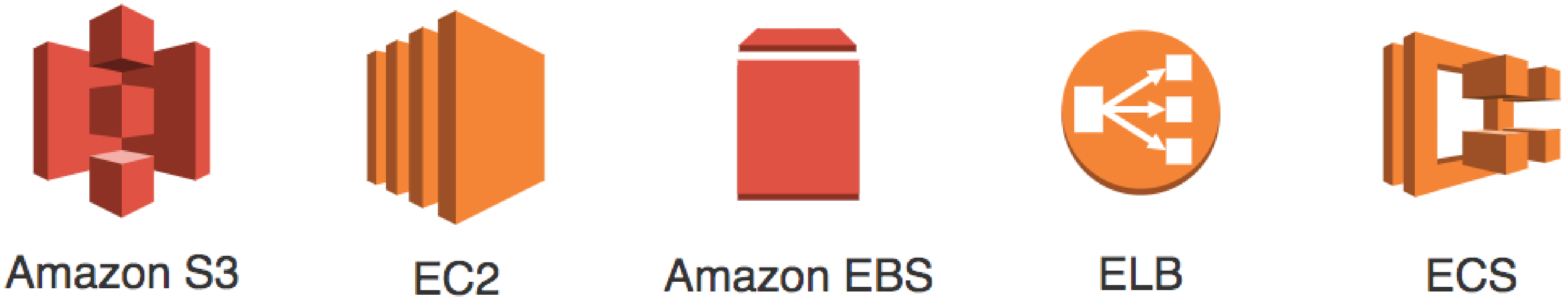
In28 Minutes

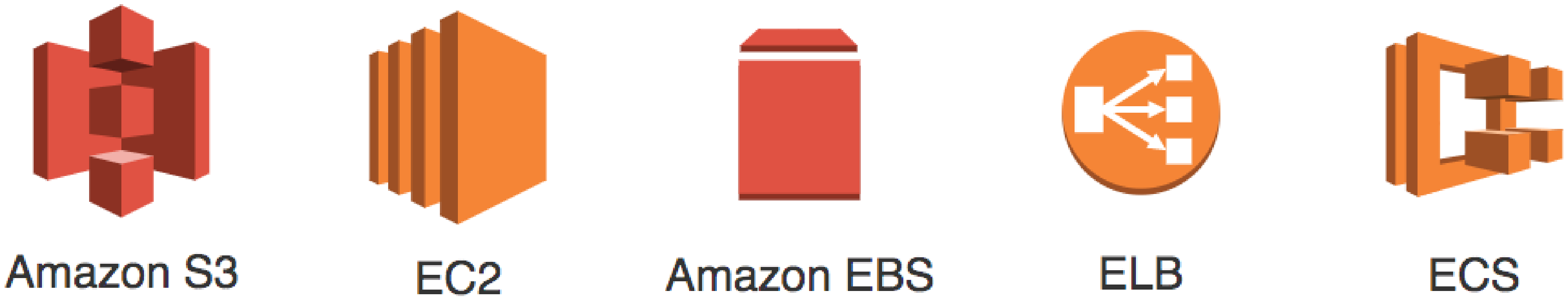
Cloud Computing with AWS

Getting Started



- Learning cloud computing is **essential**
  - For EVERYONE in IT world
- HOWEVER beginners **find the first steps a little difficult**:
  - **Varied range of terminology**: IaaS/PaaS, Private/Public/Hybrid Cloud, Serverless, Containers, Container Orchestration, Databases, Storage, DevOps, AI/ML, Datawarehouse, Data Lake ..
  - **Wide variety of services**: AWS offers 200+ services

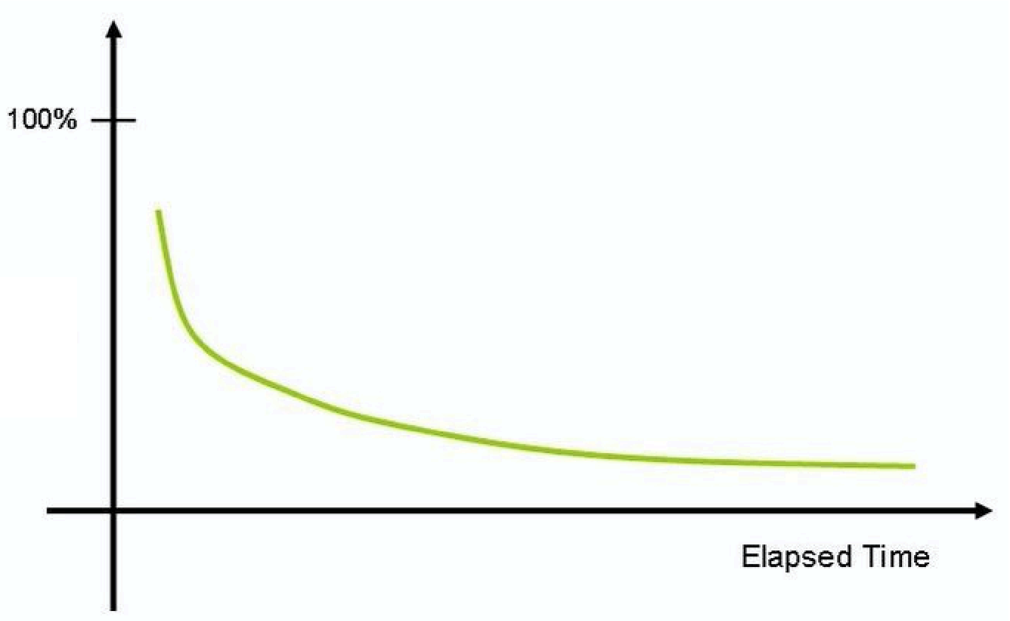
We've a Created a Simple Path to Learn AWS



- We've created a **simple path** focusing on:
  - **Fundamentals** (Understanding cloud fundamentals helps you learn AWS easily!)
- This course is **designed** for absolute beginners to the cloud
- **Our Goal** : Help you start your cloud journey with AWS

How do you put your best foot forward?

- **Learning AWS can be tricky**:
  - Lots of new terminology
  - Lots of services (200+)
- As time passes, we forget things
- How do you **improve your chances** of remembering things?
  - **Active learning** - think and make notes
  - **Review the presentation** once in a while



Our Approach

- **Videos** with:
  - Presentations &
  - Demos
- **Quizzes:**
  - To Reinforce Concepts
- (Recommended) Take your time. Do not hesitate to replay videos!
- (Recommended) Have Fun!

In28 Minutes

FASTEST ROADMAPS

in28minutes.com

Google Cloud Certifications

Azure Certifications

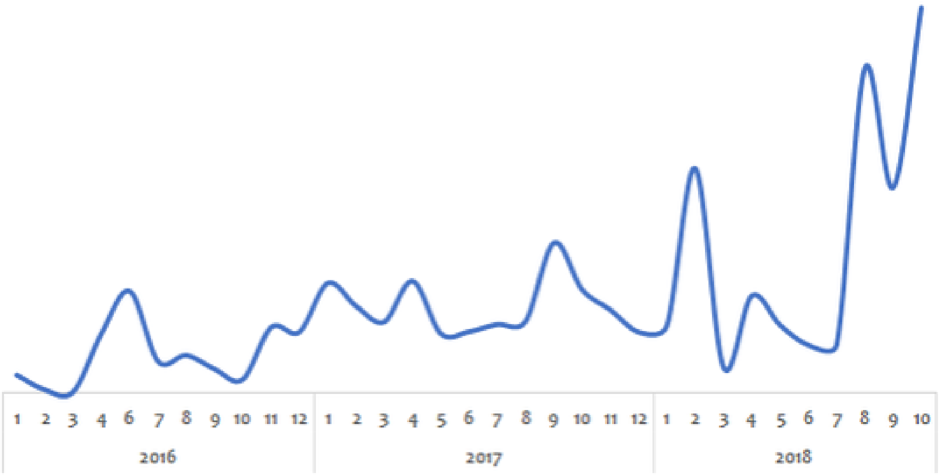
AWS Certifications

DevOps

Java Full Stack

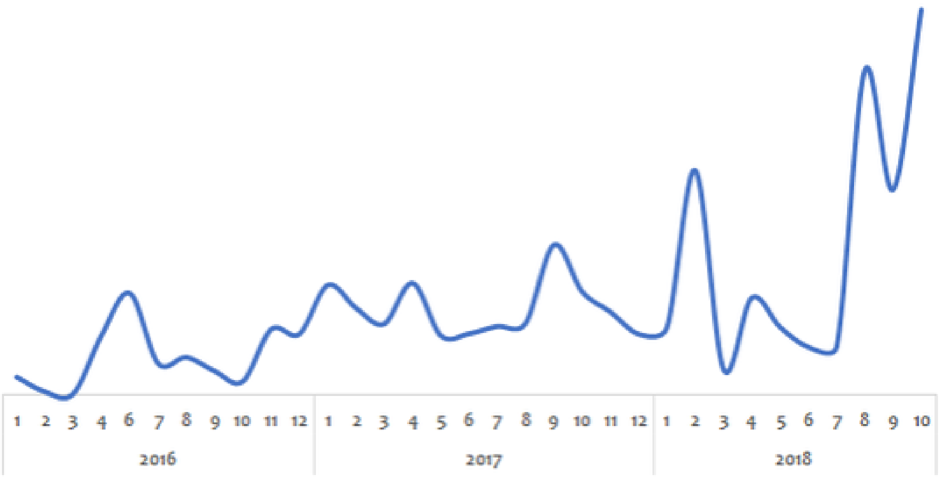
Java Microservices

Before the Cloud - Example 1 - Online Shopping App



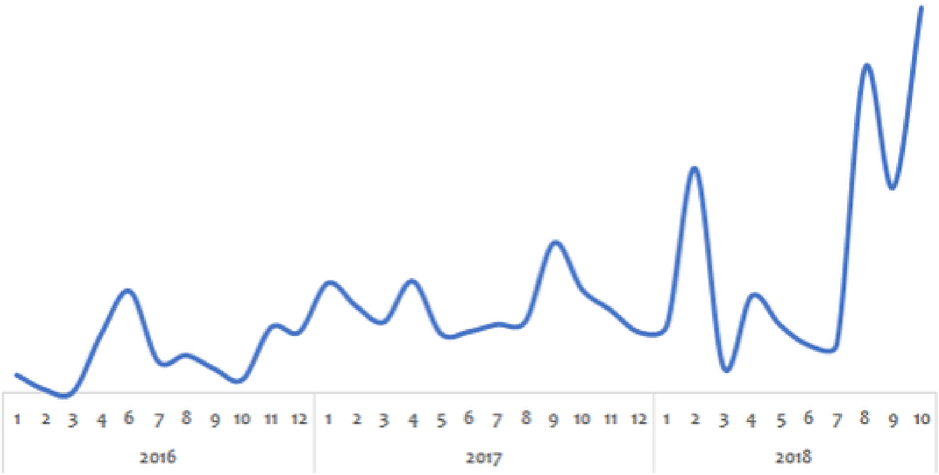
- **Challenge:**
  - **Peak usage** during holidays and weekends
  - **Less load** during rest of the time
- **Solution** (before the Cloud):
  - **Procure** (Buy) infrastructure **for peak load**
    - QUESTION: What would the infrastructure be doing during periods of low loads?

Before the Cloud - Example 2 - Startup



- **Challenge:**
  - It suddenly becomes popular.
  - How to handle the **sudden increase** in load?
- **Solution** (before the Cloud):
  - **Procure** (Buy) infrastructure assuming they would be successful
    - QUESTION: What if they are not successful?

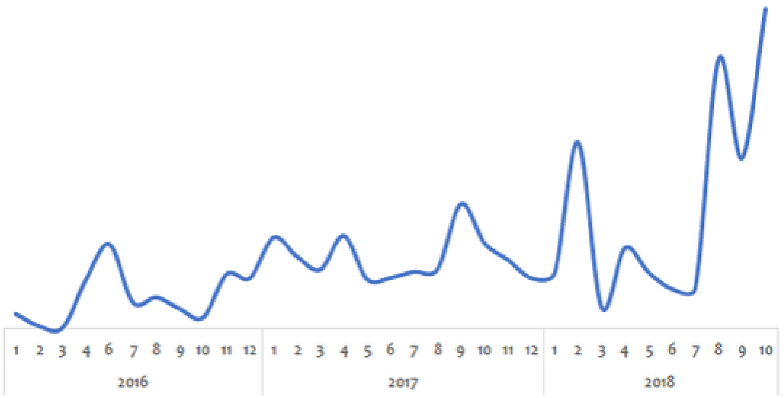
Before the Cloud - Challenges



- Low infrastructure utilization (**PEAK LOAD** provisioning)
- Needs ahead of time planning (**Can you guess the future?**)
- High cost of procuring infrastructure
- Dedicated infrastructure maintenance team (**Can a startup afford it?**)

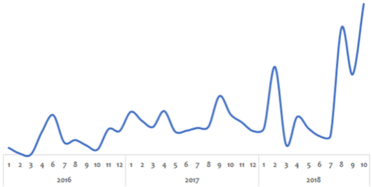
Silver Lining in the Cloud

- How about **provisioning (renting) resources** when you want them and releasing them back when you do not need them?
  - **On-demand resource provisioning**
  - Also called **Elasticity**



Cloud - Advantages

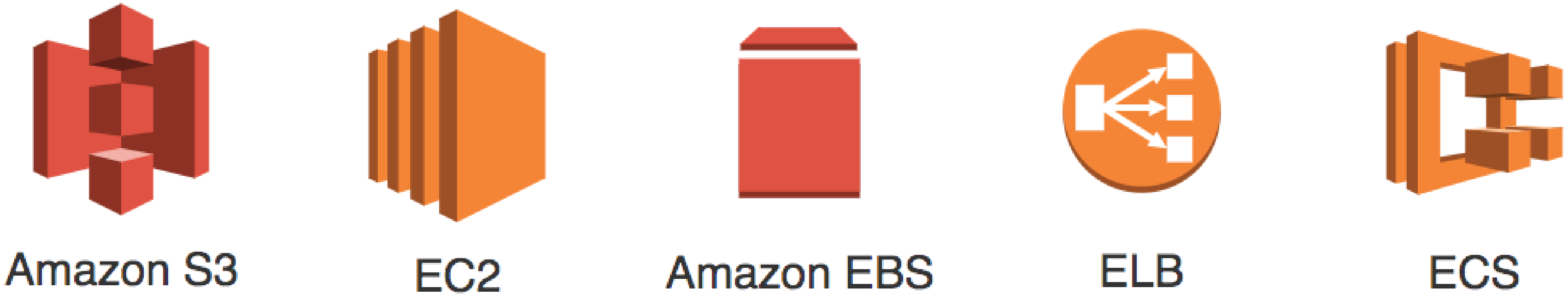
- Trade **"capital expense"** for **"variable expense"**
- Benefit from massive **economies of scale**
- Stop **guessing** capacity
- **"Go global"** in minutes
- Avoid **undifferentiated heavy lifting**
- Stop spending money running and maintaining data centers



Amazon Web Services (AWS)

- **Leading** cloud service provider
  - **Competitors:** Microsoft Azure and Google Cloud
- Provides **MOST** (200+) services
- Reliable, secure and cost-effective
- You will learn more about AWS as we go further in the course!

