with lambda like C++ and Rust, see here:  
<https://aws.amazon.com/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components/>

# Lambdas – additional benefits



- ❑ With lambda easy to automatically scale and add additional resources
  - up to 10 GB of RAM and 6 vCPU cores, per here:  
<https://aws.amazon.com/about-aws/whats-new/2020/12/aws-lambda-supports-10gb-memory-6-vcpu-cores-lambda-functions/>
- ❑ This is without using ASG's...

# How are lambda functions invoked?



- ❑ Through a variety of ways, we can invoke our lambdas:
- ❑ Directly from the AWS Console and the CLI
- ❑ We can connect lambda functions to AWS API Gateway, which exposes them as REST API calls
- ❑ Lambdas can also be used for back-end processing logic for a lot of AWS services including DynamoDB, SQS, and Kinesis

# AWS Lambdas + other services



- ❑ AWS Lambda integrates with other AWS services to invoke functions or take other actions, some examples:
- ❑ With **S3**, we can invoke a function in response to resource lifecycle events, such as with Amazon Simple Storage Service (Amazon S3).
- ❑ With **API Gateway**, we can respond to incoming HTTP requests
- ❑ We can run a function on schedule with **Amazon EventBridge**

# AWS Lambdas – labs

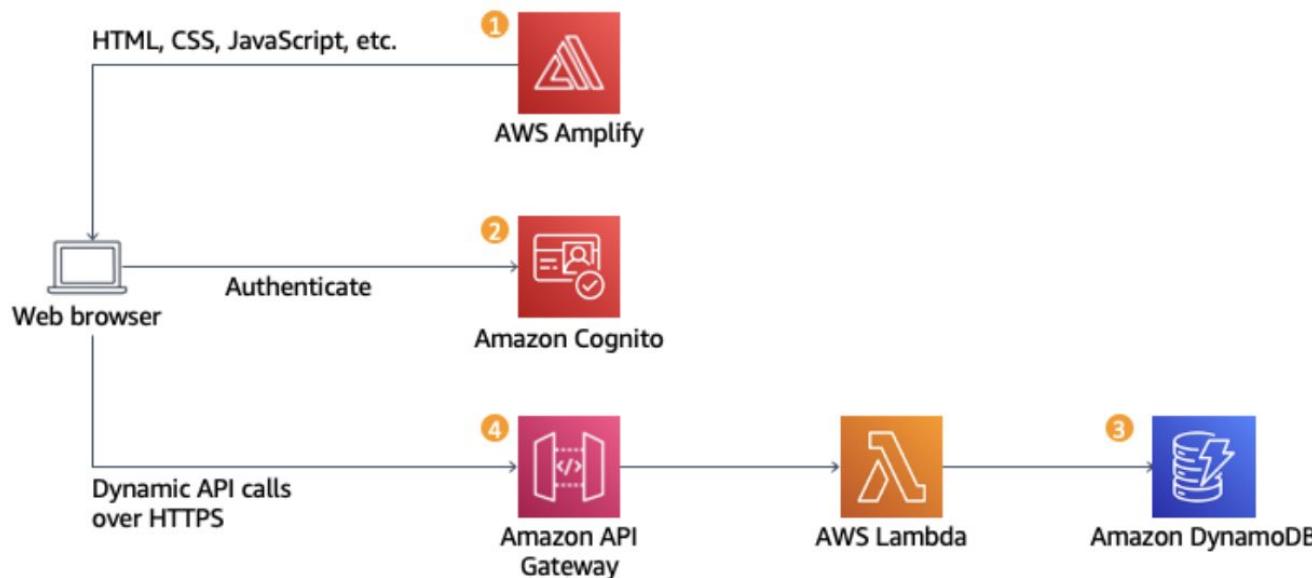


- ❑ Execute the labs available at:
  - ❑ <https://aws.amazon.com/getting-started/hands-on/create-a-serverless-workflow-step-functions-lambda/?ref=gsrchandson&id=updated>
  - ❑ <https://docs.aws.amazon.com/cloud9/latest/user-guide/serverless-apps-toolkit.html>
  - ❑ <https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html>

# Serverless in AWS example



We will build out a fully serverless app in the following lab utilizing these services:

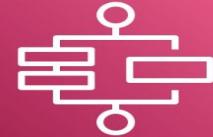


# Lab: Build a serverless web application

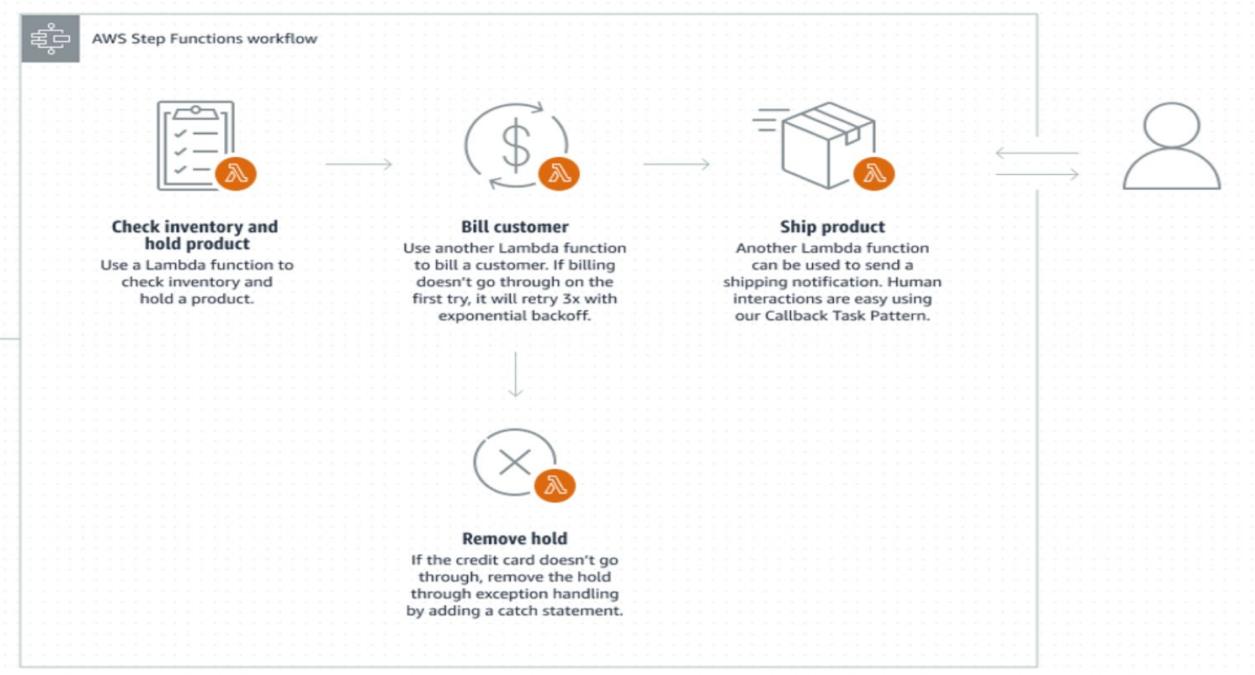


- ❑ Let's build a sample serverless web app – follow this tutorial:  
<https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-db-cognito/>
- ❑ Go ahead and use AWS CodeCommit instead of GitHub, and create an account with ArcGIS...
- ❑ This will take roughly 2 hours
- ❑ Do this in breakout rooms

# AWS Step Functions



 **AWS Step Functions**  
Using the store checkout process as an example, see how you can use AWS Step Functions by invoking Lambda functions in each step of the process.





# AWS Step Functions

- ❑ Build serverless visual workflow to orchestrate your Lambda functions
- ❑ Features: sequence, parallel, conditions, timeouts, error handling,  
..
- ❑ Possibility of implementing human approval feature - ie if human says “yes” then go forward
- ❑ Can integrate with EC2, ECS, On-premise servers, API Gateway, SQS queues, etc...
- ❑ Use cases: order fulfillment, data processing, web applications, any workflow



# AWS Step Functions Pricing

The Step Functions free tier includes **4,000 free state transitions per month**. All charges are metered daily and billed monthly.

## Free Tier

**4,000 STATE TRANSITIONS**

*per month*

*The Step Functions Free Tier does not automatically expire at the end of your 12 month AWS Free Tier term, and is available to both existing and new AWS customers indefinitely.*

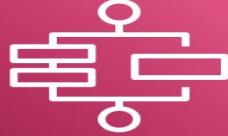
## State Transitions

**\$0.000025 PER STATE TRANSITION THEREAFTER**

*\$0.025 per 1,000 state transitions*

*With AWS Step Functions, you pay for the number state transitions you use per month. You are charged per state transition above the free tier. See the State Transitions Pricing Table for details. [\\_](#)*

*If you include retry error handling in any steps of your workflow, each retry will be charged as an additional state transition.*



# AWS Step Functions Pricing

- ❑ For large scale, it can be quite expensive
- ❑ For low scale projects it can be a feasible solution
- ❑ Note that we are still also paying for the lambda invocations as well (as soon as you are out of the 1 million requests/month and 400K GB-seconds compute time)
  - So there are really 2 charges - 1. the step functions and 2. the lambda invocations

# VPC

© DevelopIntelligence  
A PLURALSIGHT COMPANY

# VPC



- Virtual network; isolated portion of AWS cloud that you design
  - Optional dedicated tenancy
  - Supports logical separation with subnets
  - Fine-grained security
- Private address ranges specified using Classless Inter-Domain Routing (CIDR) notation
- AWS VPCs can use CIDR ranges between **/16 and /28**.
- For every one step a CIDR range increases, the total number of IPs is cut in half:

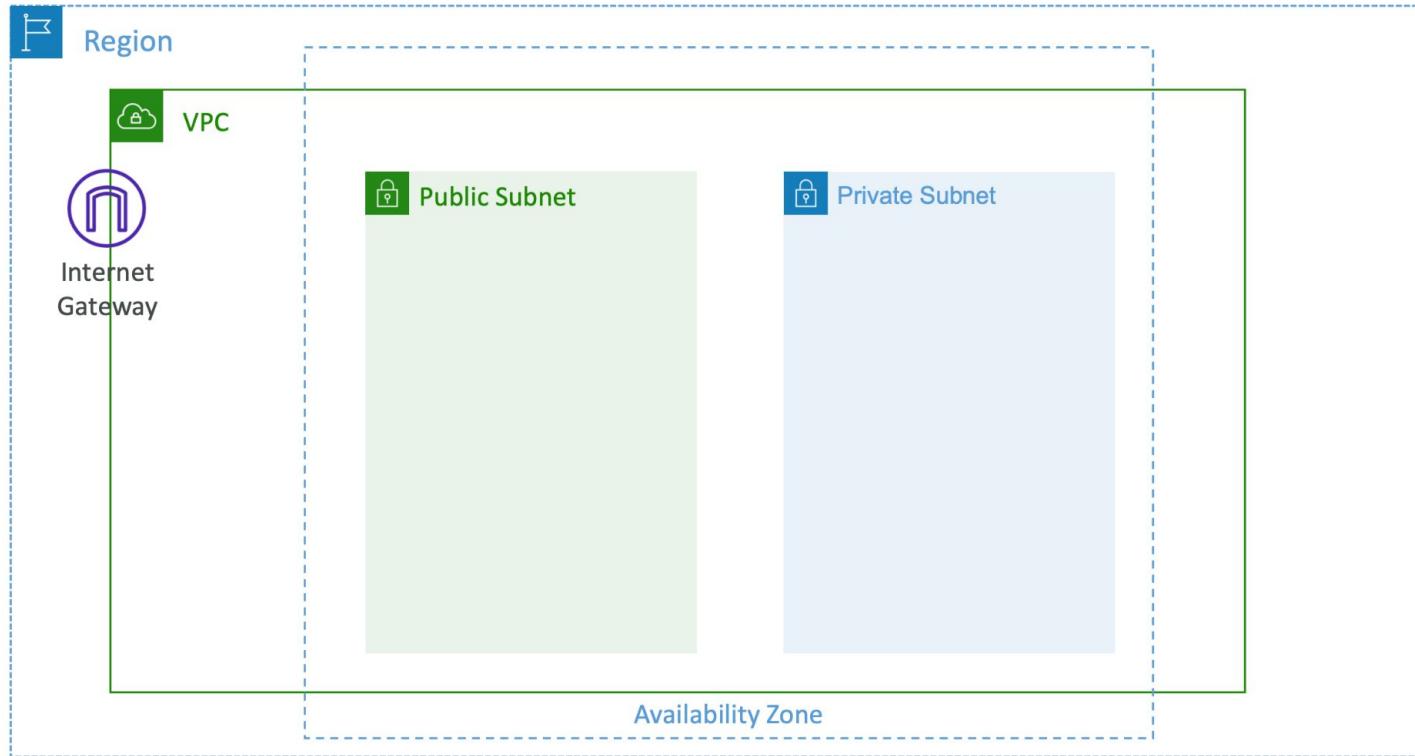
CIDR / Total IPs						
/16	/17	/18	/19	/20	/21	/22
<b>65,536</b>	<b>32,768</b>	<b>16,384</b>	<b>8,192</b>	<b>4,096</b>	<b>2,048</b>	<b>1,024</b>
/23	/24	/25	/26	/27	/28	
<b>512</b>	<b>256</b>	<b>128</b>	<b>64</b>	<b>32</b>	<b>16</b>	

# Internet Gateway



- Allows resources (e.g., EC2 instances) in a VPC connect to the Internet
  - It scales horizontally and is highly available and redundant
  - Must be created separately from a VPC
  - One VPC can only be attached to one IGW and vice versa
- 
- Internet Gateways on their own do not allow Internet access...
  - Route tables must also be edited!

# Adding Internet Gateway



# Subnets

- Subnets segment VPC address ranges even further.
- Subnets can exist within one and only one Availability Zone.
- Subnet CIDR blocks within a VPC must not overlap.
- Subnet inbound and outbound traffic can be restricted using NACLs.
- **Recommendation:** Allocate substantially more IPs for private subnets than for public subnets.



# CIDR

## Understanding CIDR – IPv4

- A CIDR consists of two components
- Base IP
  - Represents an IP contained in the range (XX.XX.XX.XX)
  - Example: 10.0.0.0, 192.168.0.0, ...
- Subnet Mask
  - Defines how many bits can change in the IP
  - Example: /0, /24, /32
  - Can take two forms:
    - /8  $\Leftrightarrow$  255.0.0.0
    - /16  $\Leftrightarrow$  255.255.0.0
    - /24  $\Leftrightarrow$  255.255.255.0
    - /32  $\Leftrightarrow$  255.255.255.255

# CIDR

<https://www.ipaddressguide.com/cidr>

# Cloud Native: microservices

## Definitions

Let's start with 2 definitions:

1. cloud native
2. microservices

## Cloud Native – what does that mean?

What does the word “native” mean to you in a technology context?

Type your answers in chat...

# Cloud Native – what does that mean?

- Story time: when the cloud started to become popular in the mid 2000s (2005 or so) everyone started to throw their applications on the cloud using the lift and shift strategy - hoping to get the cost/performance benefits the cloud promised
  - Key takeaway - lift and shift == migrating to cloud w/ minimal changes to the application before the migration
- But they found they were often getting ***higher costs and worse performance***, so they soon realized that in order to get the benefits of running on the cloud they had to ***first re-architect their applications to run effectively on the cloud***
- And hence the term ***cloud native*** was born - basically it means that if your app is cloud native then it is specifically architected to run on the cloud, and you can therefore get the benefits of ***lower costs/better performance*** that the cloud promises

# Cloud Native – what does that mean?

- Long story short: cloud native == your application has been ***designed/architected to run on the cloud effectively...***
- “Native” used in other contexts like mobile app development
  - - ie, if we develop a mobile app maybe we only develop it for iOS and not Android, in which case we would say it’s “native” to the iOS operating system..
- Basically ***native*** means its targets a specific platform
- Cloud native is an attribute of an application - if your application is cloud native it can then run effectively on the cloud!

## Microservices + Monolith - definition

- To best define microservices, I think it helps to first compare it to the term ***monolith***
- A monolithic application is basically one gigantic application sharing a single codebase
- *What challenges would we face if for example, we were running all of Amazon.com in a single codebase - a single Github repository perhaps?*
  - **Type your answers in chat!**

## Monolith - disadvantages...

- **Deployment** - If we wanted to make any change - even a very small change - to the application we would have to redeploy the entire monolith
- **Shared libraries** - Also, if were to separate amazon.[com](#) into smaller pieces - say a cart and checkout piece, an item recommendation piece, a payment processing piece, etc - then it's safe to say that those pieces of the monolithic application would share libraries, so every time we make a change to one of those libraries, we would have to figure out what piece of the monolith is using that shared library so that we don't break the entire monolith by changing a small part of the library
- **Coding language restrictions** - Another constraint w. monoliths is that if our monolith starts as say, a Java application, then as we add additional pieces to the Java app they would need to be written in Java as well - which means you are limited by decisions made in the past.
- **Overall complexity** - It would also mean that as the complexity of amazon.[com](#) grows, it becomes harder to understand if it's managed as a single entity under a single codebase.

## Monolith - solution...

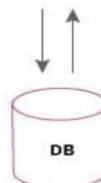
- So, how do we fight the complexity of monoliths?
- Think: how do you eat an elephant?
  - One bite at a time!
- By breaking something down into more manageable pieces - **microservices** - we can better tackle the problem and deal with complexity.
- **Microservices** are the “smaller pieces” of the larger application

## Microservices example

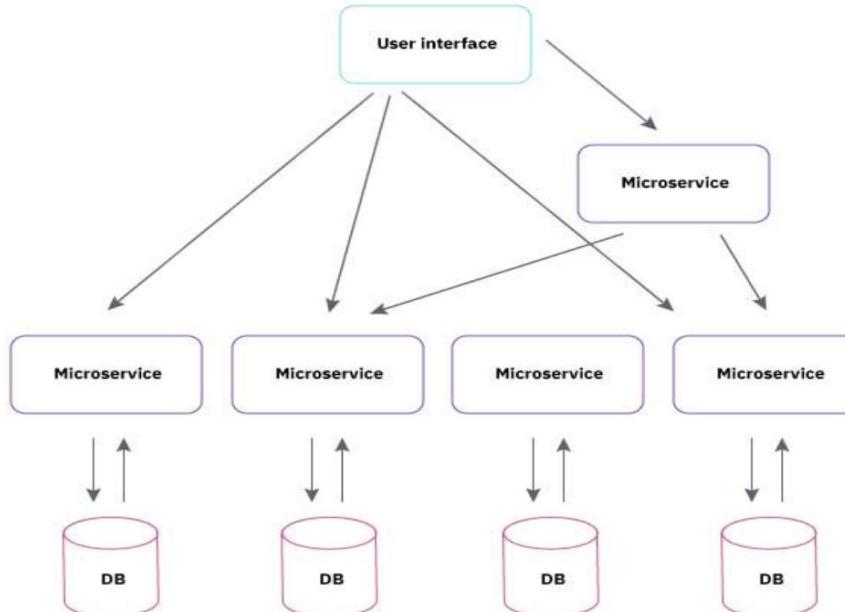
- So, if amazon.com is our “elephant” we can have broken it down into smaller, more manageable pieces
- For instance, search could function as its’ own microservice, the cart would be its’ own microservice, etc.
- Each microservice would be its’ own application, with its’ own separate Github repository
- And all of these microservices combined would give us a fully functional amazon.com site!

# Microservices vs monoliths...

## Monolithic Architecture



## Microservices Architecture



Source: [IBM blogs](#)

## Microservices characteristics

- Small, fully independent and autonomous application. Isolated from other MS's, maybe even with own database.
- Responsible for a logical part of a larger application "ecosystem". Singular responsibility.
- Can be any language. Developers on the same project can have completely different skill sets if necessary.
- Loosely coupled to other MS's. Communicate with other MS's using standardized protocols. Often a REST API or a message bus.
- People generally start with coarse grained services and then break them up into smaller and smaller pieces as the time goes by and they know more and more

## Microservices benefits

- By dividing and conquering w/ microservices, if we want to make a small change to the site, we would just update the micro service that is responsible for that portion of the site
- And the key here is that we would only need to redeploy that particular microservice, without disturbing/interfering with all the other stuff that powers [amazon.com](#).
- This is as opposed to using a monolith where we would have to redeploy all of [amazon.com](#) for a small change which doesn't make sense of course.
- *One of the clear advantages of using microservices is that it allows us to be **more agile***
  - *Agile means we can deploy quickly when changes need to be made, and respond to the needs of the business faster!*

## Microservices - more benefits

- Small and manageable pieces. No massive monoliths.
- No need to have every microservice on the same language and framework.
  - For instance, w/ ecommerce app - your payment services can be in Python, and your user cart can in Java
- Testing is more limited since you don't have to test an entire monolith.
- Can have their own development and release cycle.
- Can be deployed independently without breaking other pieces. Can be done faster.
- Independent scaling of MS's. Faults are isolated. “Graceful degradation”.
- Encourages encapsulation of business logic.
- Smaller size enables Agile development.