Best path to learn AWS!



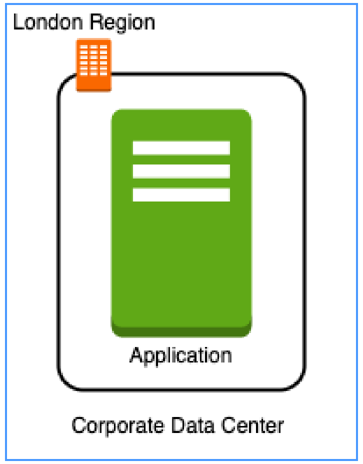
- Cloud applications make use of multiple AWS services.
- There is **no single path** to learn these services independently.
- HOWEVER, we've worked out a simple path!

Setting up AWS Account

- Create an AWS Account
- Setup an IAM user

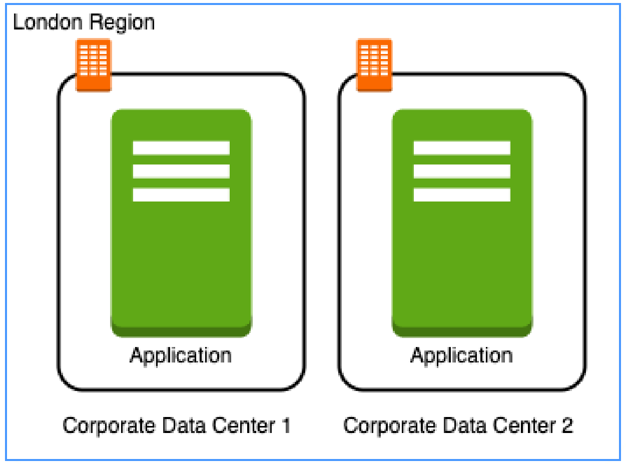
Regions and Zones

Regions and Zones



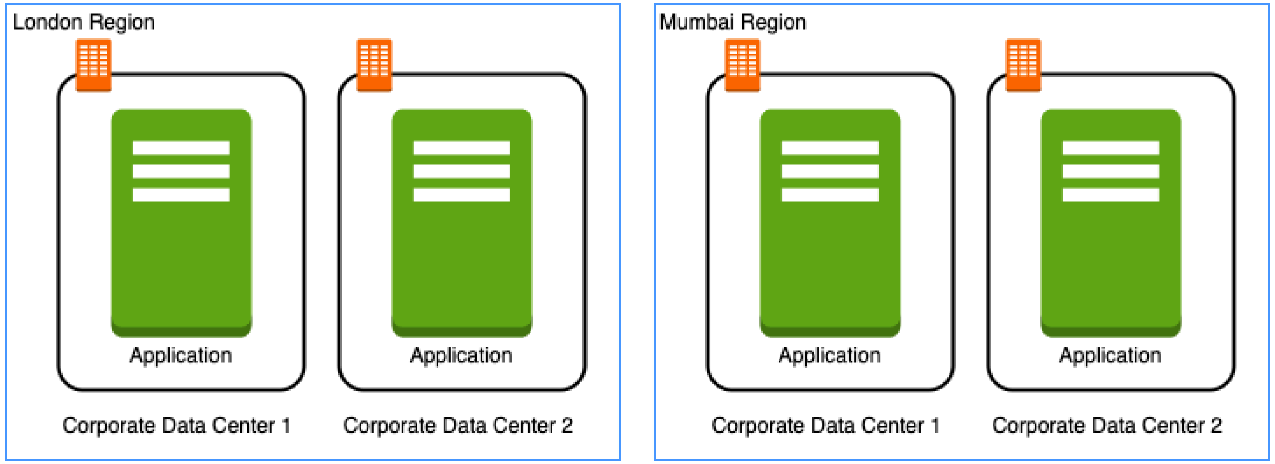
- Imagine that your application is deployed in a data center in London
- What would be the challenges?
    - Challenge 1 : Slow access for users from other parts of the world (**high latency**)
    - Challenge 2 : What if the data center crashes?
        - Your application goes down (**low availability**)

Multiple data centers



- Let's **add in one more data center** in London
- What would be the challenges?
  - Challenge 1 : Slow access for users from other parts of the world
  - Challenge 2 (**SOLVED**) : What if one data center crashes?
    - Your application is **still available** from the other data center
  - Challenge 3 : What if **entire region** of London is unavailable?
    - Your application goes down

Multiple regions



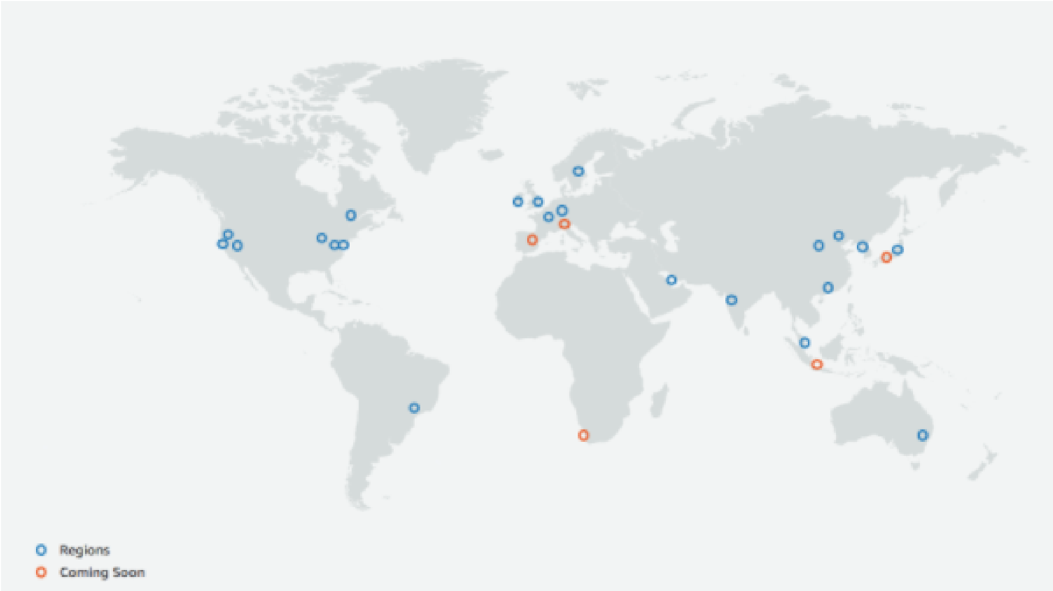
- Let's add a new region : Mumbai
- What would be the challenges?
  - Challenge 1 (**PARTLY SOLVED**) : Slow access for users from other parts of the world
    - You can solve this by adding deployments for your applications in other regions
  - Challenge 2 (SOLVED) : What if one data center crashes?
    - Your application is still live from the other data centers
  - Challenge 3 (**SOLVED**) : What if entire region of London is unavailable?
    - Your application is served from Mumbai

Regions

○ Regions
○ Coming Soon

- Imagine setting up data centers in different regions around the world
  - **Would that be easy?**
- (Solution) AWS provides **25+ regions** around the world (expanding every year)

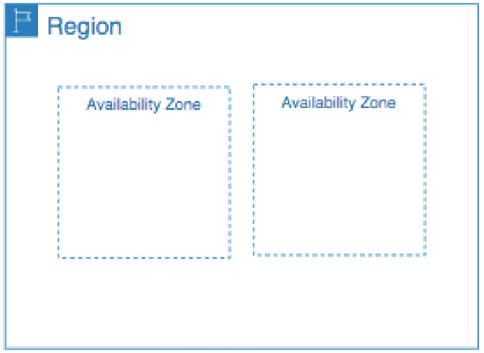
Regions - Advantages



- High Availability
- Low Latency
- Global Footprint
- Adhere to government **regulations**

Availability Zones

- Each AWS Region consists of multiple AZ's
- Each Availability Zone:
  - Can have **One or more discrete data centers**
  - has **independent** & **redundant** power, networking & connectivity
- AZs in a Region are connected through **low-latency** links
- (Advantage) **Increase availability and fault tolerance** of applications in the same region



Regions and Availability Zones examples

New Regions and AZs are constantly added

Region Code	Region	Availability Zones	Availability Zones List
us-east-1	US East (N. Virginia)	6	us-east-1a us-east-1b us-east-1c us-east-1d us-east-1e us-east-1f
eu-west-2	Europe (London)	3	eu-west-2a eu-west-2b eu-west-2c
ap-south-1	Asia Pacific(Mumbai)	3	ap-south-1a ap-south-1b ap-south-1c

EC2 Fundamentals

Introduction to EC2 (Elastic Compute Cloud)

EC2

EC2 Instances

- In corporate data centers, applications are deployed to physical servers
- Where do you deploy applications in the cloud?
  - Rent virtual machines
  - **EC2 instances** - Virtual machines in AWS
  - **EC2 service** - Provision EC2 instances or virtual machines

Understanding Important Features of EC2

EC2 Instances
ELB
Amazon EBS

- Create and manage lifecycle of EC2 instances
- **Attach storage** to your EC2 instances
- **Load balancing** for multiple EC2 instances
- **Our Goal:** Play with EC2 instances!

Reviewing Important EC2 Concepts

Feature	Explanation
Amazon Machine Image (AMI)	What operating system and what software do you want on the instance?
Instance Families	Choose the right family of hardware (General purpose or Compute/Storage/Memory optimized or GPU)
Instance Size (t2.nano, t2.micro,t2.small,t2.medium ...)	Choose the right quantity of hardware (2 vCPUs, 4GB of memory)
Elastic Block Store	Attach Disks to EC2 instances (Block Storage)
Security Group	Virtual firewall to control **incoming and outgoing** traffic to/from AWS resources (EC2 instances, databases etc)
Key pair	Public key and a private key Public key is stored in EC2 instance Private key is stored by customer

IAM & Best Practices

- **IAM**: Identity and Access Management
    - **Authentication** (the right user?) and
    - **Authorization** (the right access?)
    - **Root User**: User we created our AWS account with
        - Credentials: Email address and password
        - DO NOT user Root User for day to day activities
        - Create a new IAM User and use the IAM user for regular activities
- **Things we will do now**:
    - 1: Create an IAM Group - Developers - with admin access
    - 2: Create an IAM user - in28minutes_dev - with group Developers
    - 3: Login with IAM user - in28minutes_dev
- (Remember) Bookmark Your Account Specific AWS URL

Cloud Best Practices - Managing Costs

- With **Great Power** comes **Great Responsibility**
  - Cloud provides you with ability to create powerful resources
  - HOWEVER its important to understand the associated costs
- **5 Best Practices**
  - **1:** For the first week, monitor the billing dashboard everyday
  - **2:** Set Budget Alerts
    - 1: Enable Billing Alerts - My Billing Dashboard > Billing preferences
    - 2: Create Budget Alert - Budgets > Create a Budget > Cost Budget > Alert
  - **3:** STOP resources when you are not using them
  - **4:** Understand FREE Tier and 12 Month Limits (HARD TO DO)
  - **5:** Understand how pricing works for diff. resources (HARD TO DO)

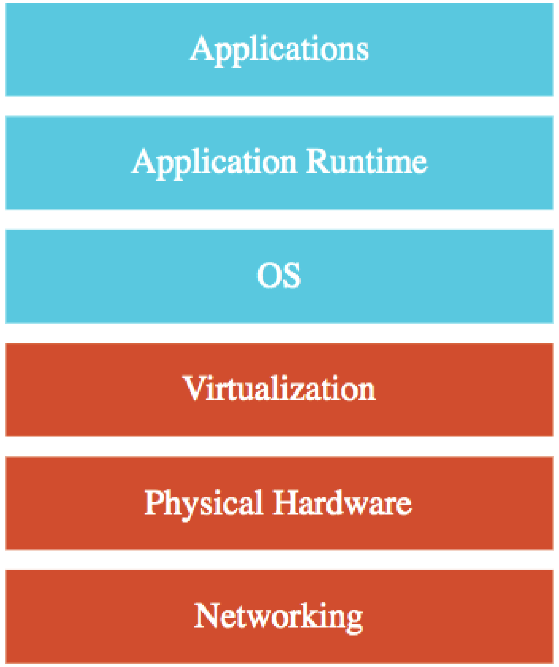
Cloud Services

Cloud Services

- Do you want to continue **running applications in the cloud**, the **same way you run them** in your **data center**?
- OR **are there OTHER approaches**?
- You should **understand some terminology**:
  - **IaaS** (Infrastructure as a Service)
  - **PaaS** (Platform as a Service)
  - ....
- Let's get on a quick **journey** to understand these!

IAAS (Infrastructure as a Service)

- Use **only infrastructure** from cloud provider
  - **Ex:** Using VM service to deploy your apps/databases
- **Cloud provider** is responsible for:
  - Hardware, Networking & Virtualization
- You are responsible for:
  - OS upgrades and patches
  - Application Code and Runtime
  - Configuring load balancing
  - Auto scaling
  - Availability
  - etc.. (and a lot of things!)



Applications

Application Runtime

OS

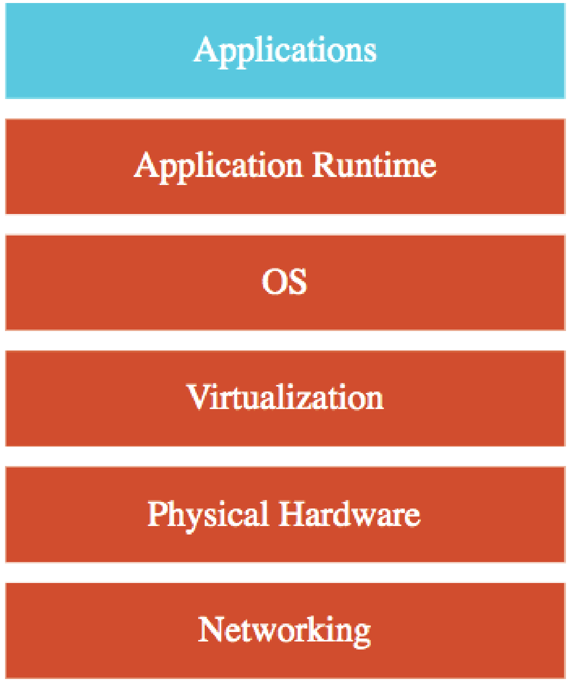
Virtualization

Physical Hardware

Networking

PAAS (Platform as a Service)

- Use a platform provided by the cloud
  - **Cloud provider** is responsible for:
    - Hardware, Networking & Virtualization
    - OS (incl. upgrades and patches)
    - Application Runtime
    - Auto scaling, Availability & Load balancing etc..
  - **You** are responsible for:
    - Configuration (of Application and Services)
    - Application code (if needed)
- **Examples:**
  - **Compute**: AWS Elastic Beanstalk, Azure App Service, Google App Engine
  - **Databases**: Relational & NoSQL (Amazon RDS, Google Cloud SQL, Azure SQL Database etc)
  - Queues, AI, ML, Operations etc!



Applications

Application Runtime

OS

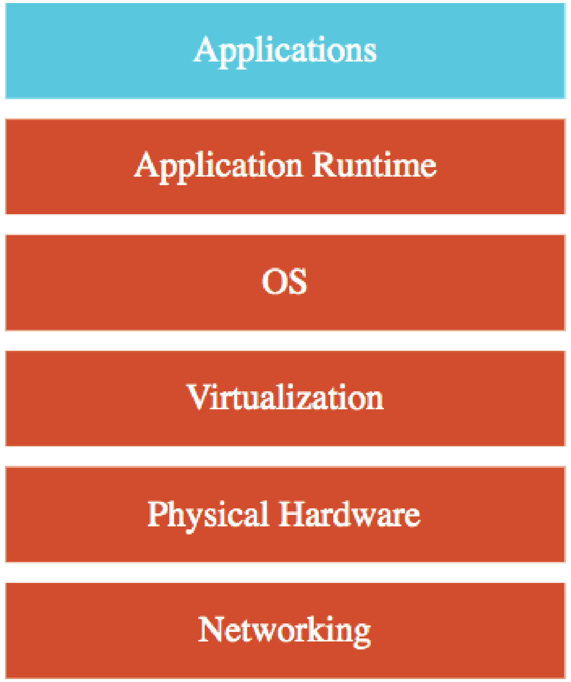
Virtualization

Physical Hardware

Networking

AWS Elastic BeanStalk

- **Simplest way** to deploy and scale your web applications in AWS
  - Provides end-to-end web application management
  - Supports Java, .NET, Node.js, PHP, Ruby, Python, Go, and Docker applications
  - **No usage charges** - Pay for AWS resources provisioned
- **Features:**
  - Automatic load balancing
  - Auto scaling
  - Managed platform updates



Applications

Application Runtime

OS

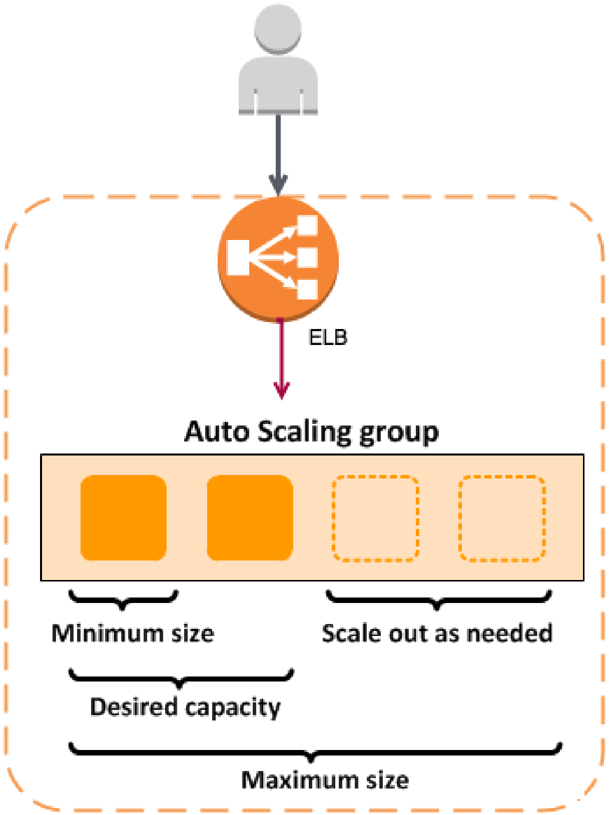
Virtualization

Physical Hardware

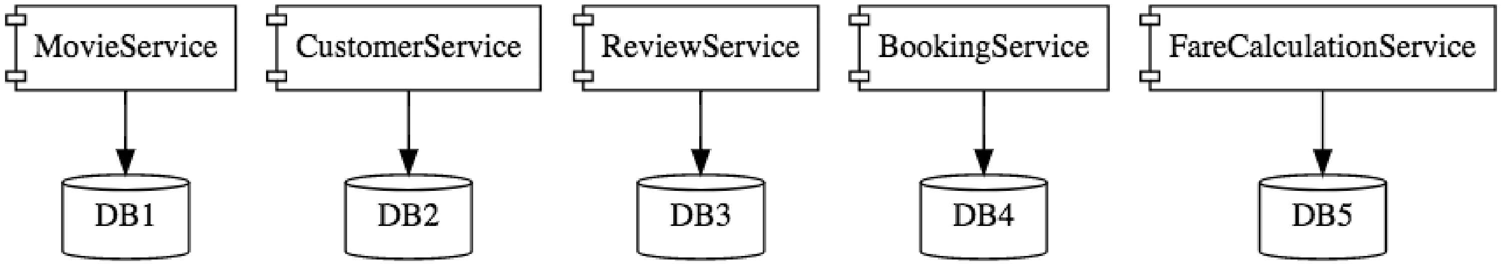
Networking

Auto Scaling Group and Elastic Load Balancing

- Applications have millions of users:
  - Same application is deployed to multiple VMs
- How do you simplify creation and management of **multiple VMs**?
  - **Auto Scaling Groups**
  - Allow you to create and manage a group of EC2 instances
- How do you distribute traffic across multiple EC2 instances?
  - **Elastic Load Balancing**



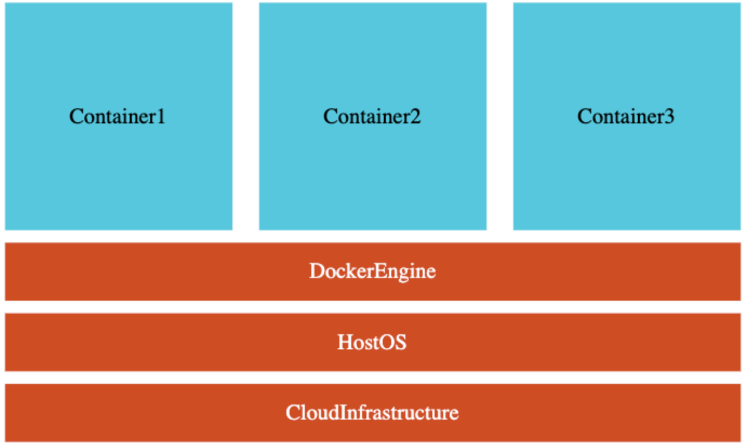
Microservices



- Enterprises are heading towards microservices architectures
  - Build small focused microservices
  - **Flexibility to innovate** and build applications in different programming languages (Go, Java, Python, JavaScript, etc)
- BUT **deployments become complex**!
- How can we have **one way of deploying** Go, Java, Python or JavaScript .. microservices?
  - Enter **containers**!

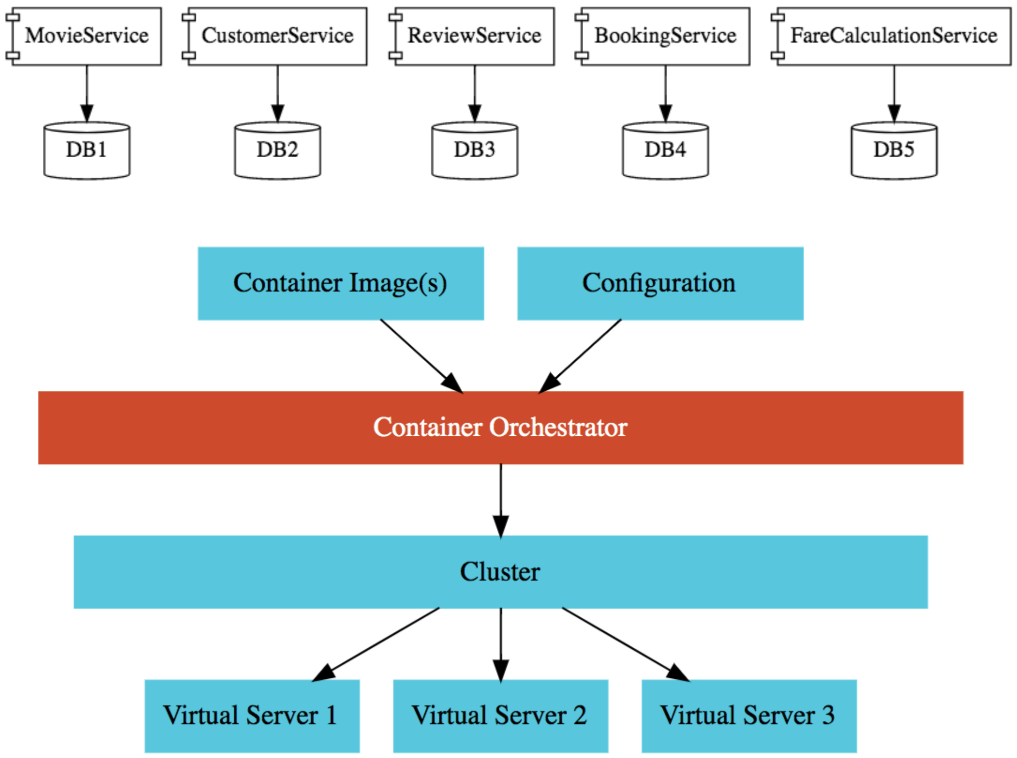
Containers - Docker

- Create **Docker images** for each microservice
- Docker image **has all needs of a microservice:**
  - Application Runtime (JDK or Python or NodeJS)
  - Application code and Dependencies
  - Runs **the same way** on any infrastructure:
    - Your local machine, Corporate data center or in the Cloud
- Advantages
  - Docker is **cloud neutral**
  - Standardization: Simplified Operations
    - Consistent deployment, monitoring, logging ...
  - Docker containers are **light weight**
    - Compared to Virtual Machines as they do not have a Guest OS
  - Docker provides **isolation** for containers



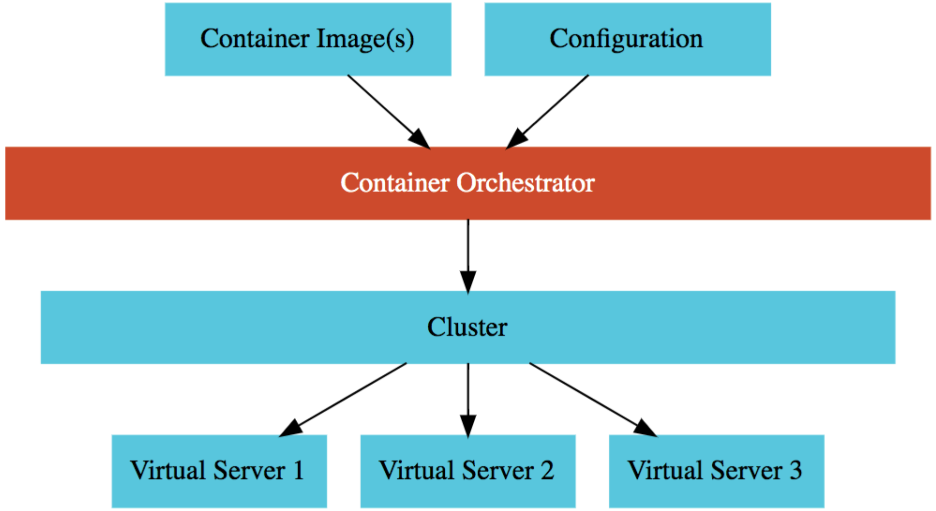
Container Orchestration

- **Requirement** : I want 10 instances of Microservice A container, 15 instances of Microservice B container and ....
- Typical Features:
  - **Auto Scaling** - Scale containers based on demand
  - **Service Discovery** - Help microservices find one another
  - **Load Balancer** - Distribute load among multiple instances of a microservice
  - **Self Healing** - Do health checks and replace failing instances
  - **Zero Downtime Deployments** - Release new versions without downtime



Container Orchestration Options

- **Cloud Neutral:** Amazon EKS
  - Kubernetes: Open source container orchestration
  - Managed service: Amazon Elastic Kubernetes Service
  - EKS does not have a free tier
- **AWS Specific**: Amazon ECS
  - Amazon Elastic Container Service
- **Fargate**: Serverless ECS/EKS
  - AWS Fargate does not have a free tier



Serverless

- What do **we think about** when we develop an application?
  - Where to deploy? What kind of server? What OS?
  - How do we take care of scaling and availability of the application?
- What if **you don't worry about servers and focus ONLY on code?**
  - Enter **Serverless**
    - Remember: **Serverless does NOT mean "No Servers"**
- **Serverless for me:**
  - You **don't worry** about infrastructure (ZERO visibility into infrastructure)
    - Flexible scaling and automated high availability
  - Most Important: **Pay for use**
    - Ideally ZERO REQUESTS => ZERO COST
- **You focus on code** and the cloud managed service takes care of all that is needed to scale your code to serve millions of requests!
  - And you pay for requests and NOT servers!