## Microservices - challenges

Now that you understand bit more about what a microservice is, what challenges can you think of w/ microservices?

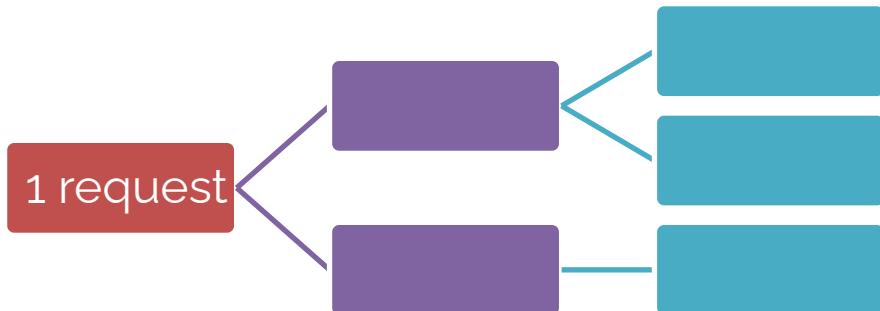
*Type answers in chat...*

## Microservices - challenges

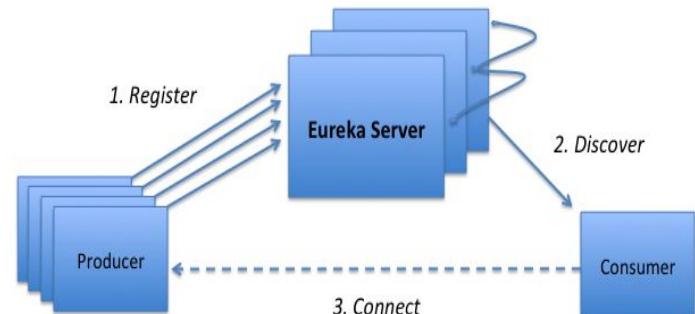
- Microservices come with their own challenges as well
  - Just like monoliths!
- A lot more moving parts. This often requires technologies that might not otherwise be used, such as message buses (like Kafka).
- Requires seasoned architects and leads to keep disaster at bay.
- Often requires more infrastructure to support, as each component has its own overhead.
  - With a monolith, overhead can often be shared.
- Coordination between dependent MS's can be more complex than a monolith.
- More parts to deploy.

# Microservices - challenges

- Service Discovery (need Service Registry)
- Fault tolerance
- Integration Testing
- Requests can fan out



Service Registry:



# Microservices: Discussion

## E-commerce application

## Microservices discussion - breakout rooms

- Think about Amazon.com - where each web widget on the homepage can be backed by hundreds of microservices. Think about microservices for personalization of home page & personalization of search.
- Pick a specific web widget from the homepage - it could be search bar, cart, item recommendations, sign in, etc...
- Now, how you would break this down into different functionalities? What are the microservices supporting those functionalities? How do they communicate? How is data stored for each service?
- Take 10 mins to think about it with your team and have one person from your breakout room share your findings.

## Microservices discussion

Major functionality:

- Product catalog (admin functions, search, recommendations, related products, etc.)
- Shopping cart (and email reminders about items in cart)
- Order processing (payment processing, emails, etc)
- Customer order cancel and refunds
- Fulfillment (send to warehouse, inventory allocation, picking, shipping, backorders, etc.)
- Reporting (internally and to the customer)
- Email marketing
- Social media integration
- Shipment tracking
- Bonus: Digital content! Movies, music, etc.

# Cloud Native

## Making it a reality

## How do I make my application cloud-native?

- ***cloud native ==*** your application has been ***designed/architected to run on the cloud effectively...***
- We've covered **what** cloud native means but now the question is **how** can we make our application cloud native?
  - What specific steps can we take to make an application cloud native?

## Cloud native - characteristics

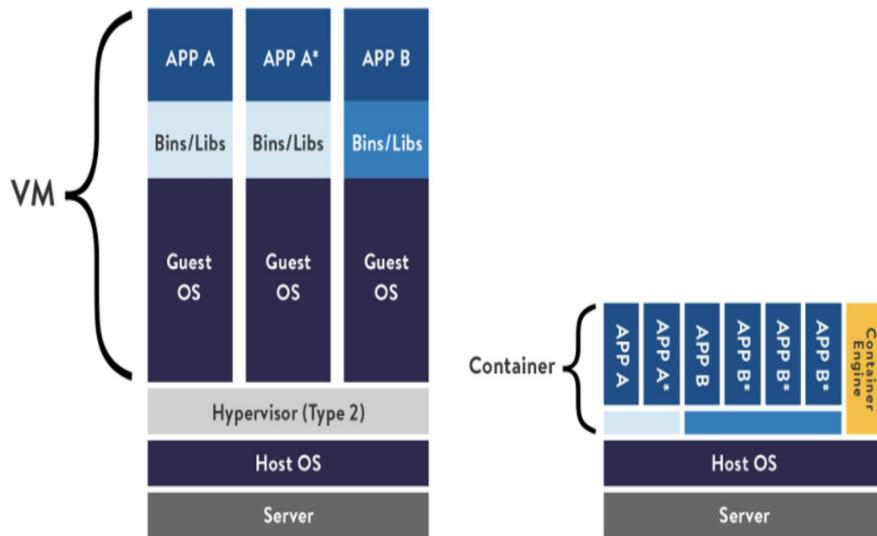
- **The Twelve-Factor App:** Baseline best practices for designing & building web applications for portability and resilience.  
Reference: <https://12factor.net/>
- **Microservices:** Independently deployable services that do one thing well (Single Responsibility Principle).
- **Self-Service Infrastructure:** Platforms for rapid, repeatable, and consistent provisioning of app environments and backing services.
- **API-based Collaboration:** Published and versioned APIs that allow interaction between services.
- **Anti-Fragile:** Systems that are resilient to stress and failure.

# Docker and Kubernetes

# Containers vs VM's revisited

- Both VM's and containers are essentially "wrappers" around our applications making it easier to run our apps on the cloud
- Containers allow us to share a copy of the OS between applications
- Apps running inside a VM each have their own copy of the OS
- Hence, VM's are heavier
- Containers are lightweight, low overhead, and we can run more apps

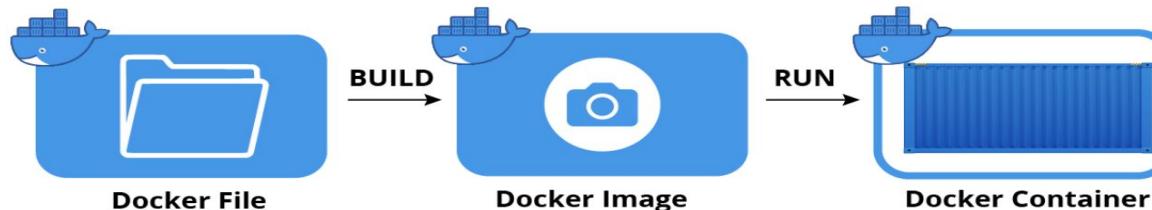
## VM vs Containers



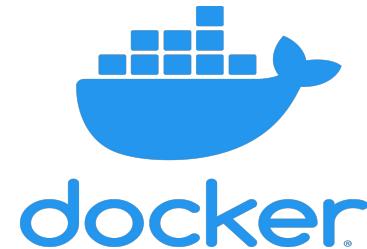
# Docker and K8s – why?

- ❑ To really understand the AWS services that use Docker and K8s we should first try to understand Docker/K8s individually
- ❑ Services such as ECS, EKS,

# Docker terminology

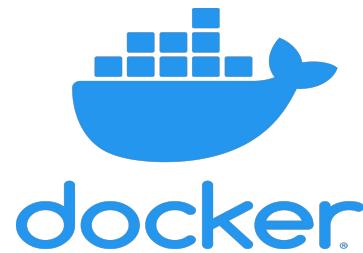


- DockerFile + your application code -> Looper -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- Docker *images* are how we share our application
- An instance of a Docker image is a running container....this is a running instance of your application!
- Don't worry if this is not clear yet, we will be playing with images and containers shortly..



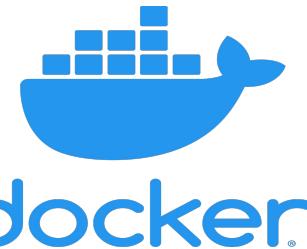
# What problem does Docker solve?

- K8s is container orchestration
- Before we get into K8s – container orchestrator - logical to cover containers first
- Concept of containers not invented by Docker
- But Docker executed the concept very well!
- Helps eliminate the 'works on my machine phenomenon'***
- Simple example: trying to get a Node.js app to work locally is tough – containerizing it makes it run consistently on different machine setups!
- The Docker ecosystem is excellent....



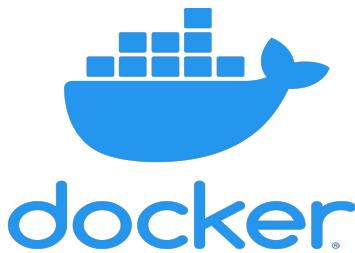
# The Docker ecosystem

- ❑ K8s is container orchestration
- ❑ Before we get into K8s – container orchestrator - logical to cover containers first
- ❑ Concept of containers not invented by Docker
- ❑ But Docker executed the concept very well!
- ❑ ***Helps eliminate the 'works on my machine phenomenon'***
- ❑ Simple example: trying to get a Node.js app to work locally is tough – containerizing it makes it run consistently on different machine setups!
- ❑ The Docker ecosystem is excellent....



# The Docker ecosystem pt 2

- ❑ Docker CLI – easily run images locally, create new images, and manage images
- ❑ Docker Engine – enables containers to run consistently on any infrastructure – various Linux distros + Windows/Server
- ❑ <https://www.docker.com/products/container-runtime>
- ❑ Docker Hub – easily share images with the world



# Docker Registries

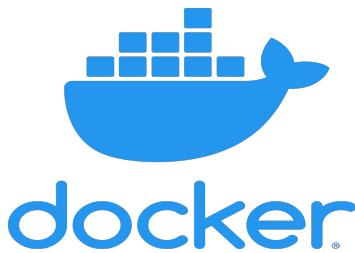
- ❑ We want to share our applications with the world – how do we do that?
  - ❑ With images – which encapsulate our application
- ❑ But where do we put those images?
- ❑ Inside a Docker registry!
- ❑ **Docker registry** is open source software that you can use to set up your own registry
  - ❑ You can see how to set one up here:  
<https://docs.docker.com/registry/>
  - ❑ *Anyone can set up their own Docker registry*
- ❑ Cloud providers also provide their own registries for storing images, like Azure, GCP, etc..



# Docker Repositories

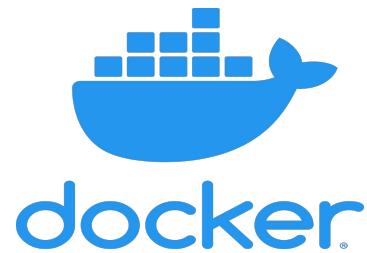
- ❑ Within a registry, you would have many repositories
- ❑ In Docker, a repository is a collection of images of the same application
- ❑ Basically, a repository is a place where multiple versions of a single application live
  - ❑ For example, there is a repository for a Jenkins image:
  - ❑ <https://hub.docker.com/r/jenkins/jenkins>
  - ❑ Repos can be public or private on Docker hub, just create an account..
- ❑ To summarize, smallest to largest: image -> repository -> Registry -> Docker Hub (Registry + Extras)

# Docker images



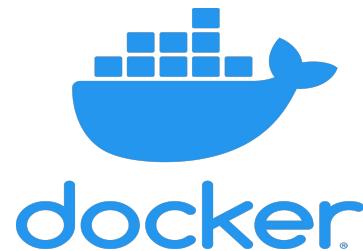
# Docker images – the details

- An image is a template for containers
- Good analogy: Docker image is like a Java class – where a Java class is a template for creating Java objects, Docker images are templates for actual containers



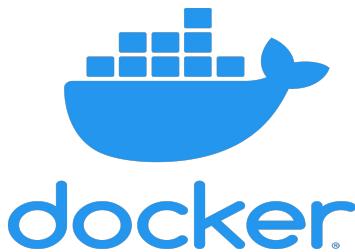
# Docker images – the details

- Images are immutable – can not be changed
- If you want to make some changes to your application or Dockerfile then you would have to create a new image



# Docker images – what's inside?

- Think about any application you've worked on in the past
- What would you need to run that app on a server that's not your local machine?
  - The source code, the dependencies, tools, etc..
  - All of that stuff is inside a Docker image!
- For instance, a React/Node.js application – what's inside that?
  - Our ReactJS files – our frontend
  - Our Node.js runtime – our backend
  - Our Node.js web framework – Fastify.js, or Express
  - Other JS dependencies/libraries – like Babel, Redux, etc..
  - Also the Alpine OS – CentOS, Ubuntu – all share the Linux kernel



# Docker images – layers?

- A Docker image is comprised of many layers internally
- You may also notice this when pulling an image – Docker will create a separate line in your command prompt for each layer it pulls for that particular image
- The layers within an image may look something like this:

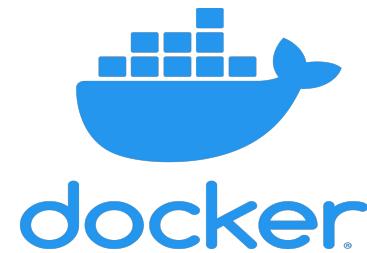
Layer 1: Linux OS: Alpine ~ 50 MB

Layer 2: Security fixes: ~ 2 MB

Layer 3: Node.js runtime: ~ 30 MB

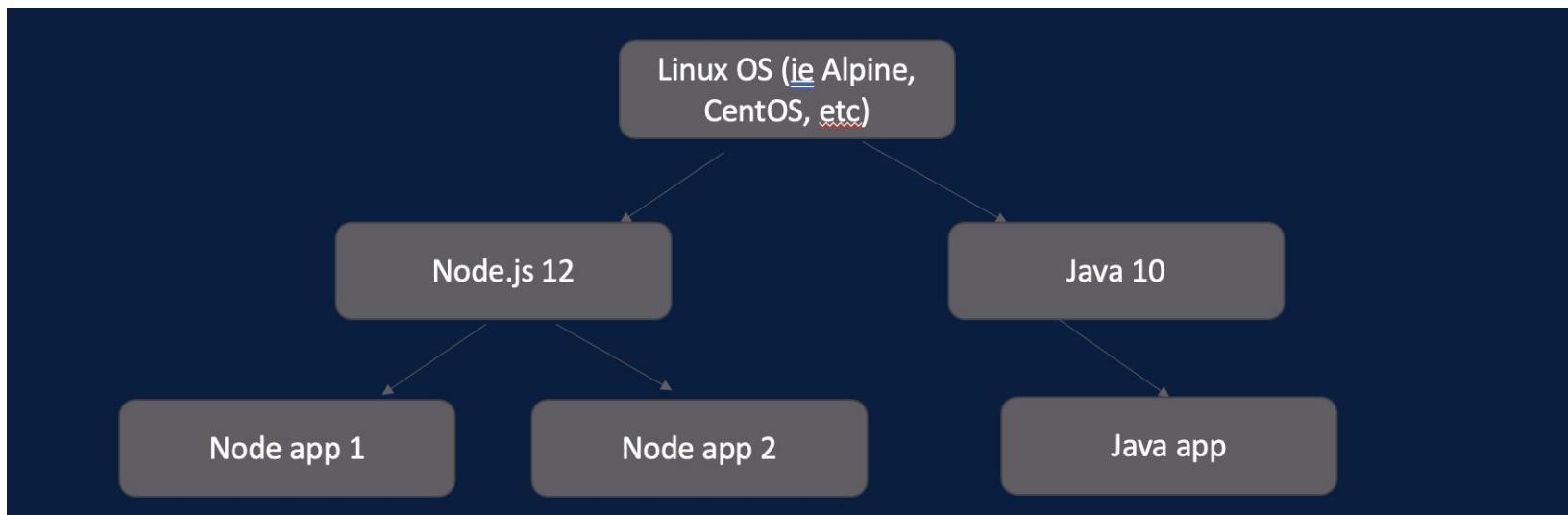
Layer 4: Your Javascript files: ~10 MB

- When pushing/pulling images, only changed layers are pushed/pulled to save on Bandwidth

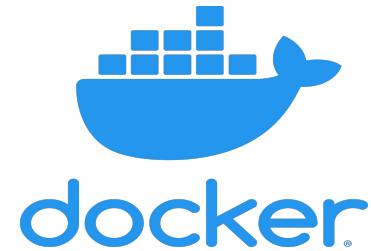


# Docker image inheritance

- Images typically inherit from other images
- Where do we specify which image we want to inherit from? In the Dockerfile! We will see more later...

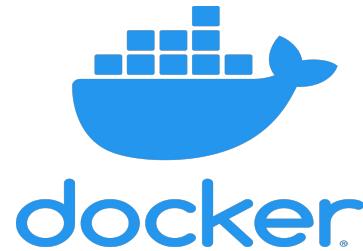


# Docker CLI



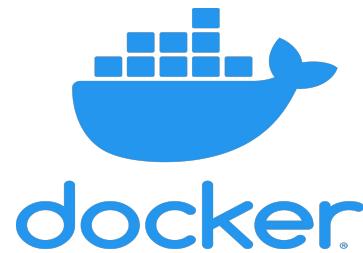
# Docker CLI

- The Docker CLI looks a lot like Linux commands
- All commands start with “docker”
- When working with the Docker CLI, there’s really 2 sets of commands:
- One set for working with *images*
- One set for working with *containers*
- Let’s cover the CLI for images first...



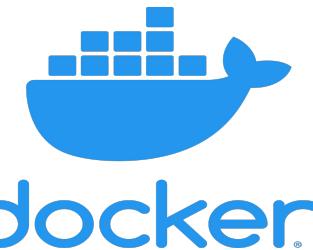
# Docker CLI - images

- ❑ docker build ... – used to build image from Dockerfile
- ❑ docker pull ... – to pull an image from a repo
- ❑ Docker run ... - to run an image
- ❑ -it – interactive and gives you command prompt inside of container
- ❑ -p – port mapping, exposes port from inside container to host
- ❑ -e – environment variable
- ❑ -v – volume mapping
- ❑ docker images – this will show you all of your images
- ❑ docker rmi ... – to delete an image (pass in the image name or id)
- ❑ Can't delete an image that has a running container (a running instance)



# Docker CLI - containers

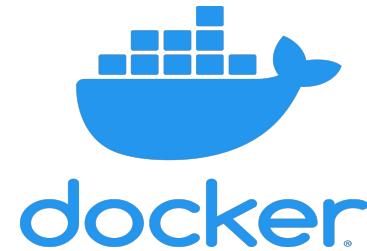
- ❑ docker ps – to list running containers
- ❑ Use –a arg to see stopped containers as well
- ❑ docker inspect ... – to inspect a running container
- ❑ docker logs ... - to view console output of a container
- ❑ docker rm ... – to remove a container
- ❑ docker start/stop/restart ... – self explanatory
- ❑ Difference between **stop** and **rm?**
- ❑ A container that's stopped can be restarted, not true if "rm'd (deleted)
- ❑ If you **stop** a container, you will still see it in **docker ps -a** , but not if you used **rm**



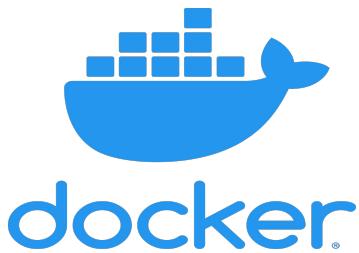
# Docker CLI – port mapping

- ❑ By default, when you run a container with docker run, it doesn't publish any of its ports to the outside world – ie, your localhost when running Docker Desktop
- ❑ Which makes sense because a container is its own isolated environment!
- ❑ So, how do we expose that internal port?
- ❑ Through the '-p' flag (p for publish) when we do a **docker run**
- ❑ So, if we do **docker run -p 8080:80** that says map port 80 in the container to port 8080 on the Docker host – your localhost in this case
- ❑ `docker run -p [host_port]:[container_port] [docker_image]`

# Docker CLI – foreground and detached



- By default, **docker run** will run in the foreground
- Which means that you will see all of the container output!
- Because the console executing docker will attach to standard input/output/error of the container
- **docker run [image\_name]** – this will run in foreground mode
- In detached mode, the container will run in the background:
- **docker run -d [image\_name]**
- Also, interactive mode can give you a shell inside the container, in this case bash:
- **docker run -it [docker\_image] /bin/bash**



## Docker ps (containers)

- To view all running containers you can use “`docker ps`”
- To view all running **plus** stopped containers you can do “`docker ps -a`”
- What is a stopped container?