AWS Lambda

- **Truly serverless**
- You don't worry about servers or scaling or availability
- You only worry about your code
- **You pay for what you use**
  - Number of requests
  - Duration of requests
  - Memory

AWS Lambda - Supported Languages

- Java
- Go
- PowerShell
- Node.js
- C#
- Python,
- Ruby
- and a lot more...

Review - AWS Services for Compute

AWS Service Name	Description
Amazon EC2 + ELB	Traditional Approach (Virtual Servers + Load Balancing) Use when you need control over OS OR you want to run custom software
AWS Elastic Beanstalk	Simplify management of web applications and batch applications Automatically creates EC2 + ELB(load balancing and auto scaling)
Amazon Elastic Container Service (Amazon ECS)	Simplify running of microservices with Docker containers Run containers in EC2 based ECS Clusters
Amazon Elastic Kubernetes Service (Amazon EKS)	Run and scale Kubernetes clusters
AWS Fargate	Serverless version of ECS and EKS
AWS Lambda	Serverless - Do NOT worry about servers

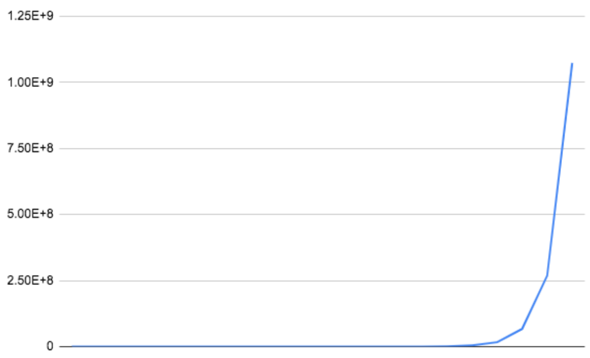
AWS Compute Services - Scenarios

Scenario	Solution
You want to run a serverless function in response to events	AWS Lambda
You want to deploy a Python application using a Managed Service	AWS Elastic Beanstalk
You want to quickly setup a Kubernetes Cluster	Amazon Elastic Kubernetes Service (Amazon EKS)
You want to setup a complex microservices architecture	Amazon Elastic Container Service (Amazon ECS) or Amazon Elastic Kubernetes Service (Amazon EKS)
Your application needs customized OS and custom Software installed	Amazon EC2

Data

Data

- Data is the **"oil of the 21st Century Digital Economy"**
- Amount of data generated **increasing exponentially**
  - Mobile devices, IOT devices, application metrics etc
  - **Variety of**
    - **Data formats**: Structured, Semi Structured and Unstructured
    - **Data store options**: Relational databases, NoSQL databases, Analytical databases, Object/Block/File storage ...
- Store data efficiently and gain intelligence
- **Goal**: Help you choose specific data format and the data store for your use case



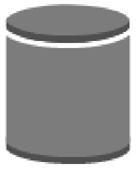
Data Formats & Data Stores

- **Data formats:**
  - **Structured**: Tables, Rows and Columns (Relational)
  - **Semi Structured**: Key-Value, Document (JSON), Graph, etc
  - **Unstructured**: Video, Audio, Image, Text files, Binary files ...
- **Data stores:**
  - Relational databases
  - NoSQL databases
  - Analytical databases
  - Object/Block/File storage

1.25E+9
1.00E+9
7.50E+8
5.00E+8
2.50E+8
0

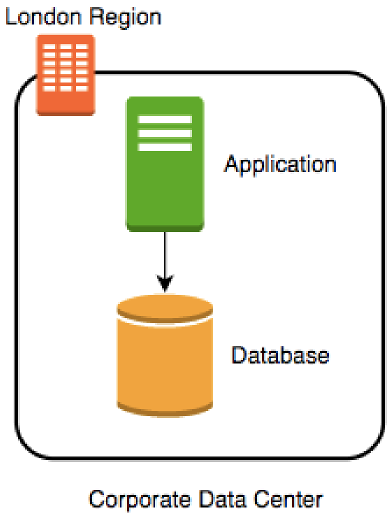
Data Stores Primer

- Data stores provide **organized** and **persistent** storage for your data
- To **choose between different data stores**, we would need to understand:
  - Availability
  - Durability
  - Scalability
  - Consistency
  - Transactions etc
- Let's start on a **simple journey** to explore these



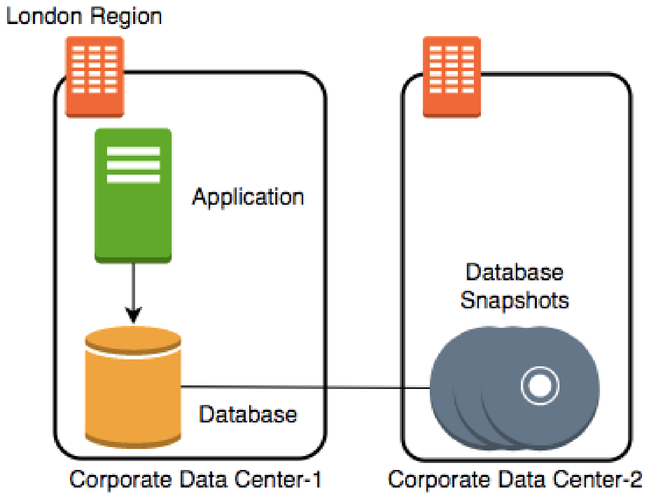
Database - Getting Started

- Imagine a database deployed **in a data center in London**
- Let's consider some challenges:
  - **Challenge 1**: Your database will go down if the data center crashes or the server storage fails
  - **Challenge 2**: You will lose data if the database crashes



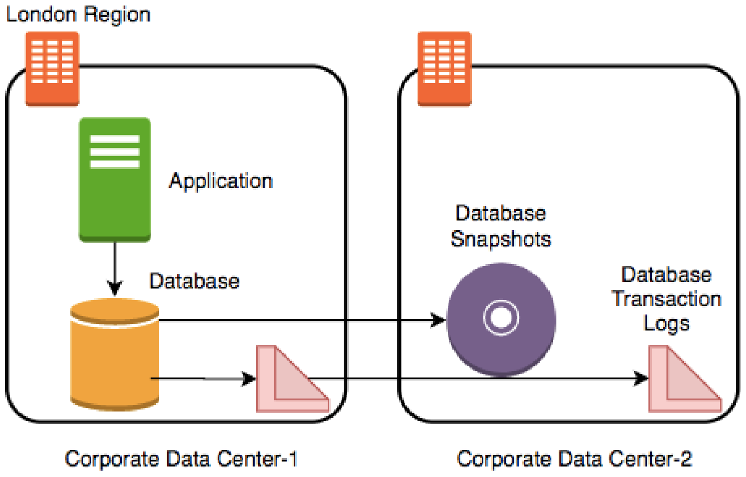
Database - Snapshots

- Let's automate taking copy of the database (**take a snapshot**) every hour to another data center in London
- Let's consider some challenges:
  - **Challenge 1**: Your database will go down if the data center crashes
  - **Challenge 2** (PARTIALLY SOLVED): You will lose data if the database crashes
    - You can setup database from latest snapshot. But depending on when failure occurs you can lose up to an hour of data
  - **Challenge 3**(NEW): Database will be slow when you take snapshots



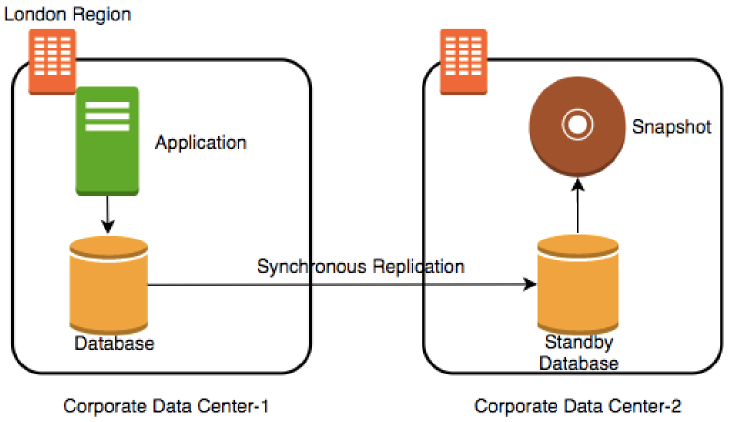
Database - Transaction Logs

- Let's add **transaction logs** to database and create a **process to copy it over** to the second data center
- Let's consider some challenges:
  - **Challenge 1**: Your database will go down if the data center crashes
  - **Challenge 2** (SOLVED): You will lose data if the database crashes
    - You can setup database from latest snapshot and apply transaction logs
  - **Challenge 3**: Database will be slow when you take snapshots



Database - Add a Standby

- Let's add a **standby database** in the second data center with replication
- Let's consider some challenges:
  - **Challenge 1** (SOLVED): Your database will go down if the data center crashes
    - You can switch to the standby database
  - **Challenge 2** (SOLVED): You will lose data if the database crashes
  - **Challenge 3** (SOLVED): Database will be slow when you take snapshots
    - Take snapshots from standby
    - Applications connecting to master will get good performance always



Availability and Durability

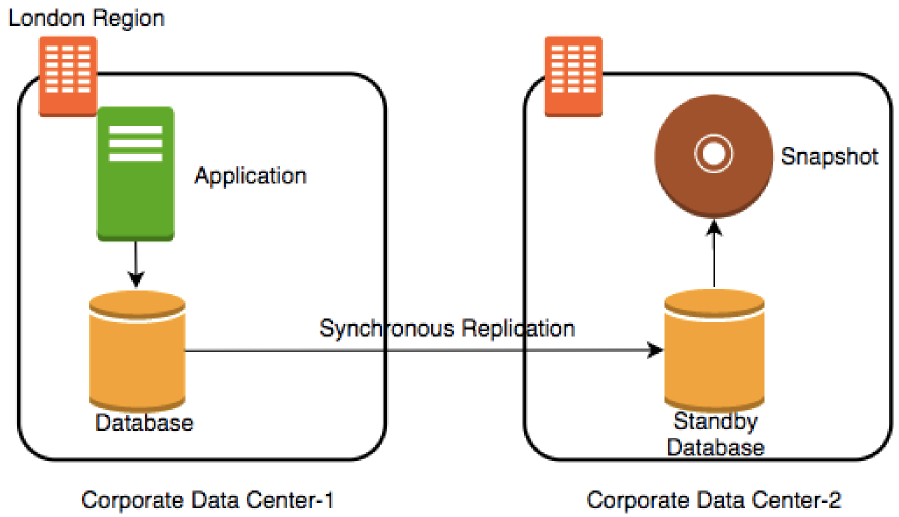
- **Availability**
  - Will I be able to access my data now and when I need it?
  - Percentage of time an application provides the operations expected of it
- **Durability**
  - Will my data be available after 10 or 100 or 1000 years?
- Examples of measuring availability and durability:
  - 4 9's - 99.99
  - 11 9's - 99.999999999
- Typically, an **availability of four 9's** is considered very good
- Typically, a **durability of eleven 9's** is considered very good

Availability

Availability	Downtime (in a month)	Comment
99.95%	22 minutes	
99.99% (4 9's)	4 and 1/2 minutes	Typically online apps aim for 99.99% (4 9's) availability
99.999% (5 9's)	26 seconds	Achieving 5 9's availability is tough

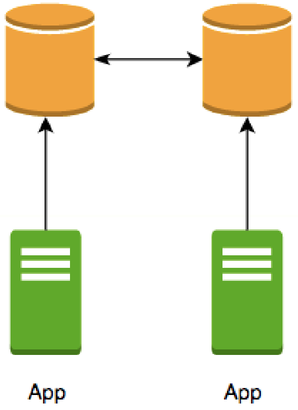
Durability

- What does a **durability of 11 9's** mean?
  - If you **store one million files for ten million years**, you would expect to **lose one file**
- Why should durability be high?
  - Because **we hate losing data**
  - Once we lose data, it is gone



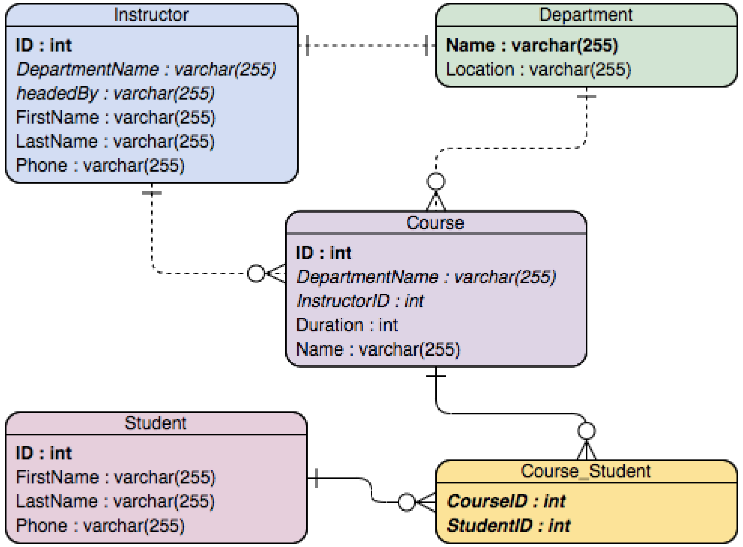
Consistency

- Creating replicas => high availability & durability
- **Consistency**: Do you get the most recent, updated data irrespective of the copy you are querying against?
  - Having replicas of data makes consistency a challenge!
  - **Strong Consistency**: Changes immediately replicated to all replicas
    - You get same data from all replicas
    - Guaranteeing Strong Consistency with Multiple replicas => Slow inserts/updates
    - Needed in most transactional applications (banking, finance, ...)!
  - **Eventual Consistency**: A little lag - few seconds - before the change is available in all replicas
    - In the intermediate period, different replicas might return different values
    - Used when **scalability is MORE important than data integrity**
    - Ex: Social Media - Facebook status messages, Twitter tweets, LinkedIn posts etc