## Docker CLI - port mapping

- By default, when you run a container with docker run, it doesn't publish any of its ports to the outside world – ie, your localhost when running Docker Desktop
  - Which makes sense because a container is its own isolated environment!
- So, how do we expose that internal port?
  - Through the '-p' flag (p for publish) when we do a **docker run**
  - So, if we do **docker run -p 8080:80** that says map port 80 in the container to port 8080 on the Docker host – your localhost in this case
- `docker run -p [host_port]:[container_port] [docker_image]`

## Docker run command - overriding

- We can override the default command that is run by the image by specifying an extra parameter
- Example: ***docker run busybox***
  - This just starts the container
- But if we do “***docker run busybox ls***”
  - It will start the container and list the contents of the current directory in the container

## Docker run - naming containers

- When creating a container, you can specify a name for the container
- If you don't specify a name, Docker will give the container a name automatically
- Docker has fun with the name when it generates it (if you don't specify one)
  - See Docker Dashboard!
- To specify a name for the container, simply do this:
  - `docker run --name mynginxcontainer`

## Docker run - pulls images

If you try to use docker run with an image that you have not already pulled locally, it will pull it!

If an image doesn't exist on the host yet, it will be pulled before running.

Run is a shortcut operation for:

- Docker pull <image>
- Docker create <image>
- Docker start <container>

example: `docker run -it 27017:27017 mongo:4.0 /bin/bash`

# Docker run - foreground/detached modes

- When running a container it can run in either foreground or detached mode
- Foreground mode is the **default**
  - This is why you get the container output in the terminal:
- Foreground - when running a container console executing docker run attaches to standard input, output and error of the container:

```
[varoona@Varoona-MacBook-Pro ~ % docker run nginx
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
```

In detached mode the container runs in background.

docker run [docker\_image]

docker run -d [docker\_image]

docker run -it [docker\_image] [shell]

example: docker run -it -p 27017:27017 mongo /bin/bash

## Docker run - detached mode

- In detached mode the container runs in background.
- Use the - d flag to specify detached mode:

```
docker run -d [docker_image]
```

- foreground runs by default:

```
docker run [docker_image]  
docker run -d [docker_image]
```

```
docker run -it [docker_image] [shell]
```

```
example: docker run -it -p 27017:27017 mongo /bin/bash
```

## Docker exec command

- **Docker exec** can be used to **execute** any arbitrary process available in a ***running*** container.
- Note that you can NOT use docker exec with an **image**, just a **container** (running instance of an image)
- Example of it's usage: you can use **docker exec** to "log in" to a shell. This is dependent on the image the container was built from.
- For example: the alpine shell is /bin/ash
- This is much like an SSH connection to a server

## Docker exec command

- Once inside the container, you can look around and inspect your container
- This is very useful for debugging.
- Example of why: Suppose you've created an image and you're not sure if the image looks like you expect. So you can get into a running container and view things.
- For example, when we built the nginx image, how did we know the file was there until we tried the web site?
- The location of the shell process depends on the container.
- ***docker exec -it nginx bash***
  - This will open up a shell in the nginx container

## docker run vs docker exec

- Docker exec is NOT meant to start a new container from an image
- For instance, if you try to run **docker exec nginx**
  - You will get an error
  - However, **docker run nginx** works just fine
- Docker exec is meant to execute a process in a running container
  - So it's good for creating a shell inside a container!
- Docker run is meant to run a new container based on an image

# Docker Lab 1



wordpress ☆

Docker Official Images

The WordPress rich content m.

# Docker lab/exercise 1

- ❑ To best understand Docker, it helps to play with it!
- ❑ Let's create our own Docker image by pulling something from Docker Hub
- ❑ Let's try Wordpress: [https://hub.docker.com/\\_/wordpress](https://hub.docker.com/_/wordpress)
- ❑ Read the instructions
- ❑ Your goals:
  - ❑ Run the image locally. Use the docker run command and a port mapping. Remember images must be run to get containers
  - ❑ View the Wordpress UI once the container is up and running
  - ❑ Try creating another Wordpress container while you still have this one running – remember not to use the same port on your localhost!
    - ❑ Also, try to view the new instance of Wordpress in your browser
    - ❑ This should help consolidate that a single image spins up multiple containers – becomes more important w/ K8s!



wordpress ☆

Docker Official Images

The WordPress rich content m.

# Docker lab/exercise 1 part 2

Challenge mode!

1. You'll need a MySQL Docker container to be the DB for WP site
2. Containers are **isolated** by default, so you can put them in a **docker network** to share access

```
docker network create my-wp
```

3. Create the MySQL container and specify some setup like DB name and login info

```
docker run --name wp-mysql -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=wp  
-e MYSQL_USER=user  
-e MYSQL_PASSWORD=admin --net=my-wp -d mysql:latest
```

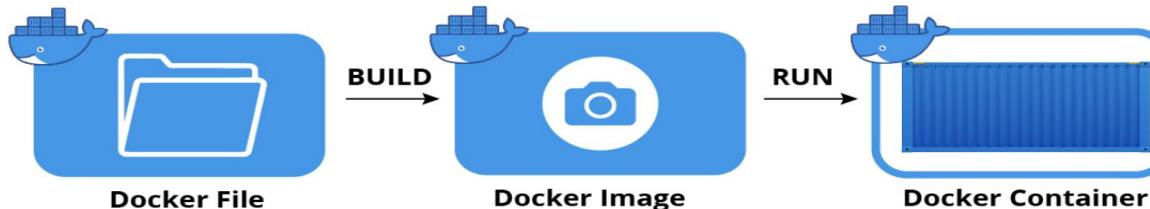
4. Run a WP container that connects to that container:

```
docker run --name wordpress -e WORDPRESS_DB_HOST=wp-mysql:3306 -e  
WORDPRESS_DB_USER=user -e WORDPRESS_DB_PASSWORD=admin -e  
WORDPRESS_DB_NAME=wp --net=my-wp -p 8081:80  
-d wordpress:latest
```

Finish setting up the site and confirm you can make posts!

Remember: the DB data is (currently) only stored in the container, so if you delete the container your data goes away!

# Docker terminology revisited



- DockerFile + your application code -> Jenkins -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- Docker *images* are how we share our application
- An instance of a Docker image is a running container....this is a running instance of your application!
- Don't worry if this is not clear yet, we will be playing with images and containers shortly..

# Docker – tags and Dockerfile

## Docker tags

- In Docker, we can tag our images to give them some metadata
  - Could be a version number, what OS they are meant to run on, etc
- Let's look at an example here for Jenkins tags:  
[https://hub.docker.com/r/jenkins/jenkins/tags?page=1&ordering=last\\_updated](https://hub.docker.com/r/jenkins/jenkins/tags?page=1&ordering=last_updated)
- To tag an image, you can run **docker tag httpd:1.2** as an example of adding a tag of 1.2
  - After the colon, we specify the tag
- Every image will have one tag by default if none is specified

## Docker tags pt 2

- What if we don't specify a tag for an image?
- By default, if you omit a tag, Docker will give it a tag name of **latest**
  - But **latest** is misleading, because it doesn't always mean that's the latest version of the image
  - Example - developer created an image, didn't tag it, so it's given the "default" tag, which is "**latest**"
  - then developer created a newer image and tagged it 1.2 – the newer one has a tag of 1.2, but **latest** will actually refer to an older image!

# Dockerfile

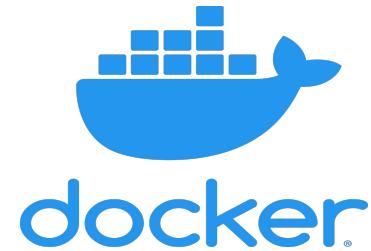
- Remember that **Dockerfile + your app source code -> Jenkins (runs docker build) == Docker Image**
- Dockerfile is how to create Docker images
- A Dockerfile is an example of Infrastructure as code
  - *Infrastructure as code (IaC) means to manage your IT infrastructure using configuration files.*
  - Before IaC, IT admins would have to **manually change configurations in order to manage their infrastructure**

## Dockerfile - more

- The commands in a Dockerfile execute from top to bottom
- And each command is also independent of other commands
- Commands are relative to the directory in which the Dockerfile lives

## Dockerfile - more

- The commands in a Dockerfile execute from top to bottom
- And each command is also independent of other commands
- Commands are relative to the directory in which the Dockerfile lives



# Dockerfile – how we create images

- Where do Docker images even come from?
- A Dockerfile is how we create Docker images

## Dockerfile build command

How do we create a new Docker image?

- By building a Dockerfile!
- **docker build -t <imagename:tag> <directory>**
  - "t" is to specify a tag

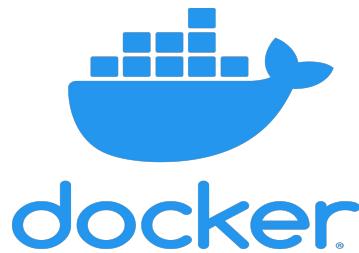
**Example:**

- **docker build -t myimage:v1 .**
- Where **<directory>** is wherever your Dockerfile lives
- Inside the **Dockerfile**, there is a reference to your app code
- We will be using this command shortly...

# Dockerfile commands

A sampling of commands you may see in a Dockerfile:

- FROM – which image you want to inherit from
- RUN – Run a Linux command in a new layer. The command is dependent on which shell is in the image.
- EXPOSE – Tells what ports this container will listen on. Does not actually publish a port, but is rather like documentation.
- COPY – copy files into the container.
- ADD – copy files into the container (same as COPY, but supports 2 other sources - additional URL source, tar extraction from local)
- ENV – Define environment variable
- ENTRYPOINT – The command to be run when container is started. This is inherited from a base image.
- CMD – provides defaults to be run after the ENTRYPOINT. Also inherited from the base image



# Dockerfile == Infra as Code

- A Dockerfile is an example of ***Infrastructure as code***
  - *Infrastructure as code (IaC) means to manage your IT infrastructure using configuration files.*
  - Before IaC, IT admins would have to ***manually change configurations in order to update/manage their infrastructure***
  - For instance, often IT admins would log in to remote machines, make updates through some commands (ie - sudo update apache, or whatever), and the problem was that there was no running history of changes
  - With IaC we can commit things like Dockerfiles to our Git repositories thereby giving us a history of our infrastructure
  - Along with the ability to restore old states

# Docker Exercise

## - Dockerfile

## Dockerfile lab

- Visit [https://hub.docker.com/\\_/httpd](https://hub.docker.com/_/httpd) and see instructions on how to use this image
- Let's grab the Docker image for Apache – a popular web server:
- Docker pull httpd
  - By default this will pull the latest image
- Then run Apache locally:
  - docker run -p 8080:80 httpd
- Then visit <http://localhost:8080> to see a page that says “It works”

## Docker lab 2 - Apache success

**It works!**

# Docker lab 2 part 2 steps

Let's take these steps:

1. Create a file named **Dockerfile**

You can do this anywhere really, don't technically need a repo – best to create a folder

2. Inherit from the base Apache image we just pulled – simply use "**FROM httpd:2.4**"

3. Add a file named test.html so that we can give Apache something to actually serve up instead of the "it works!" text. The **test.html** would be our "application code" in a real application.

You can put this in the **Dockerfile**:

```
COPY ./test.html /usr/local/apache2/htdocs/
```

So, no need for the public\_html folder as mentioned in the docs here:  
[https://hub.docker.com/\\_/httpd](https://hub.docker.com/_/httpd)

4. Now, build it! Use the **docker build** command:

```
docker build -t <imagename:tag> <directory> .
```

**What happens when you leave out the tag?**

5. Now that you've built the image – run it with **docker run -p [host\_port]:[container\_port] [docker\_image]**

Try exposing it on a different port – the instructions say 8080, but try 3500 or something else.  
If you visit localhost:3500/test.html you should see your page!

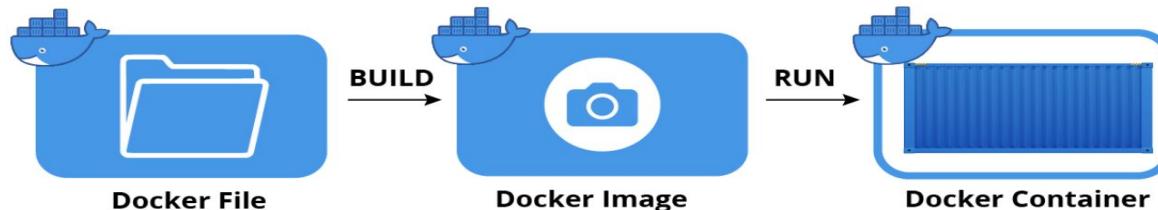
## Docker lab 2 part 3 steps

6 . Let's now see how to bring up a bash shell so we can see what's going on inside the container: **docker run -it myfirstimage /bin/bash** or **docker run -it myfirstimage /bin/sh** (if bash not part of image)

Often, you will want to step inside your container to see what's going on. This is how you would do that!

7. Go inside the **/usr/local/apache2/htdocs** directory and you should see the test.html file that you just copied over!

# Docker terminology revisited



- DockerFile + your application code -> Looper -> Docker Image -> Docker container
- DockerFile will live in your application's repository
- Your source code + DockerFile will give us a Docker image
- In KITT, profiles can essentially function as Dockerfiles
- Docker **images** are how we share our application
- An instance of a Docker image is a running container....this is a running instance of your application!
- Don't worry if this is not clear yet, we will be playing with images and containers shortly..

# Docker Networks

## Lab: Docker container isolation

Docker containers are isolated by default.

Validate this by doing a very simple exercise:

Open a terminal window and then do:

```
docker run -it busybox sh  
touch helloworld
```

Then, open a new terminal window and create another container w/ the same commands:

```
docker run -it busybox sh  
ls
```

What do you see as the output of “ls”? Do you see the helloworld file we created?

## Think:

Suppose I have an application - like Wordpress.

The application needs a database to store information about the blog - specifically it needs and uses the MySQL database to store blog posts.

Wordpress will often need to write to the MySQL database - so it needs to communicate with MySQL

So, if I have a Wordpress container and a separate MySQL container - how can those containers communicate in Docker??

Answer: ***Networks!***

## Linking - Legacy feature

- Networks are the way to go, but you should know a bit about linking....
- This is a *legacy* feature, but still useful to know.
- Networks (next) are the preferred way of communicating.
- You can explicitly link containers like wordpress and mysql to each other so that they can communicate by name.
- You link containers by including a --link option in the run command and provide the container name.
- Linking does not scale well - try linking 10 services with each other!

# Networks in Docker

- You can create Docker networks to allow containers to easily communicate.
- Types:
  - Bridge (default) – this one is created by Docker and containers go here by default. Not recommended for production. No container communication by name.
  - Bridge (user-defined) – Recommended for standalone containers running on the same host. Containers can communicate by name.
  - Overlay – allows networking across Docker hosts. Works with Swarm.
- Networks are managed with the docker network ... commands
- docker network will show all commands, example **docker network ls** will display all existing networks including 3 pre-defined networks.
- Containers are put into a network with the --network option in the docker run command.
- Existing containers can be put into a network with **docker network connect <network name> <container name>**

## LAB - Docker Network

Containers are isolated by default, so you can put them in a docker network to share access

Let's connect Wordpress to MySQL by using a ***user-defined*** Docker network!

## LAB - Docker Network

1. You'll need a MySQL Docker container to be the DB for WP site
2. Containers are isolated by default, so you can put them in a docker network to share access - **docker network create my-wp**
3. Create the MySQL container and specify some setup like DB name and login info: **docker run --name wp-mysql -e MYSQL\_ROOT\_PASSWORD=root -e MYSQL\_DATABASE=wp -e MYSQL\_USER=user -e MYSQL\_PASSWORD=admin --net=my-wp -d mysql:latest**
4. Run a WP container that connects to the mysql container: **docker run --name wordpress -e WORDPRESS\_DB\_HOST=wp-mysql:3306 -e WORDPRESS\_DB\_USER=user -e WORDPRESS\_DB\_PASSWORD=admin -e WORDPRESS\_DB\_NAME=wp --net=my-wp -p 8011:80 -d wordpress:5**
5. Finish setting up the site and confirm you can make posts!
6. Remember: the DB data is (currently) only stored in the container, so if you delete the container your data goes away!

# Advanced Containers - Docker Compose

# Docker Compose

In the last lab, we saw how to create a docker network manually. Imagine if we had more than 2 containers - the command could get quite long!

Is there a simpler,cleaner way to create a network? Answer: **yes, with Docker Compose**

- Compose is a lightweight way to run your application without the overhead of a Swarm or Kubernetes application.
- Compose is a tool for running multiple related containers together with one command. It is included with Docker by default.
- So, you don't have to start each container separately
- You specify all the configurations in **docker-compose.yml** file.
  - This is another example of **infrastructure-as-code!**

## Docker Compose

- compose creates a new user defined bridge (just like we did manually in last lab) network for your containers
- This way they can communicate with each other by name. Or you can specify your own.
- You can refer to an existing image in Docker Hub, on your local machine, a private repo, or you can refer to a Dockerfile location, and ***Compose will build the image for you before running the container.***
- To start up your app, use **docker-compose up**.
- To stop it, use **docker-compose down**.
- It will automatically look at the **docker-compose.yml** file

## docker-compose down

- this command **will stop running containers**, but it also ***removes the stopped containers as well as any networks*** that were created.

# Docker Compose - sample docker-compose.yml file

```
version: '3.3'  
services:  
  wordpressapp:  
    image: "wordpress:2.0"  
    restart: always  
    ports:  
      - "8089:80"  
  mysqlDb:  
    image: "db:2.0"  
    restart: always  
    ports:  
      - "3306:3306"  
    environment:  
      - MY_ENV=my_value
```

Compose will do 3 major things:

1. create a network so our containers can communicate
2. create the image (by building the wordpressapp Dockerfile), and then run that image to create the container
3. create the container from the mysqlDb image

# Lab: Docker Compose

## LAB - Docker Compose

- Create a new Docker Compose file
- Use two Docker Hub images. One for Wordpress, and the other a MySQL image.
  - Review the parameters we passed in when we ran the command manually in the last lab
  - You'll need to review the documentation for the images to find out what important options to pass the containers.

Bring the stack up using **docker-compose up** and try to use Wordpress blog

# Advanced Containers - exec, it

## Interactive Shell

- Containers are mostly intended to run as headless without interaction.
  - headless == no interface, not bash shell, etc..
  - Sometimes however you may want to interact and provide input
- To interact, you can run attached or attach later to interact with it.  
**docker run -it <image-name>**  
**docker attach <container-name>**
- You can now get it's output (like with docker logs), provide input if necessary.
- If you attach at run then to detach without stopping it, use the detach sequence (hold CTRL, press p then q), CTRL+ C will end the container process.
- We will see how to 'login' in the container next, and how to install software

## Exec command

- With a **running container**, you can execute any arbitrary process available on the container using **docker exec**
- Frequently you can use this to “log in” to a shell. This is dependent on the image the container was built from. example: /bin/sh or just bash
- This is much like SSH connection to a server
- Once inside, you can look around and inspect your container
- This is very useful for debugging. You’ve built an image and you’re not sure if the image looks like you expect. So you can get into a running container and view things.
  - We saw how to do this earlier with docker run -it <image name> /bin/sh
  - differences between run and exec in next slide
- The location of the shell process depends on the container.
- **docker exec -it <container name> bash**

## Docker run vs docker exec

- docker run:
  - runs an image to create a container
  - you can specify you want to create a shell to connect to the running container by doing docker run <imagename> /bin/sh
- docker exec:
  - can NOT use docker exec with an ***image***
  - has to be a RUNNING container
    - can't be a stopped/paused container
  - Can connect a shell to a ***running*** container:
    - so if you do “docker ps” you can find the container name/ID, then do a “docker exec <container name> /bin/sh”
- To summarize: docker exec is to be used with running containers, docker run is to be used with images (and it also pulls images if they don't exist locally)

## LAB - docker exec

- Start a new container from the nginx image on Docker Hub
  - Use docker run but don't specify the -it param
- Using exec, get into the container and have a look around

```
docker exec -it <container name> bash
```
- Visit the web site and make sure you see the default home page.
- Now go change the home page index.html file located in /usr/share/nginx/html/
  - Wait, how do you edit the file?
  - Try installing vim. This is a Debian OS, so use **apt-get install vim**. First update the package manager with **apt-get update**
    - apt – advanced package tool – which is used to install/remove software on various Linux distros
    - Now update the headline
- Verify that your changes show in the web site.
- Let's discuss "How will it impact the image?"
- Let's discuss "How do I persist the changes?"