Atomic Transactions

- Atomic Transaction
  - **Transaction**: A unit of work
    - Involve multiple steps
    - Example: Transfer $10 from A to B
    - Step 1: Deduct $10 from A's account
    - Step 2: Add $10 to B's account
  - **Atomicity**: All or nothing changes
    - Across multiple rows/multiple tables
- **Easier** if all data is stored on one node:
  - Typical relational (SQL) databases stored data on one node => Strong support for atomic transactions
  - In most NoSQL databases, data is distributed across multiple nodes => Limited support or expensive atomic transaction support



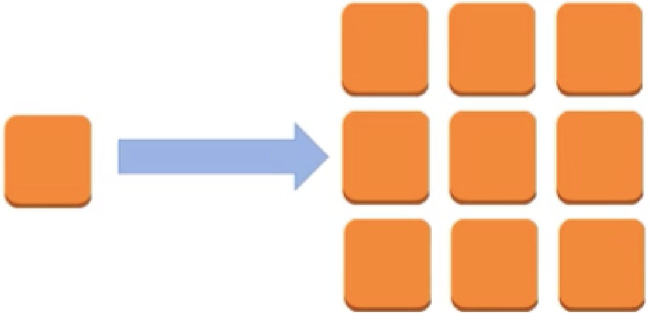
Instructor
ID : int
DepartmentName : varchar(255)
headedBy : varchar(255)
FirstName : varchar(255)
LastName : varchar(255)
Phone : varchar(255)

Department
Name : varchar(255)
Location : varchar(255)

Course
ID : int
DepartmentName : varchar(255)
InstructorID : int
Duration : int
Name : varchar(255)

Student
ID : int
FirstName : varchar(255)
LastName : varchar(255)
Phone : varchar(255)

Course_Student
CourseID : int
StudentID : int

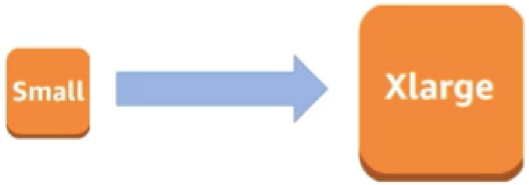


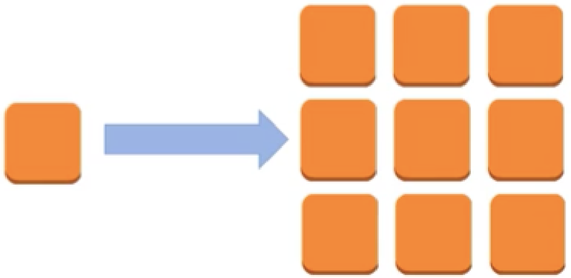
Scalability

- A system is handling 1000 transactions per second
- **10 times load** is expected next month
  - Can we handle a **growth in users, traffic, or data size** without any drop in performance?
  - Does ability to serve more growth increase **proportionally** with resources?
- Ability to **adapt** to changes in demand (users, data)
- What are the options that can be considered?
  - Deploy to a bigger instance with more CPU, memory, ...
  - Create more instances
  - ...

Vertical Scaling vs Horizontal Scaling

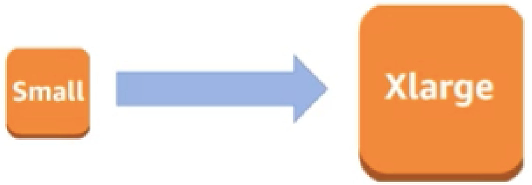
- **Vertical Scaling**: Deploying to **bigger instance**
  - A larger hard drive or A faster CPU
  - More RAM, I/O, or networking capabilities
- **Horizontal Scaling**: Using **multiple instances**:
  - Example: Use multiple VMs
  - Example: Split database data in multiple nodes
- **Key Observations**:
  - Vertical scaling has limits
  - Vertical scaling can be expensive
  - Horizontal Scaling can be complex:
    - Compute: Load Balancers etc.
    - Databases: How to split data? (and a lot of other questions!)



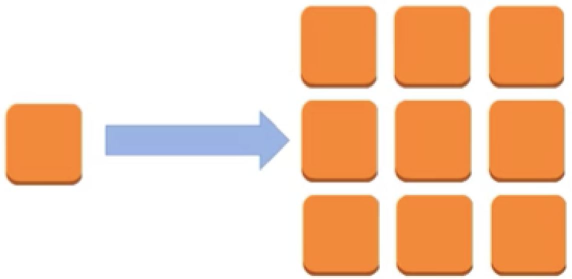


Vertical Scaling vs Horizontal Scaling for Databases

- **Vertical Scaling**: Deploying to **bigger instance**
  - Limits on max amount of data - ex: 64TB
  - Typical relational databases (MySQL etc.) only support vertical scaling for writes

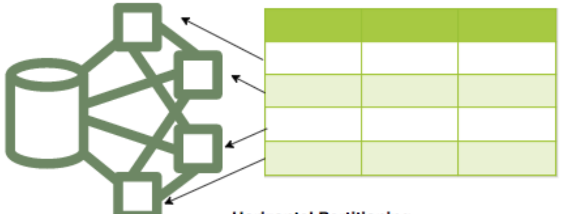


- **Horizontal Scaling**: Using **multiple instances**
  - Most NoSQL databases scale horizontally
  - NOT all NoSQL databases provide Strong Consistency
  - Most NoSQL databases provide limited (or expensive) support for atomic transactions
  - Some specially designed modern relational databases support worldwide horizontal scaling and strong transactions (Amazon Aurora, Cloud Spanner)

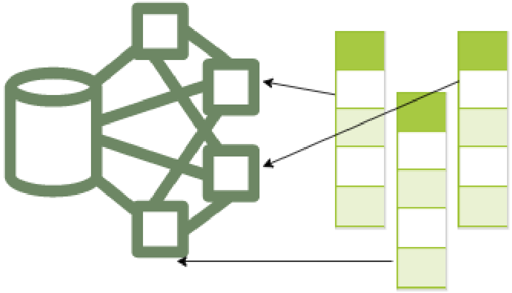


Horizontal Partitioning vs Vertical Partitioning

- **Partitioning**: Divide Tables => Many Small Parts
  - **Horizontal Partitioning**: Rows distributed across partitions
    - (Typically) A partition key used to distribute rows
    - Great for online apps (social media, for example) needing huge scale
      - **Quick updates**: Entire row stored in a partition
      - **Scales well**: No limits on how much data can be stored
    - Multi row transactions can be expensive:
      - Data divided in multiple partitions
  - **Vertical Partitioning**: Data distributed by columns
    - Each table column is stored together
    - Great for analytics and data warehouses
      - **High compression** - Store petabytes of data efficiently
      - **Faster Queries** - Execution split across multiple nodes
    - NOT so great for online/transactional apps
      - **Slow Updates** - Updating single row involves updating multiple nodes



Horizontal Partitioning



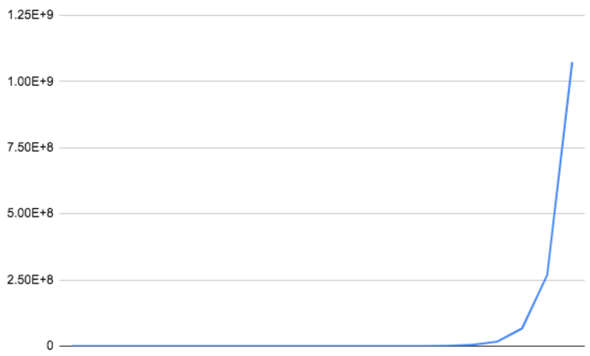
Vertical Partitioning

Exploring Data Stores - Check Your Understanding

Scenario	Solution
What is good Availability?	99.99% (4 9's) - 4 and 1/2 minutes of downtime in a month 99.999% (5 9's) - 26 seconds of downtime in a month
What is good Durability?	Typically, 11 9's durability (99.999999999%) is expected. Once we lose data, it is gone.
What are we measuring here? "Do you get the most recent, updated data irrespective of the copy you are querying against?"	Consistency
What are some of challenges with scaling Databases horizontally?	Ensuring Strong Consistency. Supporting Atomic Transactions.
What is Eventual Consistency?	A little lag - few seconds - before updates/inserts are available in all replicas. In the intermediate period, different replicas might return different values

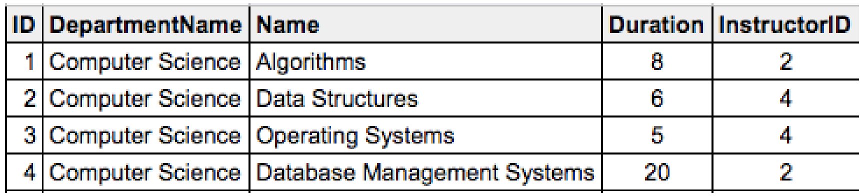
Data Formats & Data Stores

- **Data formats:**
    - **Structured: Tables, Rows and Columns (Relational)**
    - Semi Structured: Key-Value, Document (JSON), Graph, etc
    - Unstructured: Video, Audio, Image, Text files, Binary files ...
- **Data stores:**
    - **Relational databases**
    - NoSQL databases
    - **Analytical databases**
    - Object/Block/File storage

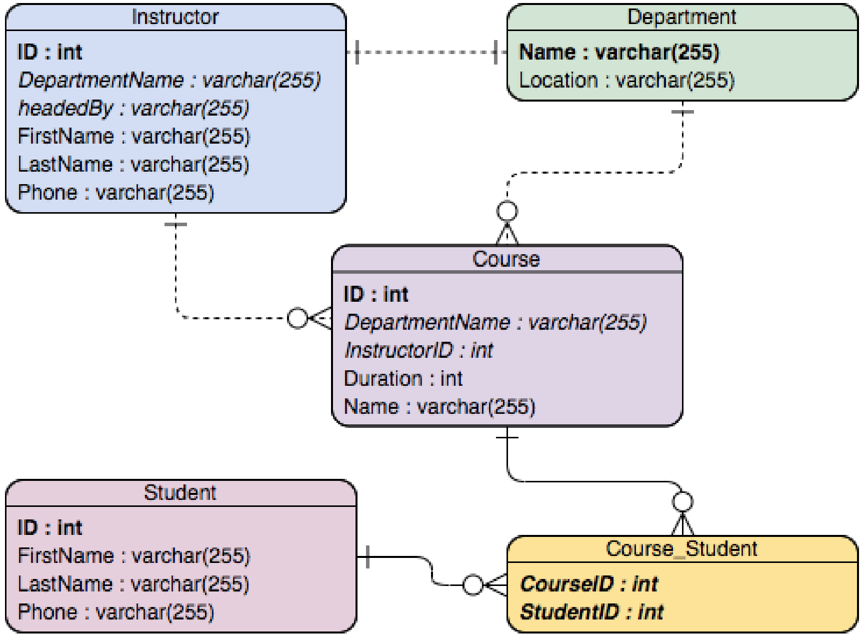


Structured Data - Relational Databases

- **Data stored in Tables** - Rows & Columns
- **Predefined schema** - Tables, Relationships and Constraints
- **Define indexes** - Query efficiently on all columns
- Used for
  - OLTP (Online Transaction Processing) use cases and
  - OLAP (Online Analytics Processing) use cases



ID	DepartmentName	Name	Duration	InstructorID
1	Computer Science	Algorithms	8	2
2	Computer Science	Data Structures	6	4
3	Computer Science	Operating Systems	5	4
4	Computer Science	Database Management Systems	20	2



Instructor
ID : int
DepartmentName : varchar(255)
headedBy : varchar(255)
FirstName : varchar(255)
LastName : varchar(255)
Phone : varchar(255)

Department
Name : varchar(255)
Location : varchar(255)

Course
ID : int
DepartmentName : varchar(255)
InstructorID : int
Duration : int
Name : varchar(255)

Student
ID : int
FirstName : varchar(255)
LastName : varchar(255)
Phone : varchar(255)

Course_Student
CourseID : int
StudentID : int

Relational DB - OLTP (Online Transaction Processing)

- **OLTP**: Applications where **large number of users make large number of transactions**
  - Transaction - small, discrete, unit of work (Ex: Transfer money to your friend's account)
  - Heavy writes and moderate reads
  - Quick processing expected
- **Use cases**: Most traditional applications - banking, e-commerce, ..
- Cloud **Managed Services**:
  - **AWS**: Amazon RDS (Aurora/PostgreSQL/MySQL/MariaDB/Oracle/SQLServer)
  - **Azure**: Azure SQL Database (SQL Server), Azure Database for MySQL/PostgreSQL/MariaDB
  - **Google Cloud**: Cloud SQL (MySQL/PostgreSQL/SQL Server), Cloud Spanner

SQL Database

Azure Database PostgreSQL

Cloud SQL

Cloud Spanner

Amazon RDS

Relational DB - OLAP (Online Analytics Processing)

- **OLAP**: Applications allowing users to **analyze/query petabytes of data**
  - **Examples**: Reporting applications, Data warehouses, Business intelligence applications, Analytics systems
  - Data is consolidated from multiple (typically transactional) databases
  - **Sample application** : Decide insurance premiums analyzing data from last hundred years
- **Cloud** Managed Services:
  - Amazon Redshift
  - Azure Synapse Analytics
  - BigQuery (Google Cloud)
- Manage petabytes of data and run queries efficiently