# Registries

## Docker registries

- We want to share our applications with the world – how do we do that?
  - With images – which encapsulate our application
- But where do we put those images?
- Inside a Docker registry!
- **Docker registry** is open source software that you can use to set up your own registry
  - You can see how to set one up here:  
<https://docs.docker.com/registry/>
  - *Anyone can set up their own Docker registry*
- Cloud providers also provide their own registries for storing images, like Azure, GCP, etc..



## Docker hub

- **Docker Hub** is a specific type of registry from Docker w/ additional features like teams, orgs, etc...
- Within Docker Hub you have many repositories, which each represent a distinct application!
  - There's repositories for Jenkins, Apache, nginx, etc – basically any application that wants to share its “Dockerized” binary – it's image.



## Docker repositories

- Within a registry, you would have many repositories
- In Docker, a repository is a collection of images of the same application
- Basically, a repository is a place where multiple versions of a single application live
  - For example, there is a repository for a Jenkins image:
  - <https://hub.docker.com/r/jenkins/jenkins>
  - Repos can be public or private on Docker hub, just create an account..
- To summarize, smallest to largest: image -> repository -> Registry -> Docker Hub (Registry + Extras)



## Docker terminology summarized

- To summarize, smallest to largest: image -> repository -> Registry -> Docker Hub (Registry + Extras)



## LAB - private registry

Let's simulate having a private registry by starting one on our local workstation and then work with it. It runs as a Docker container!

1. Find the registry image on Docker Hub. Review the instructions and run it on your machine.
2. Now try tagging an image and pushing it to your new private registry. Perhaps you can pull Apache/nginx/busybox from Docker Hub and retag it? Try it.
3. Access the REST API to look at the catalog of images

[http://localhost:5000/v2/\\_catalog](http://localhost:5000/v2/_catalog)

Note that there's more involved with setting up a production grade registry, such as authentication. If you're working with a secured registry, you just need to login with docker login.

# Docker - security: containers and root

## Running Docker containers as root

- By default containers are run as root in Docker
- Everything we've done so far has been with the root user
- To verify:
  - Run whoami inside a container - try apache:
    - docker run -it httpd /bin/bash

```
varoona@Varoona-MacBook-Pro-2 lib % docker run -it httpd /bin/bash
root@6d8a4096c73d:/usr/local/apache2#
```

```
[root@6d8a4096c73d:/usr/local/apache2# whoami
root
```

## What about images that we created?

- Even Docker images you create are run as root by default
- For instance, the image we created earlier in the class
  - Let's run it: docker run -it myveryownimage:v1 /bin/bash

```
varoona.sahgal@Varoona-MacBook-Pro-2 lib % docker run -it myveryownimage:v1 /bin/bash
root@bb07c76c5879:/usr/local/apache2#
```

## Fun fact - shell prompt signals current user info

- Also notice the “#” at the end of the shell prompt
  - That’s another signal that our shell is running as root
- If we see a “\$” instead then that means our shell is running as a regular user
- This is a Unix/Linux tradition

```
varoona.sahgal@Varoona-MacBook-Pro-2 lib % docker run -it myveryownimage:v1 /bin/bash
root@bb07c76c5879:/usr/local/apache2#
```

## What's the problem with ROOT?

- We often need to run as ROOT user to install software
- But, the issue is that leaving your containers running as ROOT leaves you vulnerable to attack from hackers
  - Larger attack surface area...
- But i thought containers are isolated from the host (the server on which they are running)?
  - Yes and no - hackers are always trying to figure out how to break the isolation between containers and host
- So, if container is running as ROOT, may give hacker access to the host machine or K8s node if running K8s
- It's not so much the damage they can do **inside** the container, but rather on the **host machine/K8s node**

## Fixing the ROOT problem

- So, once we are done doing ROOT activities like installing software, we should change the default user used by our container
- How and where do we do this?
- In our Dockerfile:

```
_ 3 4
3   RUN useradd --uid 10000 runner
4   USER 10000|
```

## Fixing the ROOT problem

- First line creates a new user **without** ROOT permissions
  - Gives that user a name of “runner”
- The second line tells Dockerfile to switch to that newly created user

```
_ 3 4
3   RUN useradd --uid 10000 runner
4   USER 10000|
```

## Fixing the ROOT problem - what's a UID

```
3 RUN useradd --uid 10000 runner
4 USER 10000|
```

- A **UID (user identifier)** is a number assigned by Linux to each user on the system.
- This number is used to identify the user to the system and to determine which system resources the user can access.
- UID 0 (zero) is reserved for the root.
- UIDs 1–99 are reserved for other predefined accounts.
- UID 100–999 are reserved by system for administrative and system accounts/groups.
- UID 1000–10000 are occupied by applications account.
- UID 10000+ are used for user accounts.

## Software that already runs as ROOT

- You might be using other Docker images that run as ROOT
- To prevent that, you can derive from the base image from that software - so you can do a FROM thatbaseimage
  - and then add the non-privileged user account in Dockerfile

# Docker security: Lab

## LAB - from ROOT to non-privileged user

- Earlier we created our own Docker image (from Apache/httpd) that runs as root
- Now, we want to modify it to run as a non-privileged user
- Go ahead and add the lines below to the end of your Dockerfile
- Rebuild another image (using docker build), name it “mysecureimage:v1”
- Then, bring up a container with a shell using “it” mode and you should see the shell has a “\$” instead of a “#”
  - That tells you its NOT running as ROOT anymore
- And of course it no longer says “root”, but rather “runner”

```
1  #!/bin/bash -e
2  set -x
3  RUN useradd --uid 10000 runner
4  USER 10000|
```

# Advanced Images

## Images - creating from containers

- There's two ways to create a Docker image:
  - Dockerfile
  - create images from running container
    - This is like going backwards! Normally: Dockerfile -> image -> container, but for now: running container -> image
- We can actually create Docker images from containers (**without** using Dockerfiles), but can you think of a reason why we would **not** want to do this?
- Type answers in chat...

## Images - creating from containers

- The primary reason not to do it this way is b/c we have then moved away from infrastructure as code (IaC)
  - b/c the Dockerfile is our infra as code - it can be version controlled and it tells us what base images we are using, what their respective versions are, what modifications have been made, etc..
  - when we create an image from a container, we can't look at the Dockerfile to tell us exactly what changes were made for that particular image
- However, doing this can be useful sometimes...

## Images - creating from containers (not Dockerfile)

- This can be useful for small experiments and proof of concepts

## Steps to create an image from a container...

The general procedure roughly looks like (docker run does both of these):

**docker create ...** (to create a new container)

**docker start ...** (this starts the container you just created)

**NOTE THAT DOCKER RUN DOES BOTH CREATE AND START**

*Make changes in the container, install software, etc.*

**docker commit ...** (this creates a new image without a tag)

- This command is new to us
- How we create an image from a container

# LAB - create an image from a container...

Let's create an image from a container. We'll start with a base image from Docker Hub. Then customize it, and finally create an image from that.

1. First, create a new container using the Nginx image
  - o docker run -it nginx bash
    - this will create a new container and also create a bash shell for you
2. Make some changes to the home page
  - o The homepage is at /usr/share/nginx/html/index.html
  - o How do you edit the file?
  - o Try installing vim. This is a Debian OS, so use apt-get install vim. First update the package manager with  
apt-get update
  - o Now update the headline
3. Then, create a new image from that container. You can name it "mynginx" AND give it a tag of v1
  - Use the **docker commit** command - read the docs:  
<https://docs.docker.com/engine/reference/commandline/commit/>
4. Now, delete the old container.
5. Finally, create a new running container from your new image (mynginx:v1) and verify that it serves the home page you updated - which lives at /usr/share/nginx/html/index.html

# Advanced Images multi-stage builds

## Multi-stage builds

- We want to keep the size of our images down
  - Because if we have maybe 1,000 instances of our container, keeping size small as possible matters even more!
- Multi-stage builds help us do that..
- How does it help?
  - Because it makes it easier to get rid of any unnecessary build artifacts that may be created during the build process
    - sometimes intermediate artifacts like Java/GO/etc SDK and other things are leftover in the build
    - We do NOT want that stuff leftover in our final image, its not necessary and takes up space

## Multi-stage builds - stages

- A Dockerfile is multi-stage if you see multiple FROM statements:
- This example below has 2 distinct stages
- The second stage references the first stage by using **COPY --from=0**
  - **0 is the first stage**
  - this says just grab what i need from the 0th (first) stage, then put it inside an Alpine OS

```
# syntax=docker/dockerfile:1
FROM golang:1.16
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app ./
CMD ["../app"]
```

## Multi-stage builds - naming stages

- We can instead name stages using “AS”
- Easier to read, and makes more sense especially if the instructions in your Dockerfile are re-ordered later, the COPY doesn’t break:
- reference:<https://docs.docker.com/develop/develop-images/multistage-build/>

```
# syntax=docker/dockerfile:1
FROM golang:1.16 AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app ./
CMD ["./app"]
```

## Images - ALPINE

- A goal of image creation should be reduction of image size.
- Some images can be very large. Many images in Docker Hub exceed 1gb.
- Additionally, many standard distros contain security vulnerabilities. If you remove all unnecessary components, you reduce security flaws.
- Many Docker users choose to use Alpine Linux, which is an extremely stripped down Linux distribution. You get the bare bones Linux.
- Alpine is only 4-5 mb in size! Additionally, the runtime footprint is less.
- Due to the small size, this means deploying to a server is very fast with the small download size.
- Alpine is built to be secure. This does not however mean you should just assume it's good enough. Security teams should still scan it for vulnerabilities.
- You'll need to install any necessary software in your Dockerfile. You don't even get curl.

## Images - other small images

- Newer images now exist for small images that bring the same benefits of Alpine, but in other possibly more friendly Linux distros. They offer a bit more balance of size and functionality than Alpine.
  - Ubuntu Minimal 18.04 is 29 mb
  - Debian Slim is around 88 mb

# Docker Volumes

## Docker Volumes: the why

- Suppose we have Jenkins running on Docker containers in production
- If our Jenkins containers come and go, how would the Jenkins data be persisted?
  - ie - data like our build jobs, our configurations in Jenkins
    - How does that data get saved if the containers running Jenkins can be deleted/restarted/etc. ?

## Why do we need Docker Volumes?

- Containers come and go, but what if we want data to persist?
  - Enter - Volumes!
- A volume can be shared with multiple containers
- When we delete a container, that does not delete the volume

## Docker Volumes: when to use

- When the Docker host is not guaranteed to have a given directory or file structure.
  - Volumes help you decouple the configuration of the Docker host from the container runtime.
- When you want to store your container's data on a remote host or a cloud provider, rather than locally.
- When you need to back up, restore, or migrate data from one Docker host to another, volumes are a better choice.

## The 3 types of Docker volumes

- Host/Binding
- Anonymous
- Named

## Bind mount versus volumes

- You may hear the term bind mounts
  - How do they compare to volumes?
- Volumes are managed by Docker
- Volumes - remember they put stuff inside the Linux VM that Docker creates
- But bind mounts put stuff inside the actual host machine - NOT the Linux VM
  - in other words, if you are running Docker Desktop on a Mac, a bind mount will create a folder on your Mac OS - so you can actually see the bind mount in Finder (same for Windows)

## Bind mounts - only good if consistent host environments

- Remember that Docker containers are meant to run in different environments
  - So, if you want to use a bind mount, best if you can guarantee the file structure
- imagine using a bind mount that points to folder1/xyz, which exists on one host machine but not on another
  - for that reason, bind mounts may not be the best solution
- Bind mount - creates a bind/connection between **host machine** and container **NOT** between the Linux VM created by Docker and the container
- <https://docs.docker.com/storage/>

## Docker Volume vs Bind Mount Syntax

- NOTE **ro** means read only volume
- backslash is to allow for new lines
- **VOLUMES:**

```
docker run \
--name=nginxtest \
-v nginx-vol:/usr/share/nginx/html \
nginx:latest
```

- **BIND MOUNTS:**

```
docker run --name devtest \
-v "$(pwd)"/target:/app \
nginx:latest
```

## Docker Volume vs bind mount Syntax

- The difference in syntax you can see is that with a bind mount we provide a path on our host machine - instead of name of a volume

## Docker Volume Syntax

- Can use the -v or - - mount flag
  - - - mount is more explicit/verbose
  - We will use the -v option
- We will use v flag to create volumes
- Can create a Volume outside context of container, ie:
  - **docker volume create myvolume**
- Docs here: <https://docs.docker.com/storage/volumes/>

## Where are Docker Volumes stored?

- Remember that Docker Desktop creates a Linux Virtual machine
  - this is true whether you are on Windows or a Mac
- Volumes are stored inside that Linux Virtual Machine NOT on the “host”
  - in your case, if you’re on a Mac that is your “host” - the macOS
  - remember macOS is NOT Linux, and obviously neither is Windows
- So, **Docker is entirely responsible for managing the volume**

# Docker Volumes Lab

## LAB - Jenkins and sharing data via volume

- First, setup a volume and attach it to a Jenkins image
- Make some changes in the Jenkins image...
- Then, create another Jenkins container instance and attach it to the same volume
- Be sure to change the ports, but keep the volume exactly the same
- In the second Jenkins instance, you should see the same changes you made in the first
  - This is because they share a volume
- You can also store volumes on cloud providers/remote hosts...

## LAB PART 2 - Jenkins and sharing data via bind mount

- Now, do the same thing except with a bind mount instead of a volume
- Notice that a bind mount will automatically create a folder if one does not exist

## LAB PART 3 - replace with Docker Compose

- Finally, writing out these long commands is a lot of work
- Let's use Docker compose to get our Jenkins containers to share a volume
- Create 2 Jenkins containers in your docker compose file
- Have them share the same volume. Change stuff in one container and see those changes reflected in the other container
- Then, try using a .env file to store the path/name of the Docker volume
  - You may need to change Docker compose version to 3.4 or above
  - And grab that value from inside the Docker compose file
  - The “.env” file has to be in the same folder as the compose file
  - See : <https://docs.docker.com/compose/environment-variables/>
- Reminder: This is what attaching a volume to Jenkins looks like on a command line:
- ***docker run --name MyJenkinsContainer11 -v myVolume1:/var/jenkins\_home -p 9092:8080 -p 40001:5000 jenkins/jenkins***

# Kubernetes

# What is Kubernetes?



- ❑ Container orchestration
- ❑ Derived from the Greek word for helmsman or pilot (hence the logo of a helm)
- ❑ Created by Google, open sourced in 2015

Kubernetes make containers *at scale* possible

# K8s history



- ❑ 2003 - internal @ Google, they used a system called Borg to help with cluster management
- ❑ 2013 – Docker released, makes containers easier than they've ever been by providing a solid ecosystem – docker images
- ❑ 2015 – K8s version 1.0 is open sourced/released and also CNCF formed at same time
- ❑ Some of CNCF founding members include Google, Twitter, Docker, VMWare, IBM..
- ❑ Goal of CNCF to advance adoption of container technology
- ❑ Amazon's ECS released 6 days after K8s
- ❑ ECS not open source like K8s
- ❑ ECS – EC2 Container Service
- ❑ Meant to work with other Amazon services

# Container orchestration - why?



# K8s terminology: pod, node, cluster



- ❑ Docker Container/Image – wrapper around an application
- ❑ Pod (k8s) – wrapper around a container
- ❑ Node – collection of pods. Minikube runs a single node k8s cluster on your virtual machine. Think of your VM as a single 'node' right now
- ❑ Cluster – collection of nodes. Analogy: server rack
- ❑ Hierarchy is: your application -> Docker container -> pod -> node -> cluster

# A note on Docker Swarm



- ❑ Docker Swarm is also a container orchestrator. By Docker of course
- ❑ But k8s is far more popular and “enterprise-grade”
- ❑ Even Docker somewhat conceded by including k8s within Docker Desktop

# Kubectl



- CLI for running commands against k8s clusters
- K8s exposes an API that we can call into



# Kubectl commands



- Common Kubectl Commands:
- Apply – apply a template file to your cluster, its like create/update:  
***kubectl apply -f workloads.yaml***
- Get<service, pod, etc> <name> - get information about the resource -  
***kubectl get pod mypod***
- Get all – return a full page of details about your cluster's resources  
Describe – get detailed information about a resource  
    ***kubectl describe service myservice***  
Logs – return console output from a pod (or a container inside it)  
    ***kubectl logs mypod***  
In production, you may also want to provide a namespace for any and all kubectl commands:  
    ***Kubectl -n <namespace> logs mypod***

# Kubectl – connect to a pod



- Common kubectl Commands
- Exec – launch a bash shell inside a pod:
- `kubectl exec -it <pod-name> -- /bin/bash`
- To connect to a service (we'll talk about services shortly):
- **`kubectl -n <namespace> get svc sample-service`**

# Walmart case study



- You have an application - Java, python, whatever...
- CG says: containerize it and run it in k8s, b/c there's advantages
- What is the first step to getting it to work?
  - 1. Add a Dockerfile to create the Docker image...
  - 2. How do we go from Dockerfile to Docker image?
    - We run docker build to create the Docker image
    - The question is - who/what will run the docker build command?
      - build scripts
      - CG: uses bamboo for CI, some teams use Jenkins
      - Why don't we just manually run "docker build"
        - Can't keep track of build artifacts (like Docker image)
        - It's easier to automate versus manually running builds - especially if you run a lot of builds
        - Good for versioning, it's faster, you can automate unit tests - HUGE
        - vulnerability scanning of builds as well
      - At the end of all the build, we end up with a Docker image assuming the build goes OK
      - Where does the Docker image go?
        - an artifact repository which stores images for maybe all applications within CG
          - Nexus, Bamboo provides one, Artifactory
    - 3. Ok, my docker image works and is in an artifact repository - so how do we get it to K8s?
      - We need something automated that will take the Docker image and deploy it to K8s
        - Hey k8s, here's an updated version of the Docker image
        - That can be a lot of different tools
          - Walmart uses something called Concord
            - basically something that will run these kubectl commands for you in an automated fashion...
          - Terraform
          - Cloud Formation



# Kubectl – example - get all

- Let's run kubectl get all, what do we see...
- It shows me both pods and services I currently have running:

```
vn0silf@m-c02xr115jgh7:~$ kubectl get all
NAME           READY   STATUS    RESTARTS   AGE
pod/hello-pod  1/1     Running   8          26d
pod/nginx-pod  1/1     Running   14         61d

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   10.96.0.1     <none>       443/TCP       62d
service/nginx-service NodePort    10.106.89.56  <none>       80:30080/TCP  60d
```

# K8s pods



Pods are lowest/smallest unit of work within K8s

Containers at a high level are wrappers around your applications

Pods are another wrapper around your containers

Your application -> Docker container -> K8s pod

Typically just one container per pod

Pods are mortal

This is key, so remember this!

# K8s – more on pods



So, K8s doesn't manage containers – it manages pods

Every pod gets a unique IP address

That IP address is not available outside the K8s cluster

So, by default, pods are not accessible outside the k8s cluster

In practical terms that means we can't directly access the application running in our container

For instance, if we are only running a pod with a Jenkins Docker image we can't open our browser and access an endpoint with our running instance of Jenkins

# Lab 1: KUBECTL + pod exercise – apply

1. First, ensure that you have Kubernetes up and running via minikube - you can run minikube start
  2. Copy the contents from the pod.yaml file here locally in a new folder (just name the folder k8s):  
<https://github.com/varoonsahgal/k8s/blob/main/pod.yaml>
- 
1. Use **kubectl apply -f pod.yaml** to apply that file – basically that creates a pod with the Docker image specified
  2. Let's do a **kubectl get all** or **kubectl get pods** to view it
  3. Use **kubectl describe** to get more info about the pod
  4. Now, lets launch a shell inside our pod to play around:
    1. Run `kubectl exec -it my-nginx-pod -- /bin/bash`

# Lab 1: KUBECTL + pod exercise – apply

5. Now, lets launch a shell inside our pod to play around:

1. Run `kubectl exec -it my-nginx-pod -- /bin/bash`
2. Once inside the pod, try running `curl http://localhost`
  1. But, it will complain that curl is not found!
  2. So, we can use apt – advanced package tool – which is used to install/remove software on various Linux distros
  3. Now, do **apt-get update**
  4. Then, run **apt-get install curl**
  5. Finally, try to **curl localhost** again
  6. You should get a response this time w/ the nginx default html!
  7. Type “exit” to leave the shell

6. Question: If I delete the pod, do another apply to create a new one, will the curl library still be there inside the new pod?