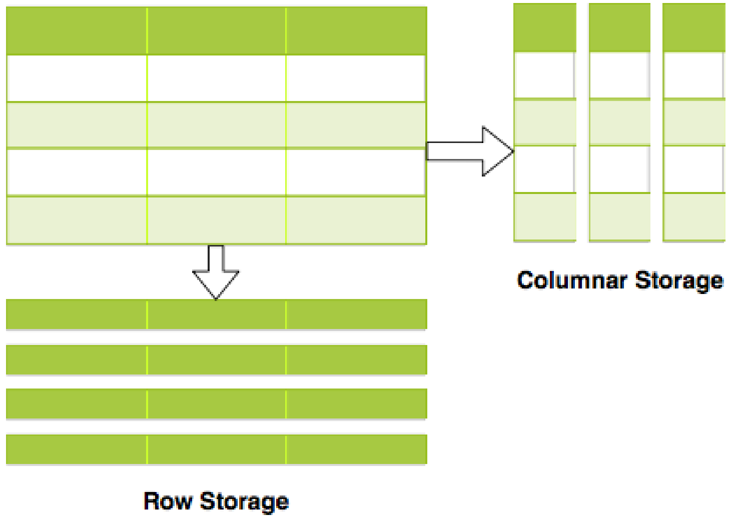
Synapse Analytics

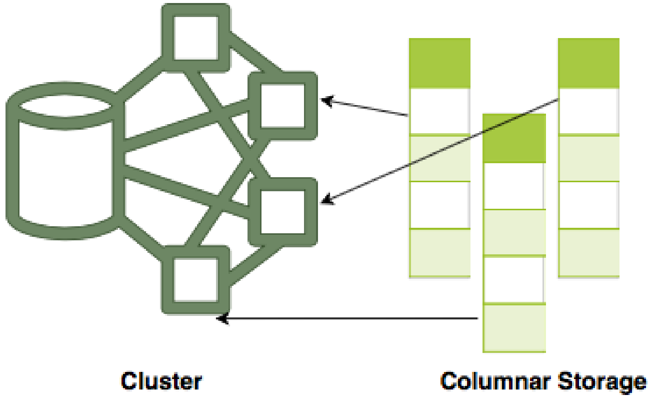
Redshift

BigQuery

Relational Databases - OLAP vs OLTP

- OLAP and OLTP use **similar data structures**
- BUT **very different approach in how data is stored**
- **OLTP databases** use row storage
  - Each table row is stored together
  - Efficient for processing small transactions
- **OLAP databases** use columnar storage
  - Each table column is stored together
  - **High compression** - store petabytes of data efficiently
  - **Distribute data** - one table in multiple cluster nodes
  - **Execute single query across multiple nodes** - Complex queries can be executed efficiently





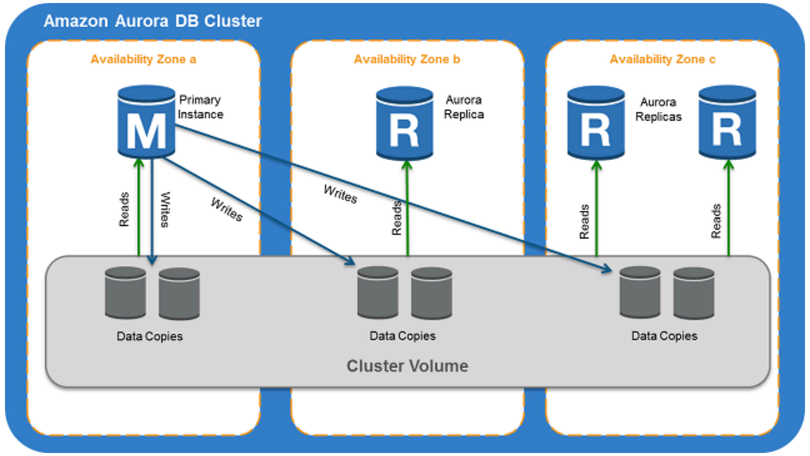
Amazon RDS (Relational Database Service)

- **Managed relational database service** for OLTP use cases
  - Manages setup, backup, scaling, replication and patching
  - Supports Aurora, PostgreSQL, MySQL (InnoDB storage engine full supported), MariaDB (Enhanced MySQL), Oracle Database and Microsoft SQL Server
  - **Features**:
    - Multi-AZ deployment (standby in another AZ)
    - Read replicas (Same AZ or Multi AZ (Availability+) or Cross Region(Availability++))
    - Storage auto scaling (up to a configured limit)
    - Automated backups (restore to point in time)
    - Manual snapshots
  - **Responsibilities:**
    - AWS is responsible for:
      - Availability, Scaling (based on your cnfgn.), Durability, Maintenance (patches) and Backups
    - You are responsible for:
      - Managing database users
      - App optimization (tables, indexes etc)

Amazon RDS

Amazon Aurora

- **MySQL and PostgreSQL**-compatible
- Uses **cluster volume** (multi AZ storage)
  - **2 copies** of data each in a **minimum of 3 AZ**
  - Up to 15 read replicas
- Provides **"Global Database"** option
  - Up to five read-only, secondary AWS Regions
    - Low latency for global reads
    - Safe from region-wide outages
  - Minimal lag time, typically less than 1 second
- Deployment Options
  - Single master (One writer and multiple readers)
  - Multi master deployment (multiple writers)
  - Serverless



https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGu

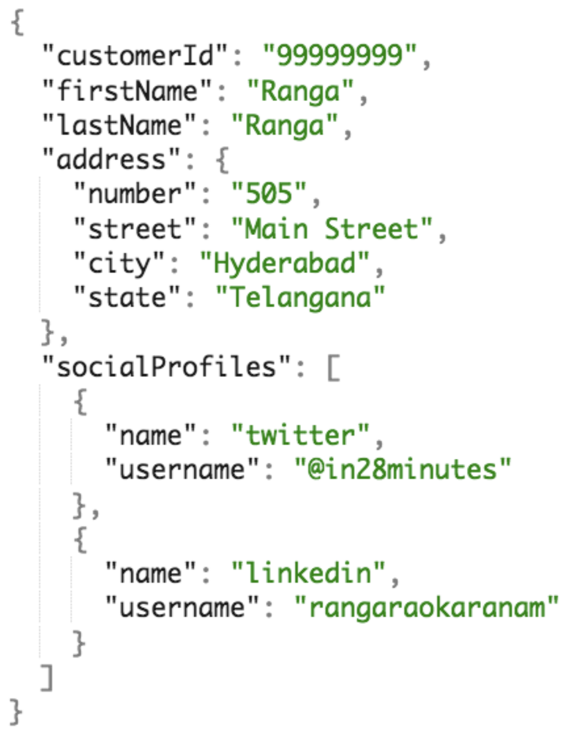
Data Formats & Data Stores

- **Data formats:**
  - Structured: Tables, Rows and Columns (Relational)
  - **Semi Structured: Key-Value, Document (JSON), Graph, etc**
  - Unstructured: Video, Audio, Image, Text files, Binary files ...
- **Data stores:**
  - Relational databases
  - **NoSQL databases**
  - **Analytical databases**
  - Object/Block/File storage

1.25E+9
1.00E+9
7.50E+8
5.00E+8
2.50E+8
0

Semi Structured Data

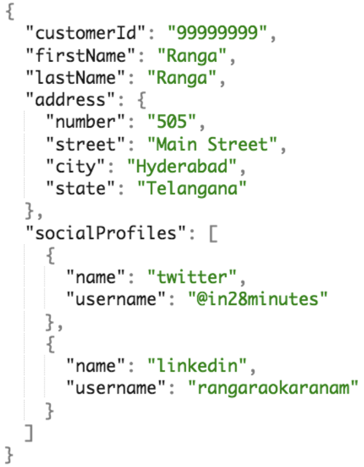
- Data has **some structure BUT not very strict**
- Semi Structured Data is stored in NoSQL databases
  - NoSQL = not only SQL
  - Flexible schema
    - Structure data **the way your application needs it**
    - Let the structure evolve with time
  - Horizontally scale to petabytes of data with millions of TPS
- **Types of Semi Structured Data:**
  - Document
  - Key Value
  - Graph
  - Column Family



```
{
  "customerId": "99999999",
  "firstName": "Ranga",
  "lastName": "Ranga",
  "address": {
    "number": "505",
    "street": "Main Street",
    "city": "Hyderabad",
    "state": "Telangana"
  },
  "socialProfiles": [
    {
      "name": "twitter",
      "username": "@in28minutes"
    },
    {
      "name": "linkedin",
      "username": "rangaraokaranam"
    }
  ]
}
```

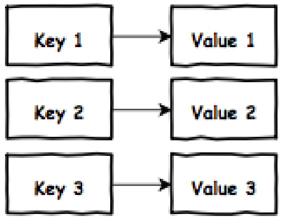
Semi Structured Data - 1 - Document

- Data stored as **collection of documents**
  - Typically **JSON** (Javascript Object Notation)
    - Be careful with formatting (name/value pairs, commas etc)
    - address - Child Object - { }, socialProfiles - Array - [ ]
  - Documents are retrieved by unique id (called the key)
    - Typically, you can define additional indexes
    - Documents don't need to have the same structure
      - No strict schema defined on database
      - Apps should handle variations (application defined schema)
  - Typically, information in one document would be stored in multiple tables, if you were using a relational database
- **Use cases:** Product Catalog, Profile, Shopping Cart etc
- **Managed Services:** Amazon DynamoDB, Amazon DocumentDB (MongoDB compatible), Azure Cosmos DB SQL & MongoDB API, Google Cloud Datastore

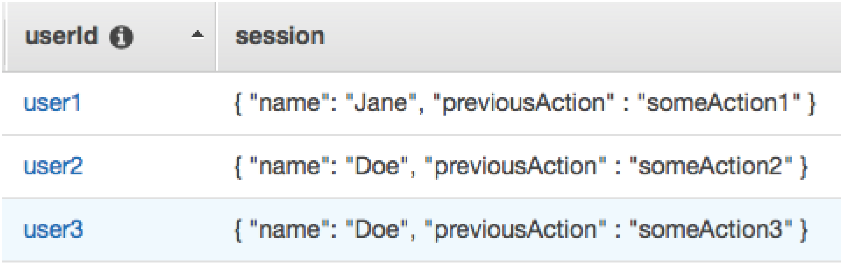


```
{
  "customerId": "99999999",
  "firstName": "Ranga",
  "lastName": "Ranga",
  "address": {
    "number": "505",
    "street": "Main Street",
    "city": "Hyderabad",
    "state": "Telangana"
  },
  "socialProfiles": [
    {
      "name": "twitter",
      "username": "@in28minutes"
    },
    {
      "name": "linkedin",
      "username": "rangaraokaranam"
    }
  ]
}
```

Semi Structured Data - 2 - Key-Value



Key Value Database



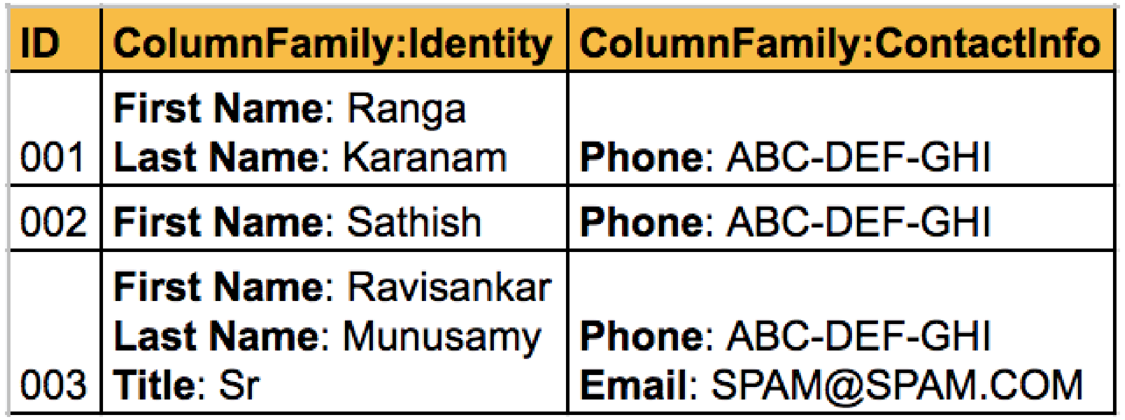
userId	session
user1	{ "name": "Jane", "previousAction" : "someAction1" }
user2	{ "name": "Doe", "previousAction" : "someAction2" }
user3	{ "name": "Doe", "previousAction" : "someAction3" }

- Similar to a **HashMap**
  - **Key** - Unique identifier to retrieve a specific value
  - **Value** - Number or a String or a complex object, like a JSON file
  - Supports simple lookups - query by keys
  - **Use cases:** Session Store, Caching Data
- **Managed Services:** Amazon DynamoDB, Azure Cosmos DB Table API, Google Cloud Datastore

Semi Structured Data - 3 - Graph

- Social media applications have data with complex relationships
- How do you store such data?
  - As a graph in **Graph** Databases
  - Used to store data with **complex relationships**
- Contains nodes and edges (relationships)
- **Use cases**: People and relationships, Organizational charts, Fraud Detection
- **Managed Service**: Amazon Neptune, Azure Cosmos DB Gremlin API

Semi Structured Data - 4 - Column Family



ID	ColumnFamily:Identity	ColumnFamily:ContactInfo
001	**First Name**: Ranga **Last Name**: Karanam	**Phone**: ABC-DEF-GHI
002	**First Name**: Sathish	**Phone**: ABC-DEF-GHI
003	**First Name**: Ravisankar **Last Name**: Munusamy **Title**: Sr	**Phone**: ABC-DEF-GHI **Email**: SPAM@SPAM.COM

- Data organized into **rows and columns** (looks similar to a relational db)
  - **IMPORTANT FEATURE:** Columns are divided into groups called column-family
    - Rows can be sparse (does NOT need to have value for every column)
  - **Use cases:** IOT streams, real time analytics, financial data - transaction histories, stock prices etc
- **Managed Service:** Amazon Keyspaces (for Apache Cassandra), Azure Cosmos DB Cassandra API, Google Cloud Bigtable

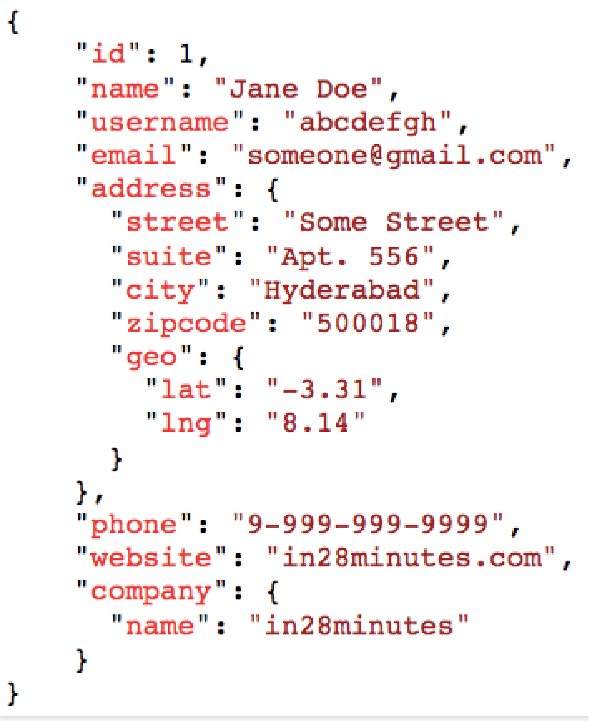
Amazon DynamoDB

- Fast, scalable, **distributed** for any scale
- Flexible **NoSQL** Key-value & document database (schemaless)
- **Single-digit millisecond** responses for **million of TPS**
- Do not worry about scaling, availability or durability
  - Automatically partitions data as it grows
  - Maintains 3 replicas within the same region
- No need to provision a database
  - Create a table and configure read and write capacity (RCU and WCU)
  - Automatically scales to meet your RCU and WCU
- Provides an **expensive serverless mode**
- **Use cases**: User profiles, shopping carts, high volume read write applications

DynamoDB

DynamoDB Tables

- **Hierarchy** : Table > item(s) > attribute (key value pair)
- **Mandatory** primary key
- Other than the primary key, tables are **schemaless**
  - No need to define the other attributes or types
  - Each item in a table can have distinct attributes
- Max 400 KB per item in table
- DynamoDB Tables are region specific.
  - If your users are in multiple regions, mark the table as **Global Table**
  - Replicas are created in selected regions



```
{
    "id": 1,
    "name": "Jane Doe",
    "username": "abcdefgh",
    "email": "someone@gmail.com",
    "address": {
      "street": "Some Street",
      "suite": "Apt. 556",
      "city": "Hyderabad",
      "zipcode": "500018",
      "geo": {
        "lat": "-3.31",
        "lng": "8.14"
      }
    },
    "phone": "9-999-999-9999",
    "website": "in28minutes.com",
    "company": {
      "name": "in28minutes"
    }
}
```

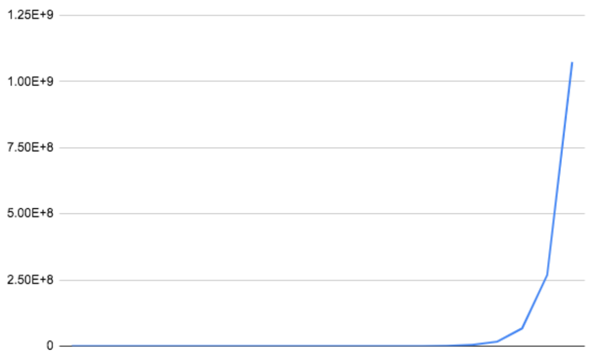
DynamoDB Consistency Levels

- (DEFAULT) **Eventually Consistent Reads** : Might NOT get latest data
  - If tried after few seconds, you will get the latest data
- **Strongly Consistent Reads**: Get the most up-to-date data
  - Reflects updates from all the previous successful write operations
  - Set `ConsistentRead` to true
  - Disadvantages:
    - May have higher latency
    - Uses more throughput capacity units
- **Supports transactions** (TransactWriteItems, TransactGetItems)
  - All-or-nothing changes to multiple items both within and across tables
    - Include PutItem, UpdateItem and DeleteItem operations
  - More expensive

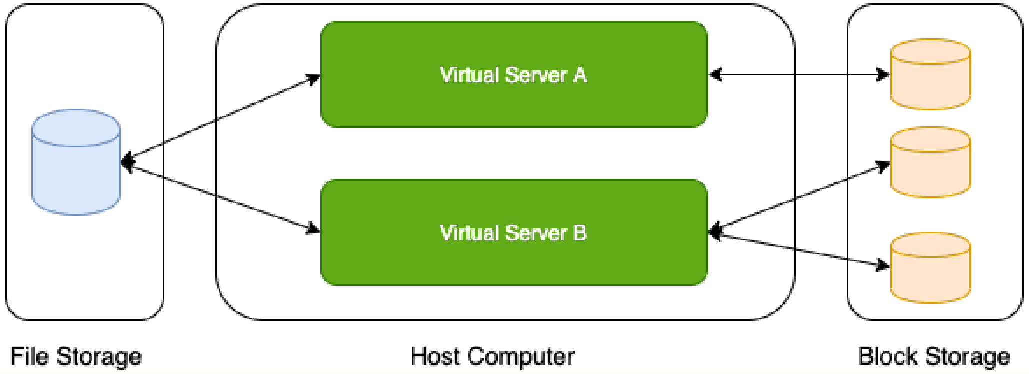
DynamoDB

Data Formats & Data Stores

- **Data formats:**
  - Structured: Tables, Rows and Columns (Relational)
  - Semi Structured: Key-Value, Document (JSON), Graph, etc
  - **Unstructured: Video, Audio, Image, Text files, Binary files ...**
- **Data stores:**
  - Relational databases
  - NoSQL databases
  - Analytical databases
  - **Object/Block/File storage**



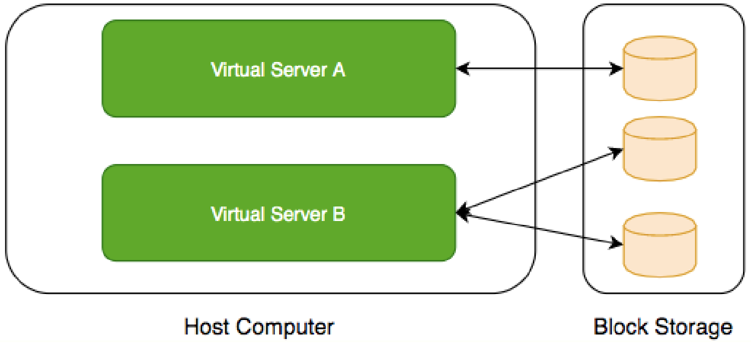
Unstructured Data



- Data which does not have any structure (Audio files, Video files, Binary files)
  - What is the type of storage of your hard disk?
    - **Block Storage** (Azure Disks, Amazon EBS, Google Cloud Persistent Disk)
  - You've created a file share to share a set of files with your colleagues in a enterprise. What type of storage are you using?
    - **File Storage** (Azure Files, Amazon EFS, Google Cloud Filestore)
  - You want to be able to upload/download objects using a REST API without mounting them onto your VM. What type of storage are you using?
    - **Object Storage** (Azure Blob Storage, Amazon S3, Google Cloud Storage)

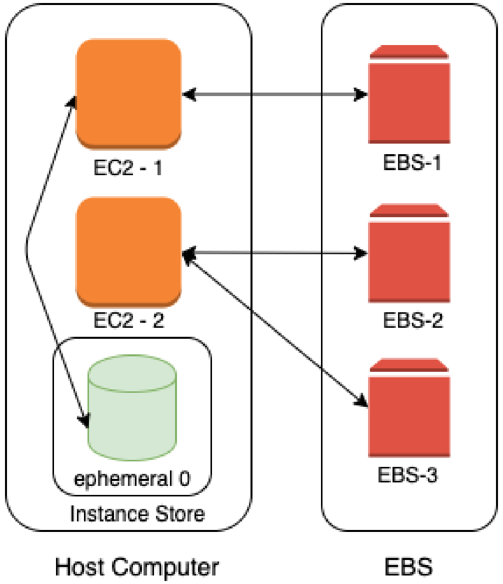
Block Storage

- Use case: Hard-disks attached to your computers
- Typically, ONE Block Storage device can be connected to ONE virtual server
- HOWEVER, you can connect multiple different block storage devices to one virtual server



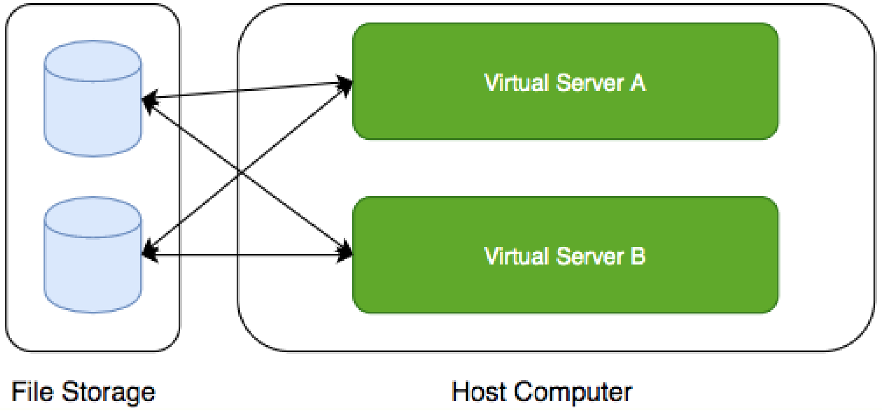
EC2 - Block Storage

- Two popular types of block storage can be attached to EC2 instances:
  - **Elastic Block Store (EBS)**
  - **Instance Store**
- **Instance Stores** are physically attached to the EC2 instance
  - Temporary data
  - Lifecycle tied to EC2 instance
- **Elastic Block Store (EBS)** is network storage
  - More durable
  - Lifecycle NOT tied to EC2 instance



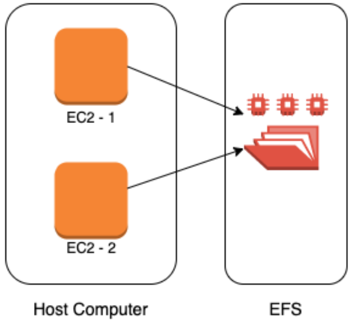
File Storage

- Media workflows need huge shared storage for supporting processes like video editing
- Enterprise users need a quick way to share files in a secure and organized way
- These file shares are shared by several virtual servers



Amazon EFS

- **Petabyte scale, Auto scaling, Pay for use** shared file storage
- Compatible with Amazon EC2 Linux-based instances
- **(Usecases)** Home directories, file share, content management
- (Alternative) Amazon FSx for Lustre
  - File system **optimized for performance**
  - High performance computing (HPC) and media processing use cases
  - Automatic encryption at-rest and in-transit
- (Alternative) Amazon FSx Windows File Servers
  - Fully managed Windows file servers
  - Accessible from Windows, Linux and MacOS instances
  - Integrates with Microsoft Active Directory (AD) to support Windows-based environments and enterprises.
  - Automatic encryption at-rest and in-transit



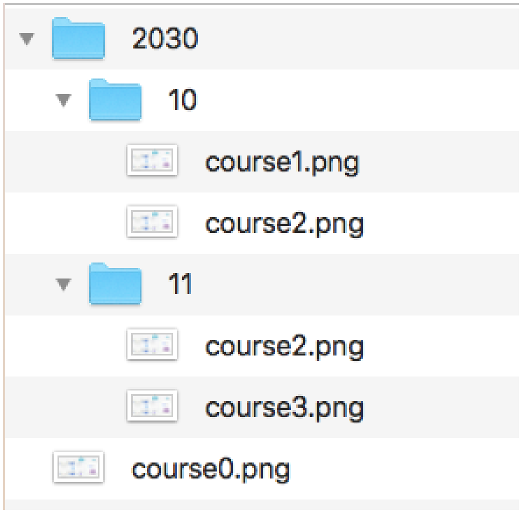
Amazon S3 (Simple Storage Service)

- **Most popular, very flexible & inexpensive** storage service
- Store objects using a **key-value** approach (objects in buckets)
- Provides REST API to access and modify objects
- Provides **unlimited storage**:
  - (S3 storage class) **99.99% availability** & **(11 9's - 99.999999999) durability**
  - Objects are **replicated in a single region (across multiple AZs)**
- **Store all file types** - text, binary, backup & archives:
  - Media files and archives
  - Application packages and logs
  - Backups of your databases or storage devices
  - Staging data during on-premise to cloud database migration

Amazon S3

Amazon S3 Key Value Example

Key	Value
2030/course0.png	image-binary-content
2030/10/course1.png	image-binary-content
2030/10/course2.png	image-binary-content
2030/11/course2.png	image-binary-content
2030/11/course3.png	image-binary-content



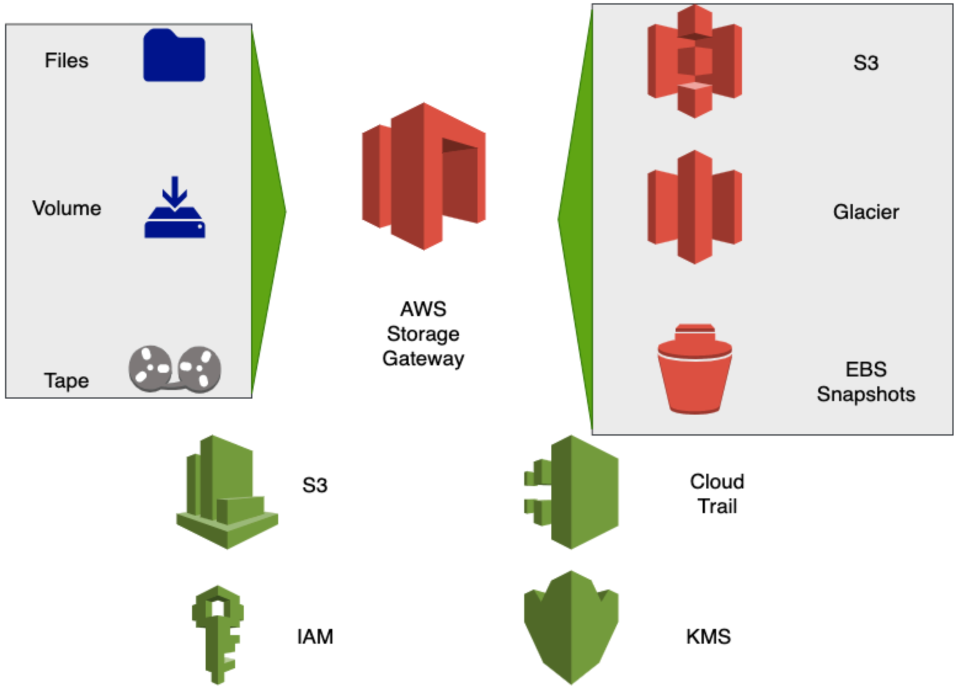
Amazon S3 Glacier

- **Extremely low cost storage** for archives and long-term backups:
  - Old media content
  - Archives to meet regulatory reqmts. (old patient records etc)
  - As a replacement for magnetic tapes
- High durability (11 9s - 99.999999999%)
- High scalability (unlimited storage)
- High security (**encrypted** at rest and in transfer)