7. Takeaway from lab: we don't yet have a way of accessing the pod from our own browser – we haven't exposed it through a service. That's coming up!



# KUBECTL example - describe

With describe we can get a lot more information about our pods

Much like “inspect” in the Docker CLI

Let’s run kubectl describe on our new pod: kubectl describe pod/my-nginx-pod

Notice ClusterIP for the service - this is the IP address assigned to the pod available **WITHIN** the cluster not **OUTSIDE** of the cluster

In other words, you can’t use this IP address to connect to the cluster from outside world

Pods can exist without services, but they will NOT be accessible, so let’s see how to access one

And a service without a pod is just an empty endpoint - will give you a “host not found”

# KUBECTL delete



Let's delete our pod

## **kubectl delete -f pod.yaml**

We can delete it by specifying the file used to create it via apply

We can also delete by name: **kubectl delete pods/pod-name**

Can we "stop" pods in K8s like we stopped containers in Docker?

Not really, as that concept doesn't transfer from Docker to K8s pods

Closest thing in K8s is to set replicas of a pod to 0 (we'll cover this soon)

Go ahead and apply the pod again as we will need it for next lab –  
**kubectl apply -f pod.yaml**

# K8s + Docker images

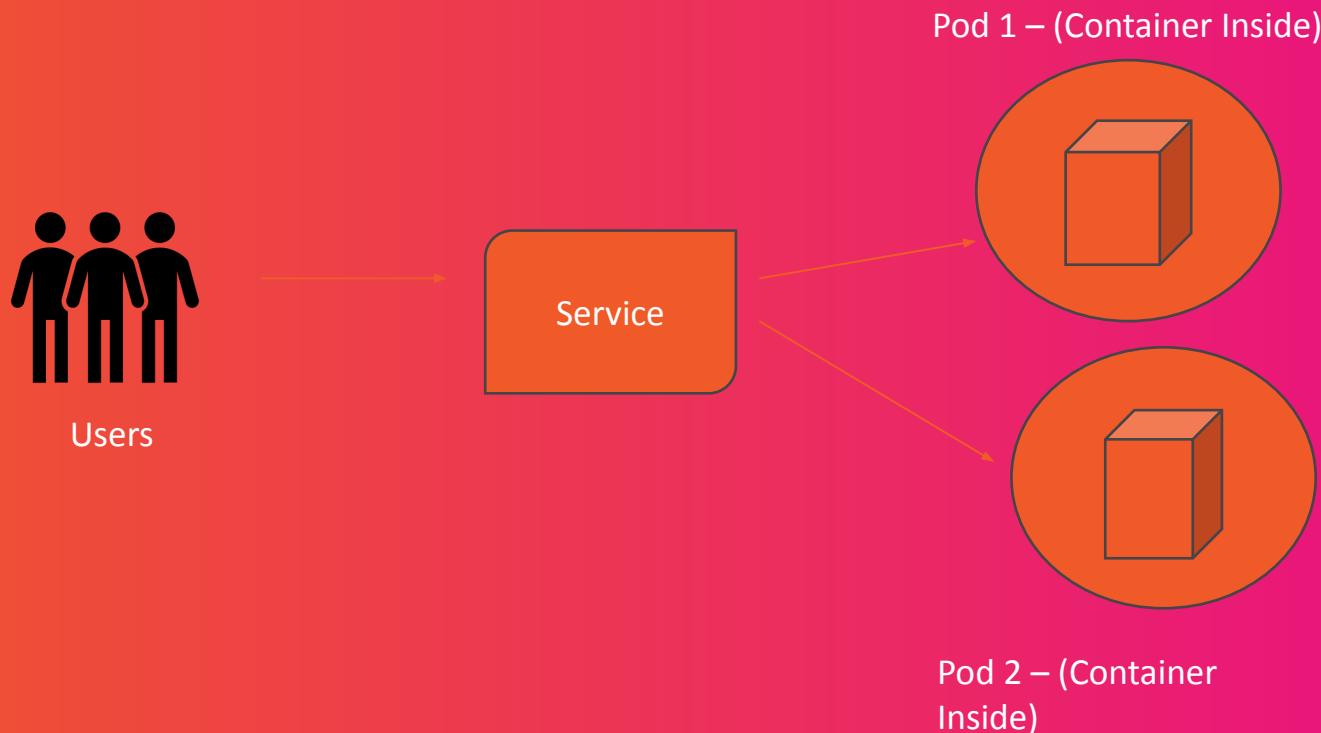
- Where does k8s look to find images?
  - It will first look at your machine to see if you have any images
  - If it doesn't find it there, then it will pull it from DockerHub
  - You can also specify the repo where it looks
  - Remember we have this in our pod.yaml file
  - (See containers section in image on right):

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx:1.17.0
```

# K8s – services

- So, how do we expose our pods to users? Through something called a **service** in K8s
- Why services? Why not just access pods directly?
  - Because pods are mortal
    - K8s often needs to kill pods – an app might be over consuming resources
    - Or when we are scaling up dramatically, new pods are added and removed at a fast pace, which makes having a stable endpoint even more important
- So, **services** give us a stable, single point of entry into our pods...

# K8s – services: expose pods



# K8s - services

- Like pods, services are another K8s construct
- Services are what essentially expose our pods – which hold our applications - to the outside world
  - So services are a gateway to our applications in K8s
- Now, if pod dies, service will handle the IP address mapping between the endpoint that the user hits and the new pod replacing it
- You can think of services as load balancers for your pods, which distribute traffic across pods

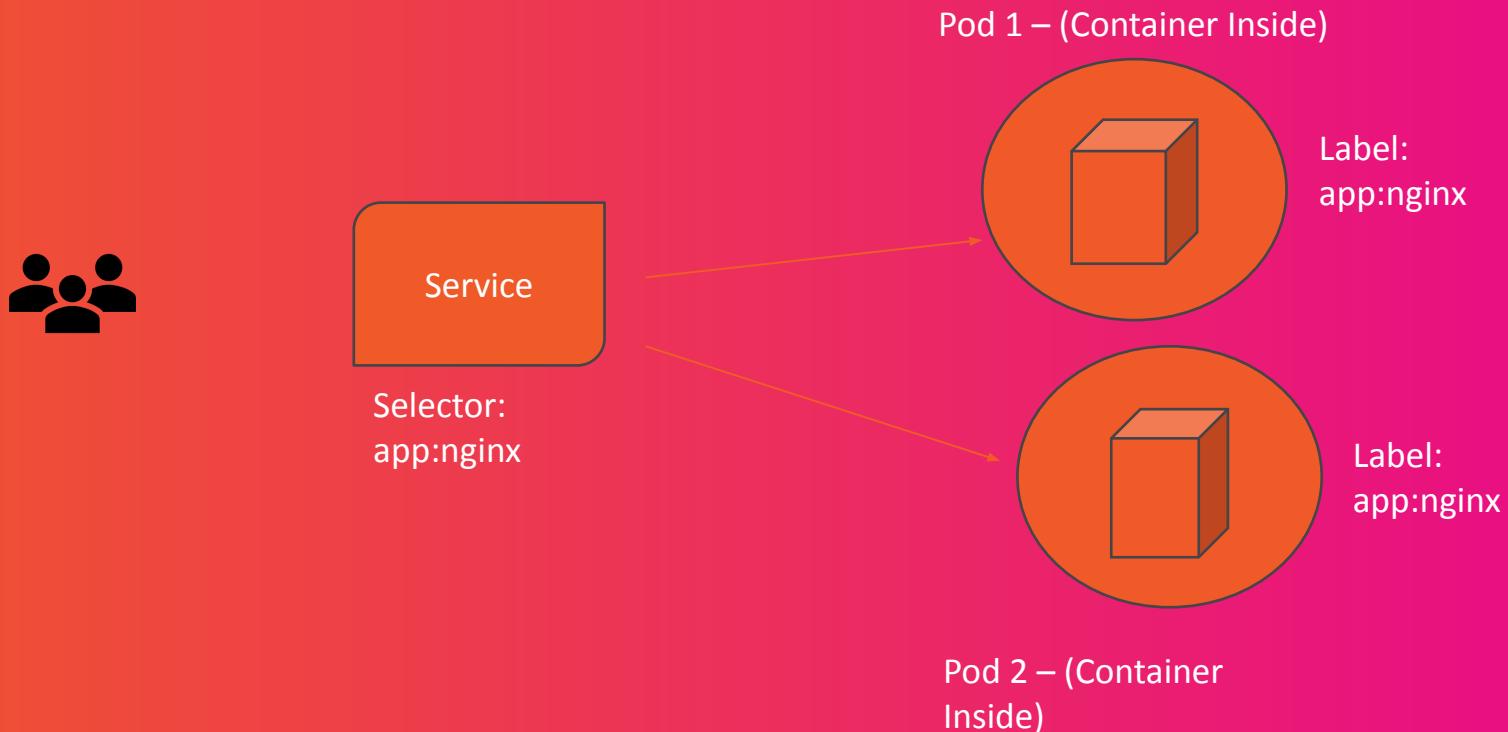
# K8s – service types

- ClusterIP – default option. Provides access inside the cluster. Good for services that should remain internal to the application like a database.
- NodePort – Provides a port outside of the cluster. Makes your service public.
  - A ClusterIP Service, to which the NodePort Service routes, is also automatically created when you use a NodePort.
  - Note: the port you expose on the cluster is only valid from 30000-32767
    - **We will be using this service type in our lab to expose a pod that we create!**
- LoadBalancer – Exposes the Service externally using a cloud provider's load balancer.
  - Only valid on cloud implementations, and creates a load balancer and attaches it to your service.

# K8s labels– services + pods

- How do we tell a service to forward traffic to a pod?
- Through labels!
  - Labels are in key:value format
  - Pods can have multiple labels
- So, a service would have a label selector and then a given pod will have a label which would match that label selector
  - Which in turn means that the service will forward traffic to that particular pod!
- Basically, the service says “all pods on the cluster with these labels are mine and i will load balance them!”

# K8s – labels in services + pods



# K8s – Service.yaml

```
! service.yaml
1   kind: Service
2   apiVersion: v1
3   metadata:
4     name: my-nginx-service
5   spec:
6     selector:
7       app: nginx
8     ports:
9       - protocol: TCP
10      port: 80
11      nodePort: 30001
12      type: NodePort|
```

- Notice the selector: **app:nginx** – in our pod.yaml we also defined a label saying **app:nginx**
  - Because the selector matches our pod.yaml label, K8s will know to put our pod behind this service!
  - NodePort is the port which we can use to access this service from browser
    - Remember port mapping in Docker? This is same idea except with K8s – nginx is running on port 80, and we are exposing it on 30001

# Lab 2: K8s services

1. Our goal here is to apply a service locally and connect it to our nginx pod. That way, we can access nginx through our browser
  1. Remember – pods are not directly accessible outside of K8s cluster, we must expose them through a service!
2. Look at the service.yaml file here:  
<https://github.com/varoonsahgal/k8s/blob/main/service.yaml>
  1. Make a copy of that file locally in the same directory as your pod.yaml file
3. Now, apply the file to create the service in K8s! Run **kubectl apply -f service.yaml**

# Lab 2: K8s services pt 2

4. You should now be able to view nginx running from your browser – go to <http://localhost:30001>
  - So, our service is exposing our pod, which in turn is running the nginx Docker container! Beautiful!
5. Try describing the service via kubectl to get more info – **kubectl describe service/my-nginx-service**
6. Try deleting the pod – **kubectl delete -f pod.yaml**, and now try accessing the service in the browser – you should see nothing!
7. Go ahead and reapply the pod

# K8s – sidecar

- Typically you would only put one container per pod
- In rare cases, a pod can have multiple (>1) containers if those containers are tightly coupled
  - The sidecar pattern is one example of when we would have multiple containers in a single pod
  - In this pattern a secondary container's only job is to support the primary container
  - For example, you may have a secondary container that just sends logs from the primary container to a central logging system

# K8s – declarative model

- Example of a pod.yaml file:

```
10 lines | 148 bytes

1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nginx-pod
5    labels:
6      app: nginx
7  spec:
8    containers:
9      - name: nginx-container
10     image: nginx:1.17.0
```

# K8s – declarative model

- K8s uses infrastructure as code concepts
  - *Infrastructure as code (IaC) means managing your IT infrastructure using configuration files.*
- Declarative vs imperative:
  - Declarative is like asking someone to bake a cake and giving them the ingredients without a recipe – the baker will figure it out
  - Imperative is like giving ingredients + recipe - telling baker exactly what to do..ie here are the exact steps you need to take to bake a cake

# K8s Replicasets

- What is self healing? When k8s automatically spins up a new pod when one goes down
- Replicasets and deployments are the two ways to achieve self-healing within K8s
- When creating a Replicaset, we don't need to define the pods as a separate resource – ie as a separate pod.yaml file.
  - Pods are created within the replicaset and are “owned” by that replicaset

# Lab - K8s Replicasets

- Grab the replicaset file and apply it locally:  
<https://github.com/varoona/sahgal/k8s/blob/main/replicaset.yaml>
  - - **kubectl apply -f replicaset.yaml**
- You can see the pods created by the replicaset if you do a **kubectl get all**
- Try deleting a pod and see what happens – it should self heal and bring it back up (kubectl delete pod/pod-name)
- Look at desired, current and ready states!
- Your service should still be running
- Note that this replicaset is still attaching to our service through the label selector
- so if service is still running you should be able to see the nginx response in the browser by going to <http://localhost:30001/>
- Try deleting the service, do you still see the nginx homepage?
- Everything under template is specific to the pods its creating
  - Note that we don't explicitly use the word pod in the replicaset.yaml file, it's implied!

# K8s Deployments – the what

- Deployments are another K8s construct (like pods, services, replicaset(s))
- Deployments actually create replicaset(s)
- Why do we need deployments if they just use replicaset(s) under the hood?
- Because deployments **allow us to rollback to prior application states** if something went wrong with our rollout
  - For instance the new version of our application does not work as planned, so we want to go back to the earlier working version. Very easy in K8s through deployments!
  - Plus we still get self healing because we are using replicaset(s) created

# K8s - Triggering Deployments

- A new deployment will trigger a new replicaset to be created
- How do we trigger a new deployment rollout? By updating the pod template in the deployment.yaml file (like the labels or image)
  - We would then need to reapply the deployment.yaml as well
  - Note that if we update the number of replicas that does not trigger a “new” deployment, just an update to the number of pods
- If we have 1000’s of replicas, it will take longer to rollout the new updates –so some users may see the older version of your application post-deploy until all pods have been updated

# K8s Deployments – Rolling updates

- If we go from v1 to v2 of our image (our application), then there will be an intermediate phase where some pods are running on v1 and some on v2!
- This is what's known as a **rolling update**, and ensures that we don't experience any downtime!
- For our lab exercise, let's pretend we are “upgrading” our application from an Apache server to a Nginx server – so we will modify the Docker image being used in the deployment that we will create....
  - So, we will go from Apache -> Nginx

# K8s – Kubectl Deployment commands

- You can rollback with: **kubectl rollout undo deployment apache-deployment**
- You can see rollout status with: **kubectl rollout status apache-deployment**
- You can see rollout history with: **kubectl rollout history deployment apache-deployment**
- You can pause and resume rollouts
- More details in the docs:  
<https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>

# Lab - K8s Deployments

- Delete any replicaset/pods that you might have running via **kubectl delete** (but leave the service)
- Grab the deployment file [here](https://github.com/varoonsahgal/k8s/blob/main/deployment.yaml) and apply it locally:  
<https://github.com/varoonsahgal/k8s/blob/main/deployment.yaml>
  - - **kubectl apply -f deployment.yaml**
- Do a **kubectl get all** and notice that it creates a replicaset, 10 pods, and also a “deployment”
- Try deleting a pod and see what happens – it should self heal and bring it back up
- Look at desired, current and ready states!
- Your service should still be running
- Note that the **deployment** is still attaching to our service through the label selector – we still need a **service** to expose our pods!
- If service is still running you should be able to see the Apache response in the browser by going to <http://localhost:30001/>

# Lab - K8s Deployments – pt 2

- Now, let's update our deployment.yaml file to use nginx image instead of httpd inside the deployment.yaml file. Go ahead and reapply the deployment file via **kubectl apply –f deployment.yaml**
- **This is the equivalent of doing a new release for our application!**
- Now, notice that a new replicaset has been created and the old one still remains
  - The old one is there to allow us to roll back if we need to!
- Go to <http://localhost:30001/> and you should see that it's updated to use the nginx homepage!
- Notice that you never experienced any downtime – because K8s is gradually terminating the old Apache pods and replacing them with the new nginx pods – it's not killing all 10 pods at once and then bringing up 10 new ones
- Also, notice that the old replicaset is still maintained – this is key for rollbacks!
- So, what if we want to rollback to our earlier deployment with the Apache container image?
  - Simple!
- Just run **kubectl rollout undo deployment apache-deployment**

# K8s Best practice - using probes

# K8s – Why do we need probes?

- Pods and their respective containers can misbehave.
- If a pod goes down (maybe due to a network issue), a deployment/replicaset can and will bring it back up to maintain a certain count
  - The replicas count if you recall from your deployment.yaml files
- However, what if our application is deadlocked? To K8s, the pod will still seem healthy and it will keep running unless we do something about that...
- So, by default, K8s has no way of knowing if our application (containers) are misbehaving
- But, we can fix that with probes! This is considered a ***best practice*** with k8s to ensure the “robustness” of your pods

# K8s –probe types

- We will focus on 2 primary probe types: liveness and readiness
- **Liveness** probes basically are a check that runs periodically (you define how often) and asks your application are you alive, are you alive, etc?
  - If the answer is "no" k8s will **restart** your pod since that should bring your container/application pod back to life
  - If the answer is "yes" nothing will happen since your pod is healthy
- **Readiness** probes are similar except they are basically checks to see if your pod is ready to accept traffic
  - Sometimes our applications need to load large data or configuration files during startup, or depend on external services after startup.
  - In such cases, you don't want to kill the application, but you don't want to send it requests either
  - So, readiness probes will prevent traffic from being sent to pods that are not ready to accept it.

# K8s –combining readiness + liveness probes

- Note that you can (and often should) combine both liveness and readiness probes for a robust pod

# K8s – Parameters for liveness/readiness probes

Both readiness and liveness probes have additional parameters

They are the same exact parameters for both probes...

And if you don't specify these parameters they do have default values as well:

**initialDelaySeconds**: Probes start running after initialDelaySeconds after container is started (default: 0)

**periodSeconds**: How often probe should run (default: 10)

**timeoutSeconds**: Probe timeout (default: 1)

**successThreshold**: Required number of successful probes to mark container healthy/ready (default: 1)

**failureThreshold**: When a probe fails, it will try failureThreshold times before deeming unhealthy/not ready (default: 3)

# Liveness probe lab

- Go ahead and open this file : <https://github.com/varoona/sahgal/k8s/blob/main/probe.yaml>
- 

You can see that this Docker image is what's being used <https://hub.docker.com/r/varoona10/teachingaid>  
teachingaid is a simple Docker image using nginx

It is setup so that it will first sleep for 30 seconds, and then delete the homepage automatically

By deleting the homepage, the httpGet to the homepage on lines 12-14 will fail

This triggers the livenessprobe

The liveness probe will then automatically restart the teachingaid nginx pod

# Liveness probe lab

- Now, copy the file locally and apply it using kubectl:  
<https://gecgithub01.walmart.com/developer-portal/wcnp-dd-files/blob/main/probe.yaml>
  - Just do a kubectl apply –f probe.yaml
- You should be able to access the service at port 30080
- Do a kubectl get all and pay attention to the pod's restart column
  - You can see that after 30 seconds the pod will be restarted by the liveness probe
- Do a kubectl describe liveness-http-nginx and you should see the restarts
- What would happen if we did NOT have a liveness probe? Try it out!

# K8s – Liveness probe lab

Our example is a bit contrived in that we auto-delete the homepage in the teachingaid Docker image

But, imagine if a container/pod is deadlocked – it's still “running” but not doing anything productive

With liveness probes we can check for that sort of thing and have K8s automatically restart it when need be!

# AWS Lambdas vs Containers

# AWS Lambdas vs Containers



- ❑ At CG, we want to promote creating web applications with containers – effectively “Dockerizing” your applications
- ❑ Lambdas are meant to be used for event-driven stacks + short-burst use-cases as opposed to applications with a long lifetime

# AWS Lambdas vs Containers



- ❑ When implementing a microservice architecture, using Docker/containerization is best
- ❑ This is b/c Lambdas are somewhat limited in their support for different programming languages
- ❑ Whereas with Docker you can use any language that can run on Linux/Windows
- ❑ Remember AWS lambdas are limited to AWS! But Docker containers we can move to other cloud providers like Azure/GCP if we need to

# AWS Lambdas vs Containers



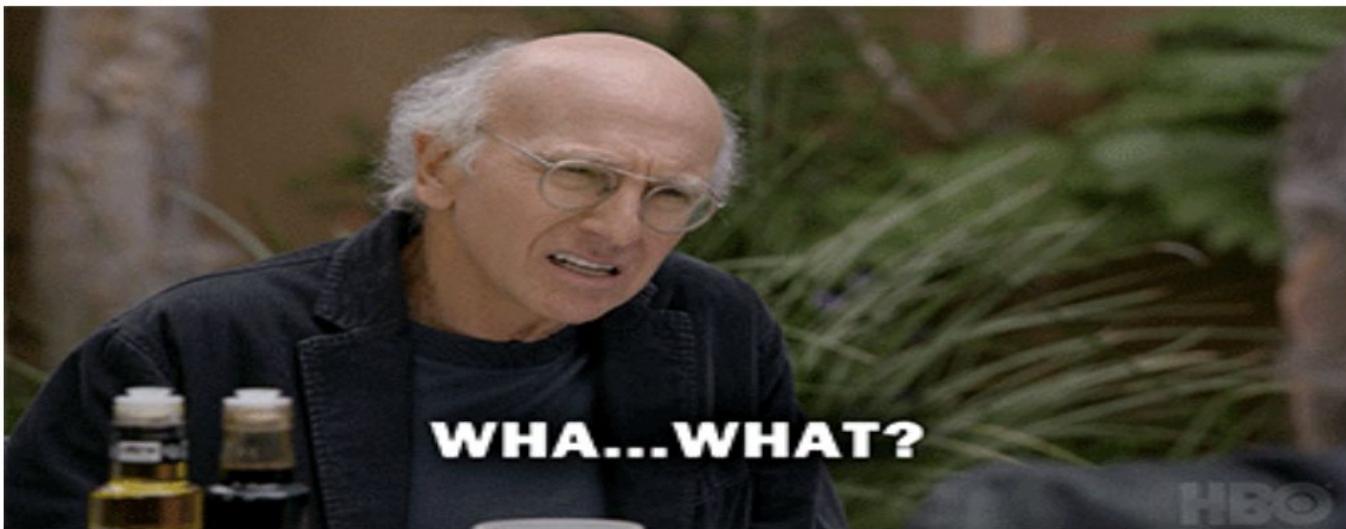
- ❑ Since lambda is a function based service, it's very challenging to manage an entire service as a collection of inter-connected Lambda functions
- ❑ Keep in mind that you can still have an application that's containerized, which calls into a Lambda
- ❑ Sometimes – it's not an either/or w/ Lambdas vs Containers – you can have a container based web app calling into lambdas

# Container Based Solutions in AWS

# ECS, EKS, EC2, Kubernetes?



- ❑ Let's make sense of all these acronyms - and discuss the differences!



# ECS vs K8s



- ❑ ECS (Elastic Container Service), like Kubernetes, is container orchestration
- ❑ However, ECS is not open source – it's AWS proprietary
- ❑ Kubernetes *is* open-source
- ❑ Kubernetes was open-sourced July 21<sup>st</sup>, 2015
- ❑ ECS was released as a service just 6 days later July 27<sup>th</sup>, 2015
- ❑ Fargate was released in Nov. 2017

# EKS – Elastic Kubernetes Service



EKS – managed service for K8s in AWS

So, you don't have to be an expert in managing K8s clusters

Before EKS was released in Nov. 2017, customers were still running K8s on AWS, but it required a lot more work on their end – they had to deal with patches/upgrades/load balancing all on their own

So they asked AWS to come up with a managed service for Kubernetes – EKS!

# Fargate – ECS + EKS



- ❑ Fargate allows you to run containers without having to manage servers or clusters of EC2 instances...
- ❑ Fargate can be used with both ECS and EKS
- ❑ Fargate is not a separate AWS service, has to be used with ECS or KS

# ECS – launch types

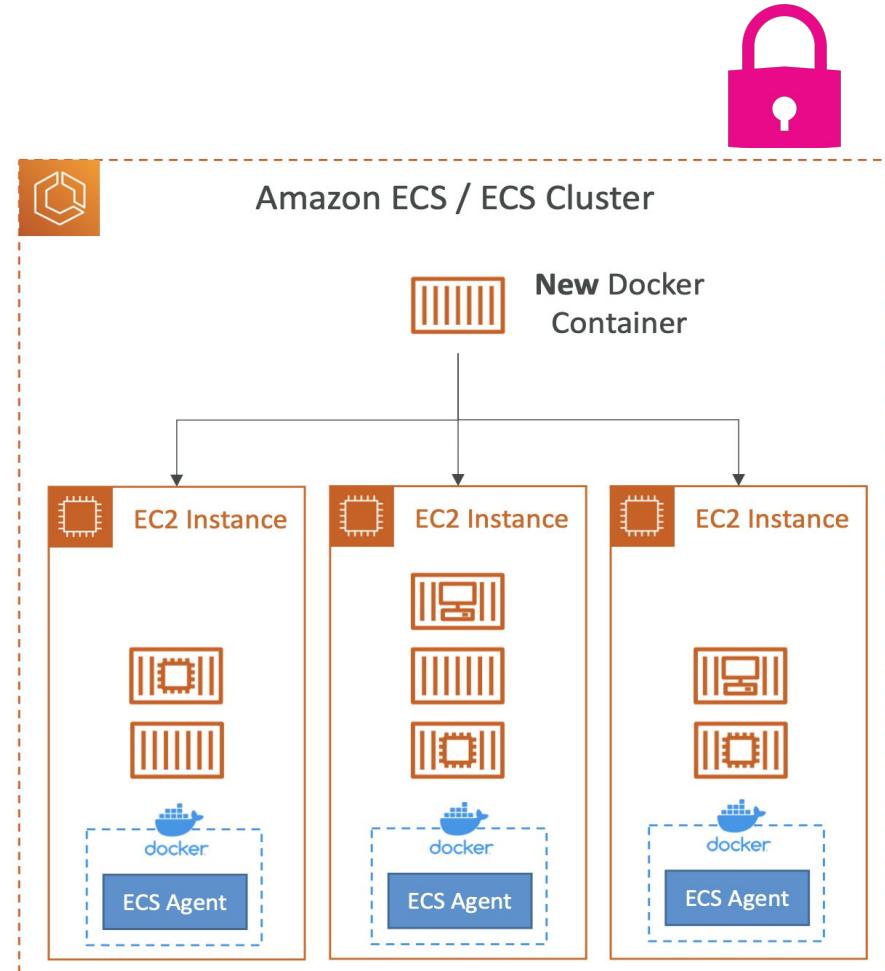


- ❑ There's essentially 2 ways of using ECS
- ❑ These are known as *launch types*
- ❑ 1. ECS + EC2
- ❑ 2. ECS + Fargate
- ❑ Let's explore and understand both options starting with ECS + EC2...

# ECS + EC2 – option 1

Your Docker containers will live inside EC2 instances (EC2 =~ IaaS virtual machines)

- With ECS + EC2 – you have to provision the EC2 instances in **advance** – remember EC2 is IaaS – you maintain them as well
- ECS however does provide container orchestration for us – it starts and stops containers for us
- Note that each EC2 instance has an ECS Agent which basically says “this EC2 Instance is part of the ECS Cluster”

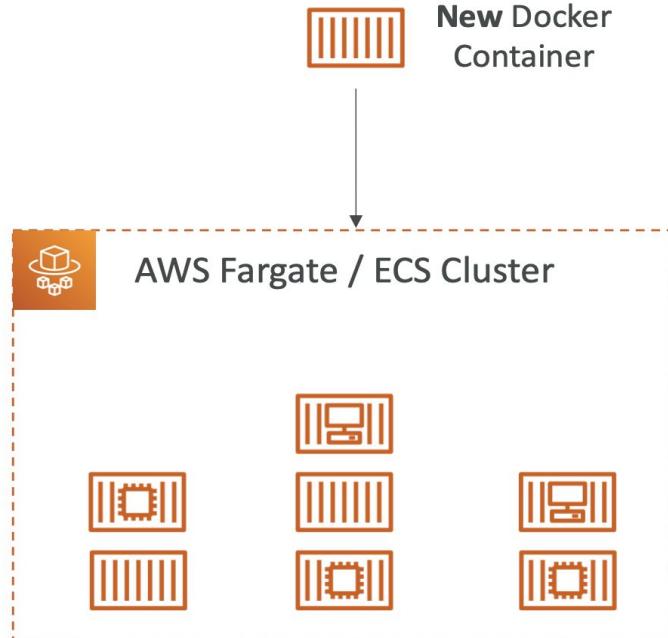


# ECS + Fargate – option 2



❑ Fargate is considered **serverless**

- ❑ With ECS + Fargate, no EC2 instances
- ❑ So, you do NOT need to provision the infrastructure in advance – no EC2 instances to deal with. This is PaaS
- ❑ Then, to scale, you would just need to create what's known as task definitions
- ❑ Then, ECS will run those tasks for you
- ❑ Fargate is easier than ECS + EC2 launch type



# ECS Labs – perform in order



- ❑ <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/get-set-up-for-amazon-ecs.html>
- ❑ <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-cluster-console-v2.html>
- ❑ [https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs\\_cwet.html](https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_cwet.html)
- ❑ <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/tutorial-cluster-auto-scaling-console.html>

# ECS Tasks – analogy to K8s



- Remember our deployment.yaml from the k8s lab?
- Tasks in ECS are actually quite similar to deployments in K8s
- Because in a task, like a deployment resource in K8s, we specify things like the Docker image to use, CPU/memory usage, etc..
- Remember that both ECS and K8s are container orchestrators
  - same goal, just different implementations

# EKS – Key Points



- Also K8s is cloud-agnostic and supported by all major cloud providers like Azure/GCP
- So using EKS instead of ECS gives an extra level of “safety” in case you want to change cloud providers
- Because again, ECS is AWS proprietary technology – other cloud providers can’t use it unlike K8s

# EKS vs ECS



- ECS is container orchestration, not open source
- EKS is K8s which is open source
- Both provide container orchestration...
- So, which one should we use? How
- Sometimes migrating K8s from one provider to another then EKS may be best bet
- EKS good if you already running K8s before EKS introduced

# EKS vs ECS



- ECS is container orchestration, not open source
- EKS is K8s which is open source
- Both provide container orchestration...
- So, which one should we use? How
- Sometimes migrating K8s from one provider to another then EKS may be best bet
- EKS good if you already running K8s before EKS introduced

# EKS Hands On



- ❑ Execute the tutorial here:
- ❑ <https://docs.aws.amazon.com/eks/latest/userguide/getting-started-console.html>
- ❑ And another one here:
- ❑ <https://github.com/varoonsahgal/cg-cloud-foundations/wiki/EKS-Lab>

# ECS – clusters



- ❑ ECS uses the term clusters just like Kubernetes
- ❑ To launch Docker containers using ECS, we use ECS tasks

# Operating in the Cloud

# Monitoring in the Cloud



- Monitoring & logging are key considerations in any Cloud environment
- Systems (you hope) will be running around-the-clock – maximizing business benefit
- Unless you want to directly “babysit” those systems around-the-clock, you will need automated monitoring, logging and alerting to notify you of any issues
- Allows you to optimize handling for those exceptional cases when there is a problem

# Monitoring in the Cloud



- Capabilities exist to log and aggregate log data from the public Cloud
- There are likely already processes in place to monitor on-premise systems
- The goal is a strategy that allows you to pull that data together so you can analyze and make decisions holistically

# Endpoint Protection & Security



- The services exposed by a company used to provide its business value are critical
- The data consumed by a company in the provision of that business value could be very sensitive
- There are multiple regulations in place requiring the protection of sensitive data (e.g., PCI, SOX, HIPAA and GDPR)
- Failure to adhere to those regulations can cost a company significantly – either in actual \$'s or in reputation (which can be more damaging)

# Endpoint Protection & Security



- The issue is not only one of data security – there are “bad actors” that work to take down sites and services
- One of the ways that service can be hindered is through a DDoS (Distributed Denial of Service) attack
- For DDoS, attackers will attempt to “flood” a service with so much bogus volume that it becomes unable to satisfy real business requests

# Endpoint Protection & Security



- Most Cloud platforms provide services to help you protect against a DDoS attack
- Can include API management services (subscriptions, key-based access, throttling)
- Web Application Firewall (or WAF) is another service provided by Cloud platforms to monitor, filter and block (if required) incoming traffic

# Endpoint Protection & Security



## Potential Challenges

- The threat and regulatory landscapes are constantly evolving
- In a Hybrid Cloud scenario, creating a solution that gives you comprehensive protection across your on-premise resources, your Cloud resources and the connectivity between the two is not trivial
- Optimal application security and infrastructure security requires planning and specialized skillsets
- Good architectural practices (e.g., Least Privilege and Secure-by-Default) can help limit the “blast radius”

**DEMO**



## Operational Management in the Cloud

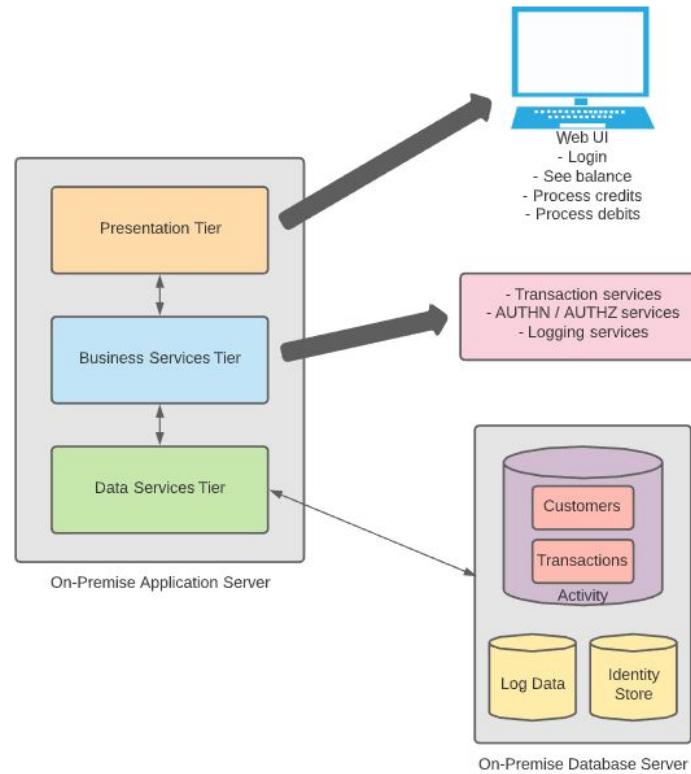
# Cloud Modernization

# Application Portfolio Assessment

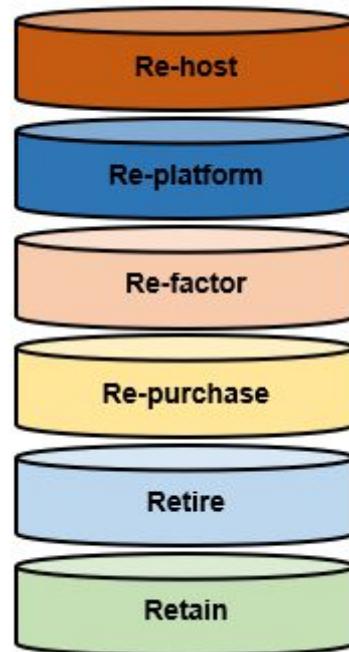
- Will likely include
  - Review of the current state architecture
  - Discussion with stakeholders about roadmap and plans for target state
  - Discussion on expected timing
- The goal is to ensure that migration efforts are focused on the activities that will bring greatest business benefit against optimized cost



# Banking App – Current State



# The 6 R's – Application Migration Strategies

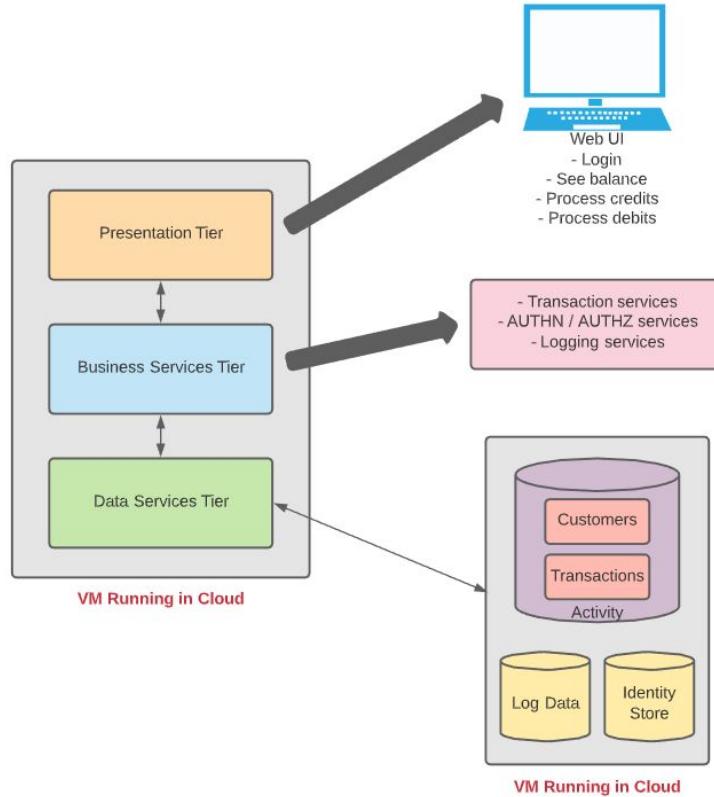




- AKA “lift & shift”
- For all intents and purposes, involves recreating the on-premise infrastructure in the Cloud
- Sometimes used to expedite retirement of a data center



- Can be a mechanism to quickly migrate workloads and see immediate cost savings, even without Cloud optimizations
- There are third-party tools available to help automate the migration
- Once the application is in the Cloud, it can be easier to apply Cloud optimizations vs. trying to migrate and optimize at the same time

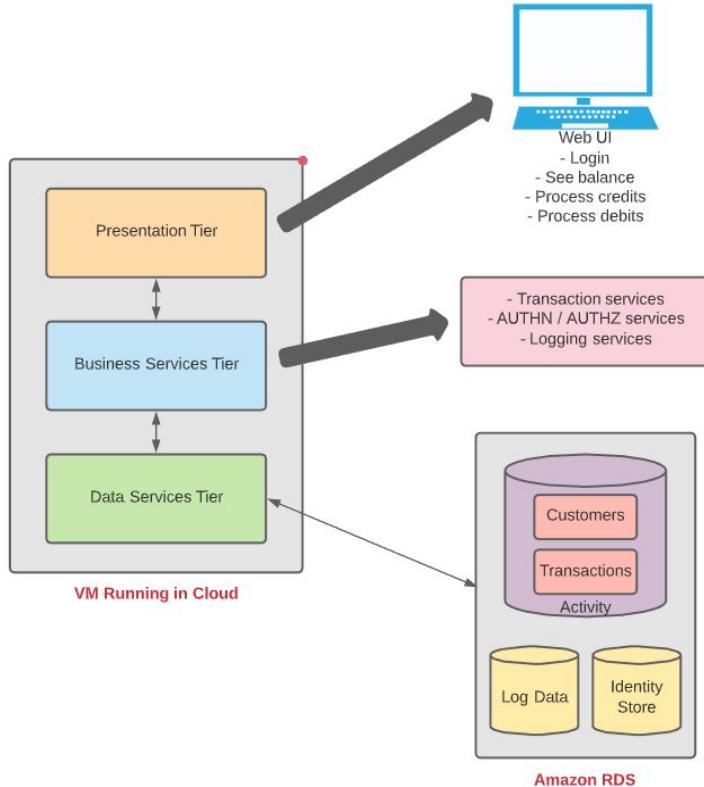




- AKA “lift, tinker & shift” or “lift & fiddle”
- Core architecture of the application will not change
- Involves recreating most of the on-premise infrastructure in the Cloud with a few Cloud optimizations



- Those optimizations will involve a move to one or more Cloud native services for a specific, tangible business benefit
- A common example is moving to a managed database (e.g. Relational Database Service, or RDS, in AWS)
- Enables migration speed while providing cost savings or benefit in a targeted portion of the application's architecture





- Involves rearchitecting the application to maximize utilization of Cloud native optimizations
- Likely the most expensive and most complex of the available options
- The application profile needs to fit, and business value must be identified commensurate with the cost required to execute



- Good option if the application can benefit from features, scalability or performance offered by the Cloud
- Examples include rearchitecting a monolith to microservices running in the Cloud or moving an application to serverless technologies for scale



## Re-purchase

- Good fit for applications that are candidates for moving from on-premise licensing (for installed products) to a SaaS model
- Often applications that have been installed to manage a specific type of business capability
- Examples can include Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), HR or e-mail (among others)



- In some cases, an enterprise may have a “healthy” percentage of legacy applications that are no longer being used or maintained (especially for larger organizations)
- When completing the Application Portfolio Assessment and associated interviews with application owners, look for opportunities to recommend retire of the unused legacy apps
- This type of application can represent a “quick win”



- The associated savings can potentially be advantageously factored into the business case for the Cloud modernization effort
- Finally, retiring unused apps reduces attack surface area from a security perspective



- Applications that must remain in place but that cannot be migrated to the Cloud without major refactor
- This type of application profile may prevent the ability to completely move out of the data center (at least in the interim)
- From a cost perspective, it can be difficult for an enterprise to take on both the operating expense of Cloud while continuing to carry the capital expense of a data center and on-premise infrastructure
- The hybrid model can be a good solution to support where needed

## LAB: Application Hosting

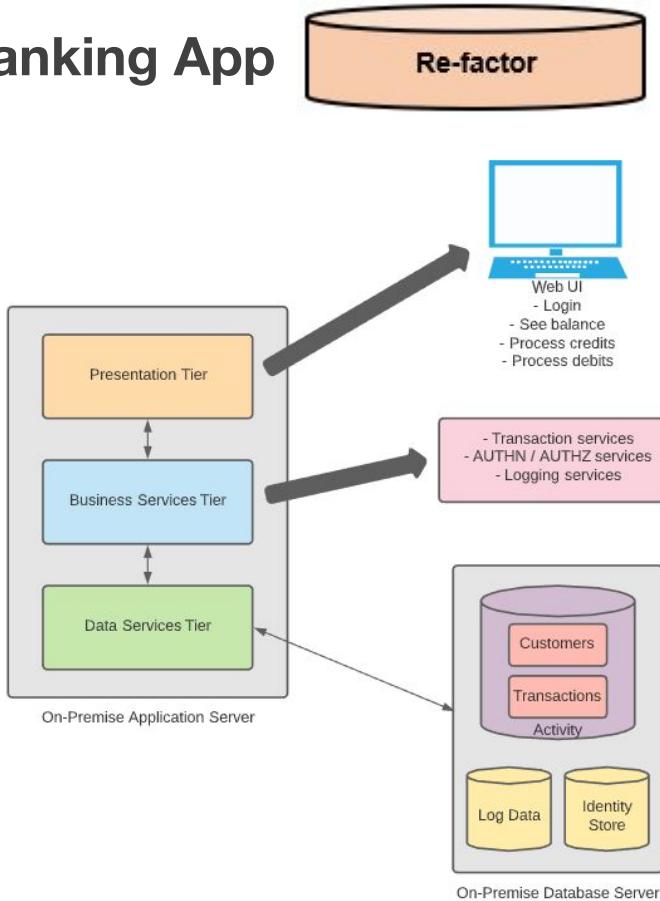
Scenario: Your company (a global leader in FinTech) is currently hosting all infrastructure used to power the business in an on-premise Data Center. This includes a mainframe for primary business functions (customer management, account management, accounts payable, accounts receivable), several Web Apps (for customer interaction), several Web APIs providing backend data and functionality to the UIs, and a system used to manage data feeds from several security cameras used at corporate offices for observation and security.

As a member of the technical staff, you have been asked to provide thoughts and recommendations on moving from the Data Center to the Cloud.

In your assigned breakout room, discuss as a group and be prepared to provide the following: 1) Thoughts & recommendations on which of the 6 R's you'd recommend for each of the high-level infrastructure components used at the company, and 2) any business or technical drivers behind your recommendations.

Nominate someone (or volunteer) to share your group's ideas.

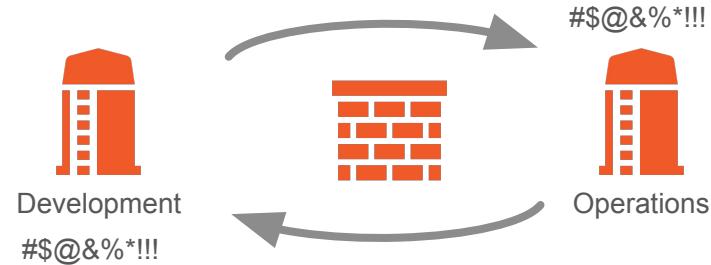
# Banking App



If you wanted to sell an investment project to a client for rearchitecting this application for migration to the Cloud, what are some of the potential benefits you might put forth as justification?

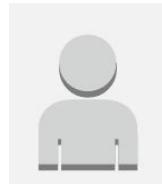
# DevSecOps – What it is & why it's valuable

# Approach Used In the “Olden” Times

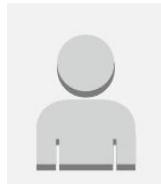


- Development & Operations siloes
- Lack of understanding of the other perspective
- Lack of communication & partnership
- “Fire and forget” engagement

# DevSecOps – The New Way



Development

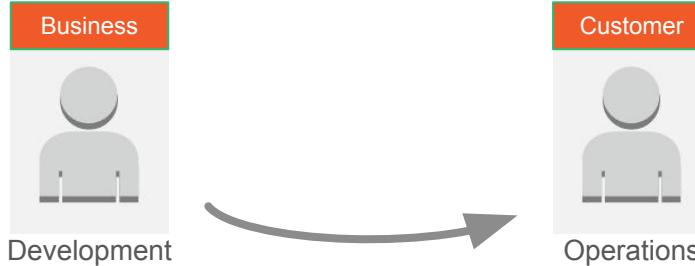


Operations

- Development & Operations work together
- Each understands & appreciates the position of the other
- Good communication, partnership, & collaboration
- Ongoing engagement – continuous improvement

# DevSecOps – The Three Ways

## The First Way: Flow/Systems Thinking

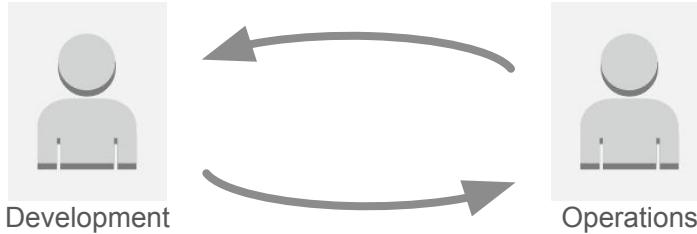


Source: Gene Kim  
(<https://itrevolution.com/the-three-ways-principles-underpinning-devops/>)

- Holistic vs. siloed view – success is defined by the performance of the entire system vs. a specific team or individual
- Starts with requirements (defined by the business) and not called “done” until value is delivered to the customer
- Target outcomes:
  - Known defect never passed to downstream teams
  - Favor global performance vs. focusing on local optimization
  - Goal is increased flow (value)
  - Asking ourselves the question: “What don’t I know about our systems?”

# DevSecOps – The Three Ways

## The Second Way: Amplify Feedback Loops

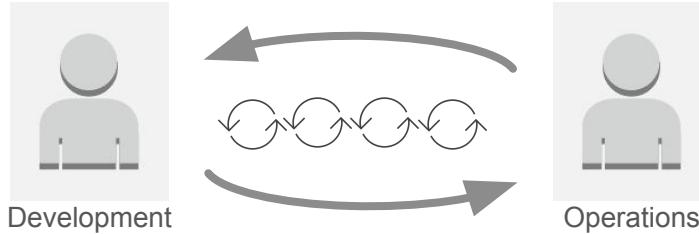


Source: Gene Kim  
(<https://itrevolution.com/the-three-ways-principles-underpinning-devops/>)

- Create right-to-left feedback loops
- Shorten & amplify so information needed to make continuous improvements & continuous correction is readily available
- Target outcomes:
  - Understanding all customer perspectives (internal & external)
  - Responding to all customer concerns (internal & external)
  - Knowledge is power!

# DevSecOps – The Three Ways

The Third Way: Culture of Continual Experimentation & Learning

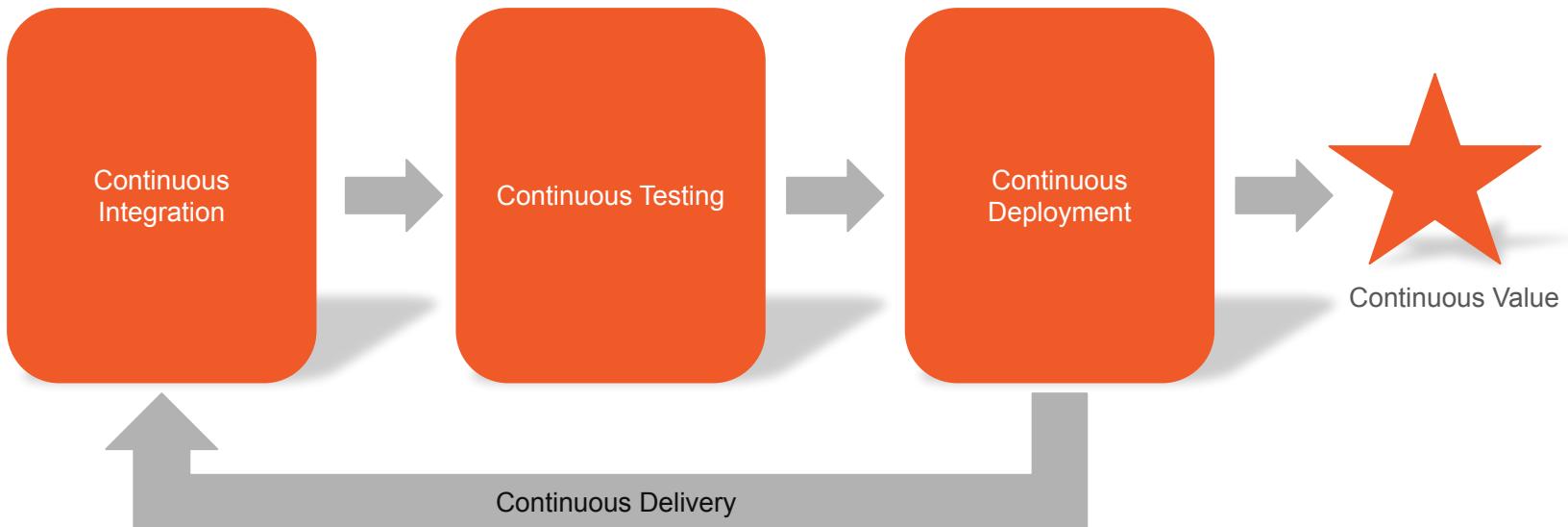


Source: Gene Kim  
(<https://itrevolution.com/the-three-ways-principles-underpinning-devops/>)

- Culture that fosters:
  - Continual experimentation, risk-taking, and learning from failure
  - Practice makes perfect!
- Target outcomes:
  - Allocating time for continual improvement (e.g., resolution of technical debt as a standard “category” of work)
  - Taking risks and seeing them pay off – No guts no glory!
  - Expecting failure & building resiliency into the process

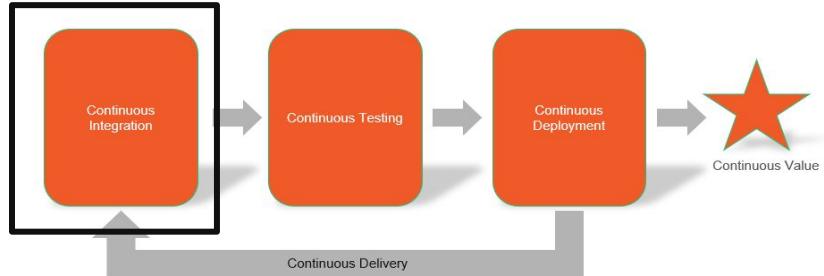
# Continuous Delivery

# Continuous Delivery



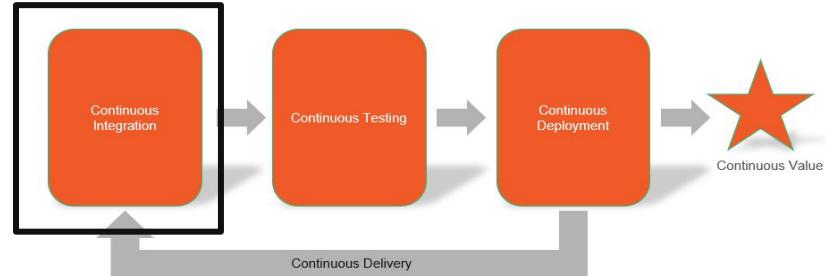
# Continuous Integration

# Continuous Integration



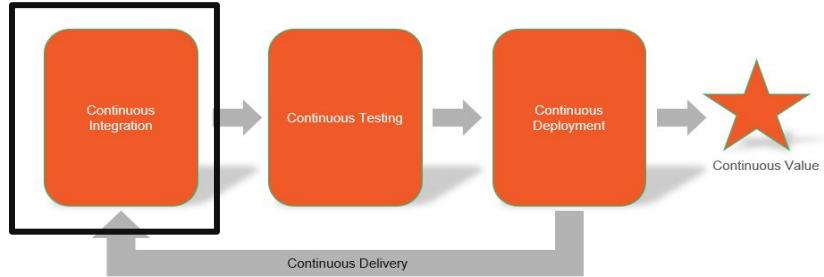
- Changes from individual developers are merged & system is validated “as a whole”
- Code changes are detected & automated processes are executed to assess
- Any issues uncovered are communicated to the developers for remediation (feedback)
- Targeted testing (i.e., unit testing) executed as part of CI flow

# Continuous Integration



- Initiation is often event-driven
- Rather than waiting for a “critical mass” of changes before starting our testing, each change can be exercised & validated as it gets persisted to the centralized code repository
- Amplifies the feedback loop rather than burying in a “sea” of other changes
- Sets the stage for multiple types of validation (including security validation, or DevSecOps)

# Continuous Integration

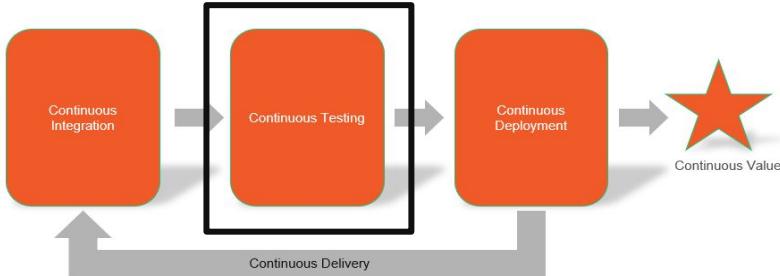


## Best practices:

- Minimize dependencies between features – each feature should be able to stand on its own
- Work to drive out manual steps and make automation the priority
- Stabilize data sources & other system dependencies through mocking
- Include assessment of code quality (e.g., cyclomatic complexity) & unit test coverage as key parts of verification workflow

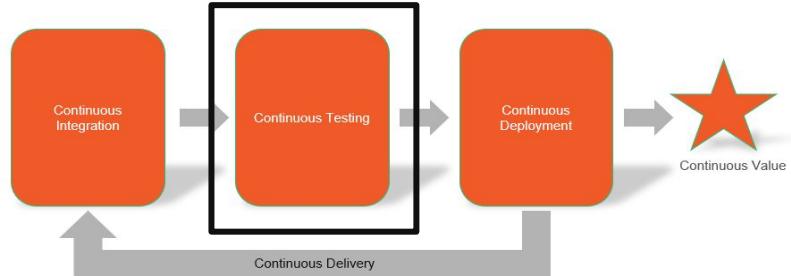
# Continuous Testing

# Continuous Testing



Common Types	Description
Unit testing	Testing discrete units of code in isolation from other code (e.g., testing a single method). Leverages mocking to stabilize other system dependencies so that tests are repeatable. Code coverage often used to assess quality & sufficiency of unit testing. Should be automated.
Regression testing	Testing that verifies that newly introduced changes for development of a feature do not inadvertently break other, existing features. Automated unit tests can also fill the role of a regression test suite nicely. Quickly confirming no breakages.
Integration testing	Testing that verifies that groups of components & component features operate correctly when combined with other features (e.g., multiple functions or modules that provide a larger “chunk” of functionality). Usually represents a layer above unit testing but below functional testing. Should be automated.
Functional testing	Testing that verifies functional requirements for critical workflows in the software end-to-end. Can be (and probably should be) implemented as automated black box tests, requiring minimal knowledge & minimal assumptions about the inner workings of the software.
Acceptance testing	Testing that verifies the functionality of the software against specific acceptance criteria defining the difference between “good” & “bad”. May include manual testing/utilization by a subset of testers to verify that the software will operate correctly when leveraged by end users in production. AKA does the software do what it is supposed to from the end user’s perspective?
Security testing	Testing that verifies the functionality of the software against critical security requirements as identified & prioritized via threat modeling. Through a combination of static & dynamic testing, assesses the software for security vulnerabilities or deficiencies from a compliance & regulatory perspective (A.K.A. DevSecOps). Examples include SQL Injection and Cross-Site Scripting.
Performance testing	Testing that verifies that the software will meet defined SLAs (Service Level Agreements) when exercised under load or stress. Can also be used to validate the software’s ability to elastically scale (out or in) based on volumes. How does the software perform under load?

# Continuous Testing

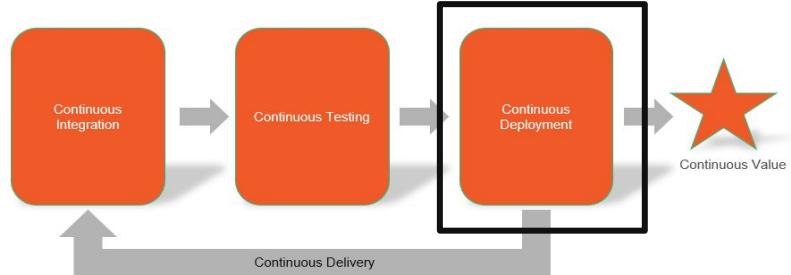


How does Continuous Integration help promote software quality?

- Maximizes the use & benefits of automation to enable “early & often”
- Provides multiple “hooks” within the workflow for plugging in the various types of testing
- Enables integration of testing results into the pipeline as quality gates
- Accelerates test result feedback for visibility & effective disposition

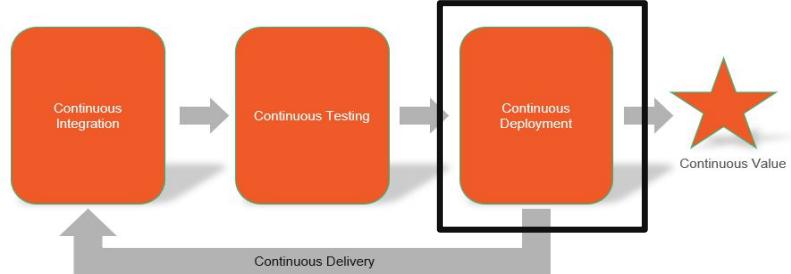
# Continuous Deployment

# Continuous Deployment



- Does not necessarily mean continuously deployed, but rather continuously *ready* to deploy
- Through a release pipeline, seeks to push latest version of the software through the environments at the frequency required by the business
- Intended to help get innovation “quickly” into the hands of users & customers for serving needs & gathering feedback

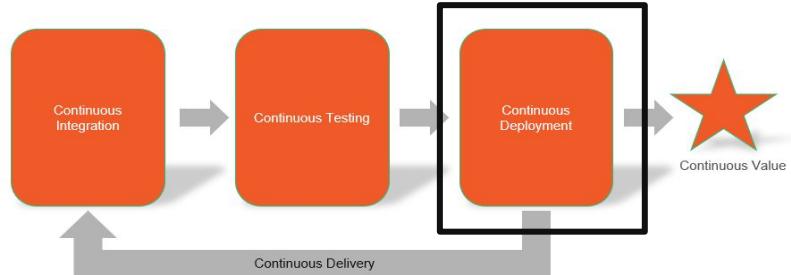
# Continuous Deployment



Version management:

- Often leverages semantic versioning (major.minor.patch)
- Often leverages explicit versioning
- Seeks to keep a clear distinction & separation between each version of the software
- Goal is version traceability & visibility when upgrade is possible, or rollback is required

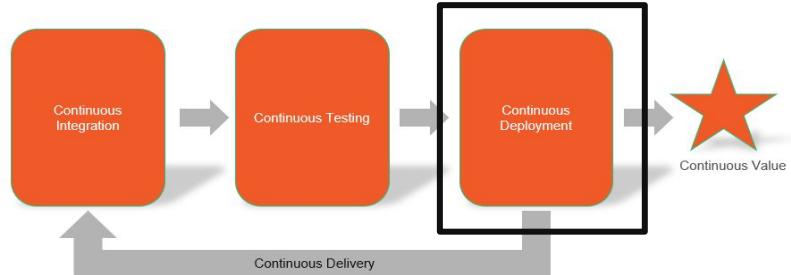
# Continuous Deployment



Blue/Green testing:

- Maintains two (or more) identical hosting environments
- Provides the opportunity to run multiple versions of the software in parallel
- Production traffic is managed through routing – switch routing from old to new, & verify
- If issues uncovered, routing can be switched back to previous version for quick rollback

# Continuous Deployment



Canary testing:

- Provides the opportunity to run multiple versions of the software in parallel
- Small portion of traffic can be routed to new version to ease in adoption
- Allows verification of new version in lower risk manner (like a canary in a coal mine)
- As confidence in new version builds, more and more traffic can be transitioned over until new version is used exclusively

## LAB: Responsible AI

Using the details of the Knight Capital Group incident (<https://dougseven.com/2014/04/17/knightmare-a-devops-cautious-tale/>) and the information reviewed in the past few slides, in your assigned breakout room, discuss the following:

- What are some of the primary issues you see with Knight Capital Group's approach (and that you believe might have led to their "downfall")?
- Where could the principles of CI/CD and DevSecOps have assisted in preventing those issues?

Nominate someone (or volunteer) to share your group's ideas.

# So, What Went Wrong?

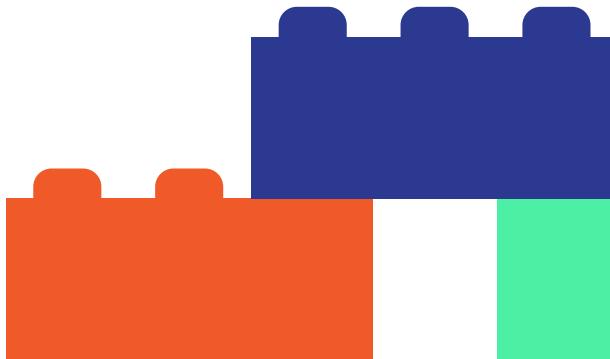
-  Repurpose of legacy code and legacy flags/variables without proper quality checks
-  Fully manual deployment
-  No “kill switch” and no clear path to rollback
-  Insufficient observability and monitoring to alert resources to issues
-  Forced to test and troubleshoot live in Production

# Building Blocks & Knight Capital Group

Software remains in a “ready to release” state enabling agility

Copious and effective monitoring alerts us to problems quickly so we can respond quickly

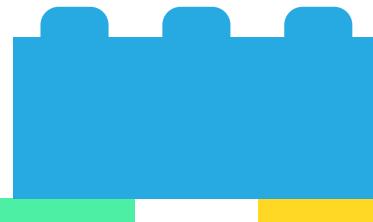
## Continuous Delivery



## Continuous Integration

Automated testing finds issues quickly and helps ensure quality/reduce fear when releasing

## Continuous Monitoring



## Continuous Deployment

Automated release at the “push of a button” reduces element of human error



## Automated Infrastructure

Automated build out of secure runtime environments (and ability to quickly roll back in the event of a discovered issue) drives stability

# AWS CodeBuild – The What & How

# Walkthrough – AWS Codebuild

## LAB: AWS CodeBuild

Execute the tutorial available at  
<https://docs.aws.amazon.com/codebuild/latest/userguide/getting-started.html>

## LAB: AWS CodeBuild – Build Badges

Execute the tutorial available at  
<https://docs.aws.amazon.com/codebuild/latest/userguide/sample-build-badges.html>

## **LAB:**

### AWS CodeBuild – Docker Sample

Execute the tutorial available at  
<https://docs.aws.amazon.com/codebuild/latest/userguide/sample-docker.html>

**DEMO**



Docker Custom Image in  
CodeBuild

# AWS CodePipeline – The What & How

## LAB: AWS CodePipeline

Execute the “Simple Pipeline Walkthrough” tutorial available at  
<https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-simple-s3.html>

## LAB: AWS CodePipeline

Execute the CloudFormation/CodePipeline tutorials available at  
<https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-cloudformation.html>

# Infrastructure as Code – a case study



# Infra as Code – Why? A case study

- Let's look at a great case study from a company called 1password to further understand why we use IaC
- Case study here:  
<https://blog.1password.com/terraform-1password/>

# IaC - 1Password Use Case



- Deployed in AWS to 3 regions:

- N. Virginia, USA us-east-1
- Montreal, Canada ca-central-1
- Frankfurt, Germany eu-central-1

- Each region has 4 environments running 1Password:

- production
- staging
- testing
- development

# IaC - 1Password Case



- 3 regions \* 4 environments per region == **12 total environments to support/maintain**
- What does an individual environment look like in terms of AWS services?

# 1password- Each Environment:



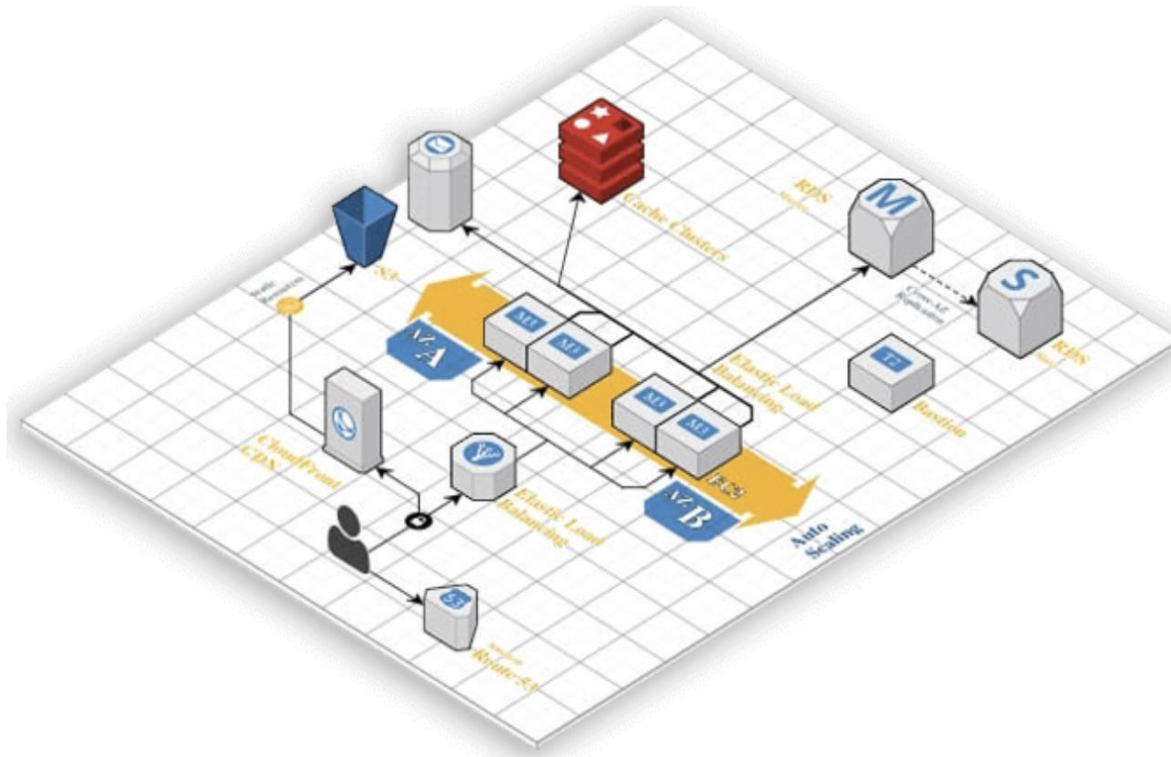
12 Environments, where each environment is comprised of these services:

- Virtual Private Cloud
- Amazon Aurora database cluster
- Caching (Redis) clusters
- Subnets
- Routing tables
- Security roles
- IAM permissions
- Auto-scaling groups
- Elastic Compute Cloud (EC2) instances
- Elastic Load Balancers (ELB)
- Route53 DNS (both internal and external)
- Amazon S3 buckets
- CloudFront distributions
- Key Management System (KMS)



# 1password- Simplified Diagram:

Services drawn out:





## IaC - 1Password Case

- Remember: 3 regions \* 4 environments per region == **12 total environments to support/maintain**
- Imagine logging into each one of these 12 environments and ***manually*** updating the infrastructure (maybe through the AWS console)
- That would be very error prone and challenging to get right!



# IaC – our hardware becomes software

- With infrastructure as code this becomes much easier – our *hardware becomes software*, which means no more manual updates to our infrastructure
- Here is some sample Terraform (what we use for IaC at CG):

```
resource "aws_subnet" "b5app" {  
    count          = "${length(var.subnet_cidr["b5app"])}"  
    vpc_id         = "${aws_vpc.b5.id}"  
    cidr_block     = "${element(var.subnet_cidr["b5app"], count.index)}"  
    availability_zone = "${var.az[count.index]}"  
  
    tags {  
        Application = "B5"  
        env         = "${var.env}"  
        type        = "${var.type}"  
        Name        = "${var.env}-b5-b5app-subnet-${count.index}"  
    }  
}
```

# IaC – our hardware becomes software

- Let's talk some more about Terraform – our IaC tool of choice at CG!



# Terraform – CG's IaC standard



# What is Terraform?

- Terraform is IaC
- Remember the Dockerfile? Which is IaC for our *containerized applications*?
- Terraform is IaC for our *cloud resources*
- With Terraform, we can provision our AWS resources with the definition of a file -



# Why Terraform?



- CloudFormation – *IaC for AWS* was released in 2011, which of course only worked with AWS
- However, the need for a *cloud-agnostic IaC provider* was still present
- So that we could provision resources across different providers such as Microsoft Azure, GCP, in addition to AWS
- Enter: **Terraform!**

# Why Terraform?



- Terraform also has a *simpler language* than CloudFormation called **HCL** (Hashicorp Configuration Language)
- Terraform is also *open source* – so it's easier to troubleshoot and understand (peek under the hood!)
- Managing services outside of AWS such as DataDog and PagerDuty are also possible

# Terraform – brief history



- ❑ Terraform (open source) released in 2014 by company called Hashicorp
- ❑ Didn't really take off until 2017

# Terraform – options for running



## Open Source

Self-managed | always free

Download

Download the open source Terraform binary and run locally or within your environments.

## Terraform Cloud

Managed Terraform

Try Terraform Cloud



Terraform Cloud enables infrastructure automation for provisioning, compliance, and management of any cloud, datacenter, and service.

## Terraform Enterprise

Terraform Enterprise is a self-hosted instance of Terraform Cloud with features like audit logging and SAML single sign-on.

hosted (TFE – Terraform

Enterprise)

- At CG, we use TFE

# Terraform – demo + lab exercise



- First a demo...
- Execute the lab here:  
<https://github.com/varoonsahgal/cg-cloud-foundations/wiki/Terraform-Lab>

# Capstone Flow



Invokes API  
which invokes  
Lambda to load  
initial todo items  
from S3



AWS Lambda



S3 Standard



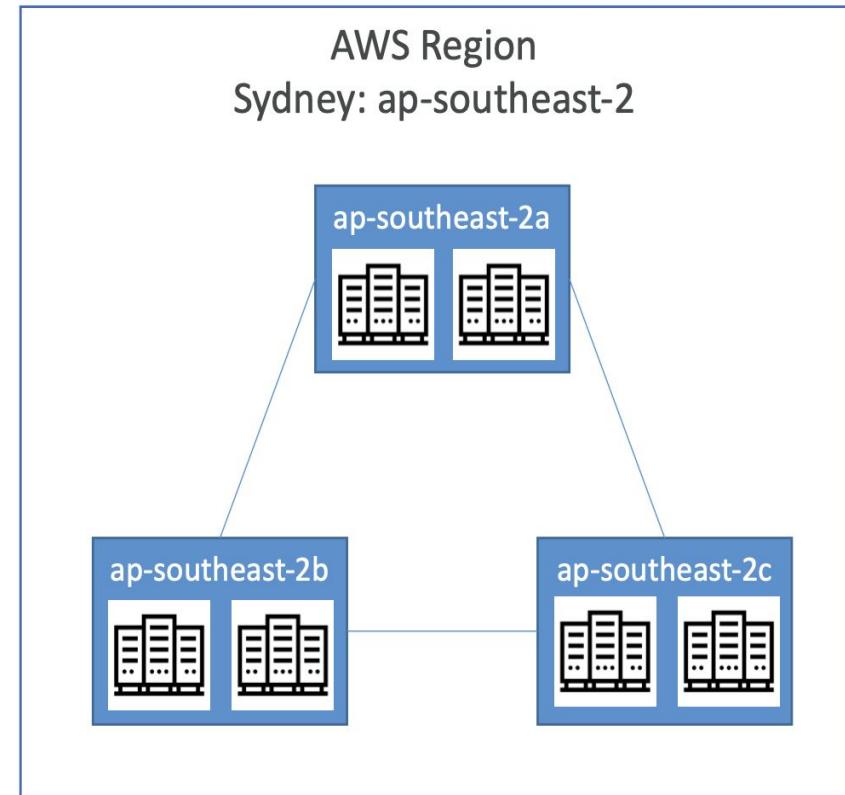
Terraform

# Architecting for HADR

# Ensuring HA for an application



- ❑ Any infrastructure component must be in at least 2 AZ's
- ❑ A few AWS services do it automatically for you, but for others you need to architect it
- ❑ For higher service quality – deploy in 2 regions



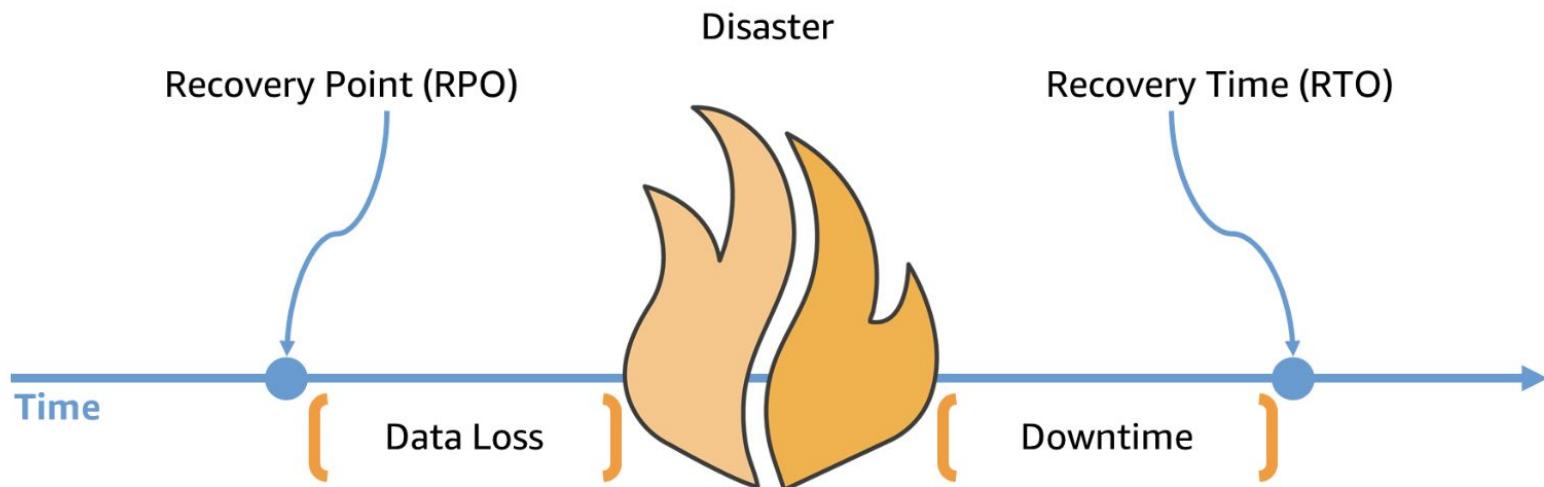
# DR Terminology

- RTO – Recovery Time Objective - The maximum acceptable delay between the interruption of service and restoration of service. This determines an acceptable length of time for service downtime.
- RPO - The maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data.
- Lower numbers represent less downtime and data loss. The tradeoff is that lower #'s == more spend on resources and operational complexity

# DR Objectives

**How much data can you afford  
to recreate or lose?**

**How quickly must you recover?  
What is the cost of downtime?**



# Capstone

# Capstone

<https://github.com/varoona-sahgal/cg-cloud-foundations/wiki/Capstone>

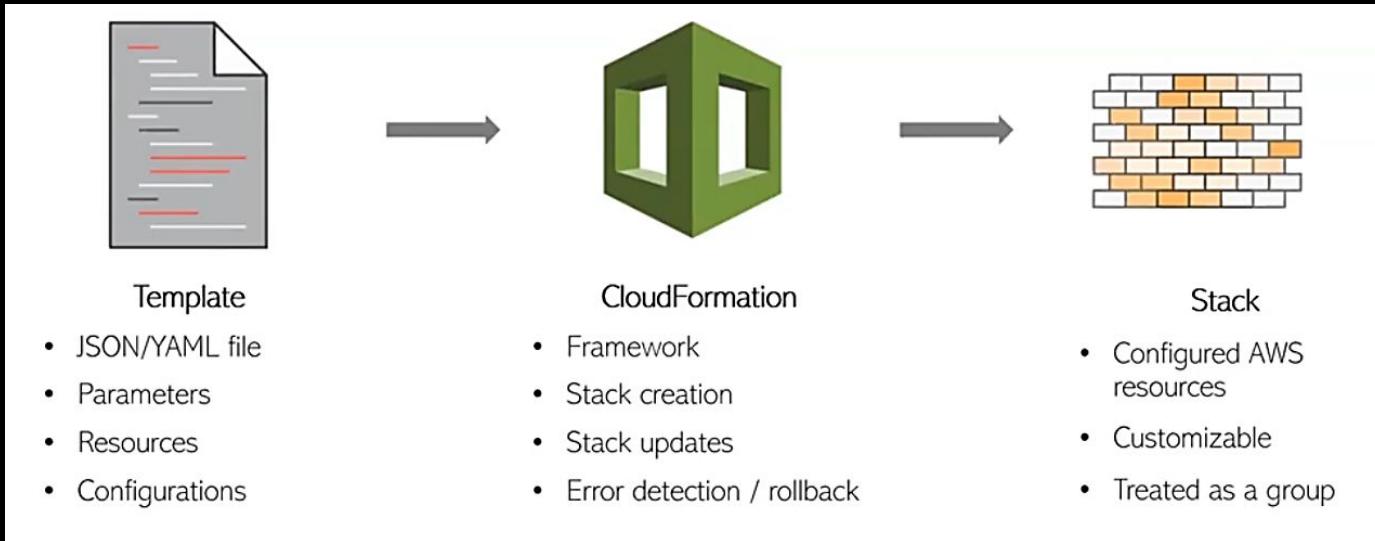
# Infrastructure as Code

- IaC is a practice in which traditional infrastructure management techniques are supplemented by or replaced with code-based tools and software development techniques.
- Cloud gives you the option to interact with infrastructure via code (simple API calls).
- What you want >> Resource
- The way you want to customize it >> Attributes

# How could it help my Organization

- Save time & bring standards (reduce manual intervention)
- Version controlled and hence ability to go back
- Foundation step to enable self-service in your organization

# The Flow



# Configuration Management Options

- Tools like Chef, Ansible, Puppet
- AWS Systems Manager
- EC2 UserData
- Managing Golden Image

# Infra as Code + Pipeline

- Pipelines are the top-level component of continuous integration, delivery, and deployment.
- Pipelines comprise:
  - Jobs, which define what to do. For example, jobs that compile or test code.
  - Stages, which define when to run the jobs.
- Gitlab Pipelines are used to trigger Terraform code
- There are multiple stages in the pipeline.
- There is a manual approval stage in this.

# Terraform lab

- Execute the lab here:  
<https://github.com/varoona-sahgal/cg-cloud-foundations/wiki/Terraform-Lab>



# Thank you!

If you have additional questions,  
please reach out to me at:  
(email address)