Amazon Glacier

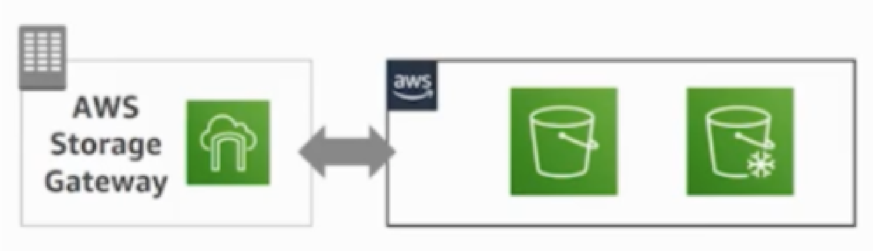
AWS Storage Gateway

- **Hybrid storage** (cloud + on premise)
- Unlimited cloud storage for on-premise software applications and users with good performance
- (Remember) Storage Gateway and S3 Glacier **encrypt data** by default
- **Three Options**
  - AWS Storage File Gateway
  - AWS Storage Tape Gateway
  - AWS Storage Volume Gateway



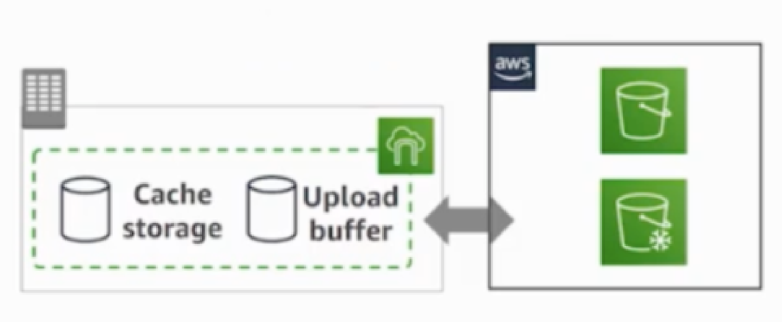
AWS Storage File Gateway

- **Problem Statement:** Large on-premise file share with terabytes of data
  - Users put files into file share and applications use the files
  - Managing it is becoming expensive
  - Move the file share to cloud without performance impact
- AWS Storage File Gateway provides cloud storage for your file shares
  - Files **stored** in **Amazon S3 & Glacier**



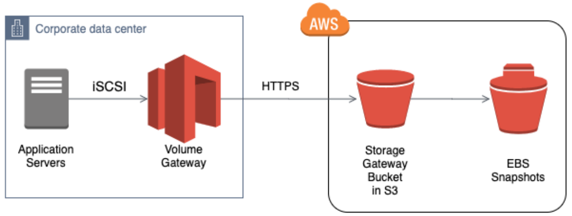
AWS Storage Tape Gateway

- Tape backups used in enterprises (archives)
    - Stored off-site - expensive, physical wear and tear
- **AWS Storage Tape Gateway** - Avoid physical tape backups
- **No change needed** for tape backup infrastructure
- Backup data to virtual tapes (actually, Amazon S3 & Glacier)



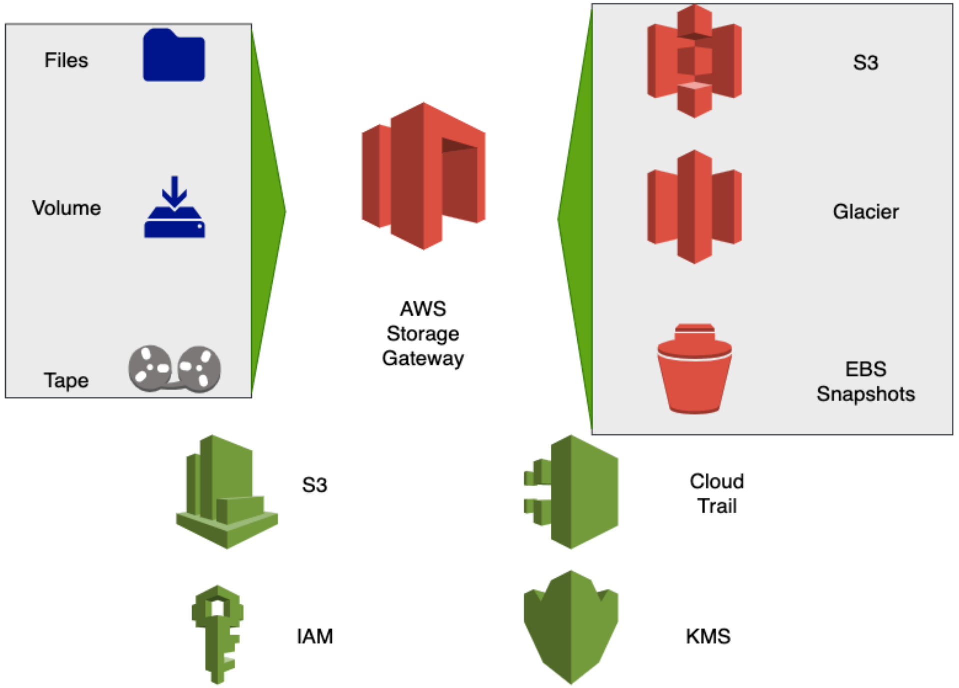
AWS Storage Volume Gateway

- **Volume Gateway** : Hybrid block storage
- Cloud-backed iSCSI block storage volumes
- Your connected applications think they are using local block storage
- (Option 1) **Cached** (Gateway Cached Volumes):
  - Primary Data Store - **AWS - Amazon S3**
  - **On-premise cache** stores frequently accessed data
- (Option 2) **Stored** (Gateway Stored Volumes):
  - Primary Data Store - **On-Premises**
  - Asynchronous copy to AWS
  - Stored as EBS snapshots



AWS Storage Gateway - Summary

- Key to look for : **Hybrid storage** (cloud + on premise)
- File share moved to cloud => **AWS Storage File Gateway**
- Tape Backups on cloud => **AWS Storage Tape Gateway**
- Volume Backups on cloud (Block Storage) => **AWS Storage Volume Gateway**
  - High performance => **Stored**
  - Otherwise => **Cached**



Quick Review - Storage in AWS

Type	Description
Object	**Amazon S3** (Very Flexible) Store large objects using a key-value approach
Block	Storage connected to one EC2 instance. Your Hard Disks. **Elastic Block Storage**(EBS - Permanent) **Instance Store** (Ephemeral)
File	File Share. Share storage between EC2 instances. **EFS** (Linux) **FSx Windows** **FSx for Lustre** (High Performance)
Archival	**Amazon S3 Glacier** Extremely low cost storage for archives and long-term backups.
Hybrid	**AWS Storage Gateway** Cloud + On Premise

Data Stores - Scenarios

Scenario	Solution
A start up with quickly evolving schema for storing documents	Amazon DynamoDB
Transactional local database processing thousands of transactions per second	Amazon RDS
Store complex relationships between transactions to identify fraud	Amazon Neptune
Database for analytics processing of petabytes of structured data	Amazon Redshift
File share between multiple VMs	Amazon EFS
Storing profile images uploaded by your users	Amazon S3

Identity and Access Management

Typical identity management in the cloud

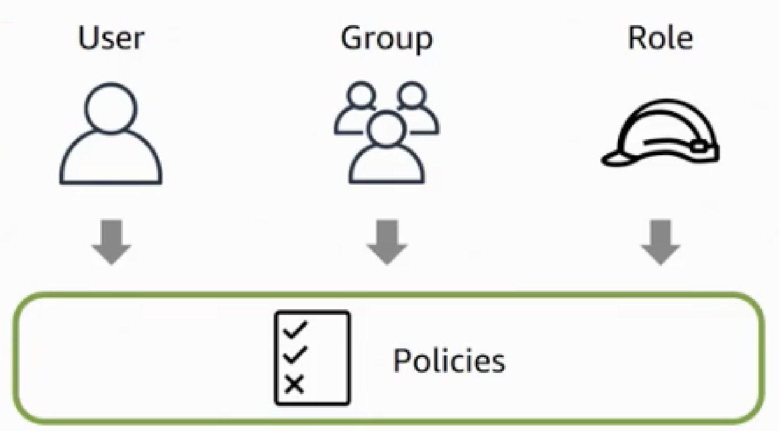
- You have **resources** in the cloud (examples - a virtual server, a database etc)
- You have **identities (human and non-human)** that need to access those resources and perform actions
  - For example: launch (stop, start or terminate) a virtual server
- How do you **identify users** in the cloud?
- How do you configure resources they can access?
- How can you configure what actions to allow?
- In AWS, *Identity and Access Management (IAM)* provides this service

AWS Identity and Access Management

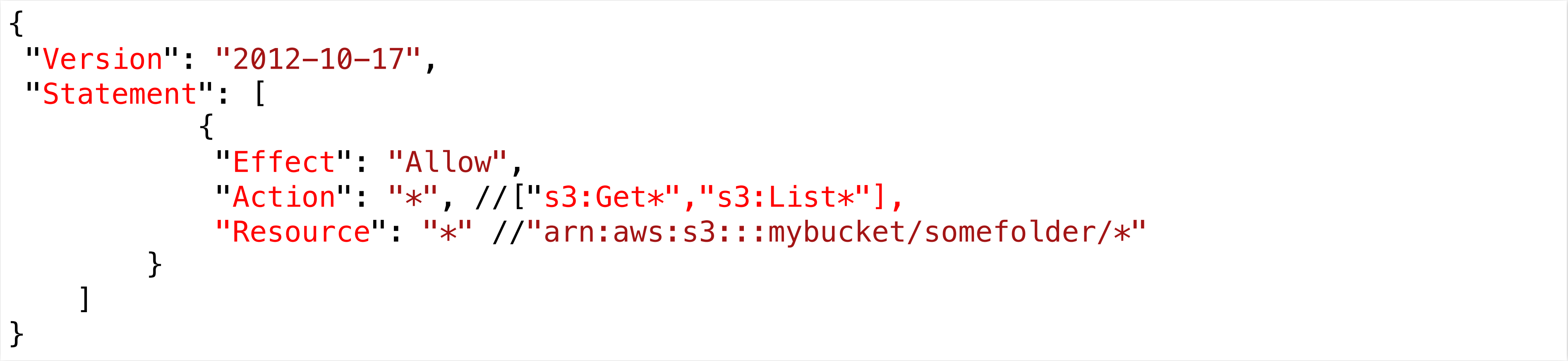
- **Authentication** (the right user?)
- **Authorization** (the right access?)
- Provides very **granular** control
  - Limit a single user:
    - to perform single action
    - on a specific AWS resource
    - from a specific IP address
    - during a specific time window

Important IAM Concepts

- **IAM users**: Users created in an AWS account
  - Have credentials attached (name/password or access keys)
- **IAM groups**: Collection of IAM users
- **Roles**: Temporary identities
  - Does NOT have credentials attached
  - (Advantage) Expire after a set period of time
- **Policies**: Define permissions
  - For IAM users, groups and roles



AWS IAM Policies - Authorization



```
{
 "Version": "2012-10-17",
 "Statement": [
           {
            "Effect": "Allow",
            "Action": "*", //["s3:Get*","s3:List*"],
            "Resource": "*" //"arn:aws:s3:::mybucket/somefolder/*"
        }
    ]
}
```

- Policy is a JSON document with one or more permissions
  - **Effect** - Allow or Deny
  - **Resource** - Which resource are you providing access to?
  - **Action** - What actions are allowed on the resource?
  - **Condition** - Are there any restrictions on IP address ranges or time intervals?
  - Give Read Only Access to S3 buckets - `"Action": ["s3:Get*","s3:List*"]`

IAM Best Practices - Recommended by AWS

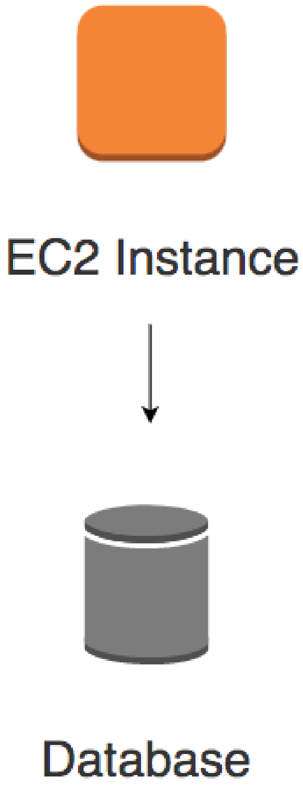
- **Users** – Create individual users
- **Groups** – Manage permissions with groups
- **Permissions** – Grant least privilege
- **Auditing** – Turn on AWS CloudTrail
- **Password** – Configure a strong password policy
- **MFA** – Enable MFA for privileged users
  - (Hardware device - Gemalto, Virtual device - An app on a smart phone)
- **Roles** – Use IAM roles for Amazon EC2 instances
- **Sharing** – Use IAM roles to share access
- **Rotate** – Rotate security credentials regularly
- **Root** – Reduce or remove use of root

AWS IAM

Data Encryption

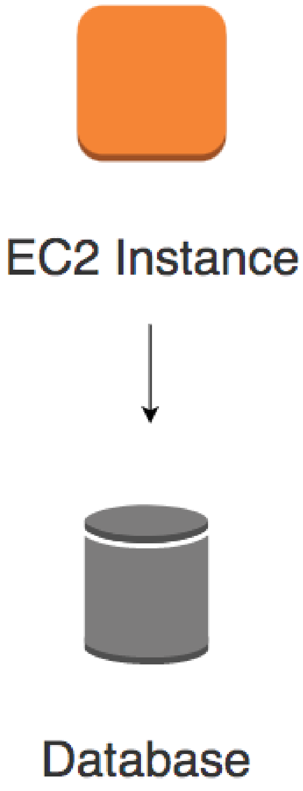
Data States

- **Data at rest:** Stored on a device or a backup
  - Examples : data on a hard disk, in a database, backups and archives
- **Data in motion:** Being transferred across a network
  - Also called **Data in transit**
  - **Examples** :
    - Data copied from on-premise to cloud storage
    - An application in a VPC talking to a database
  - **Two Types:**
    - In and out of AWS
    - Within AWS
- **Data in use:** Active data in a non-persistent state
  - Example: Data in your RAM

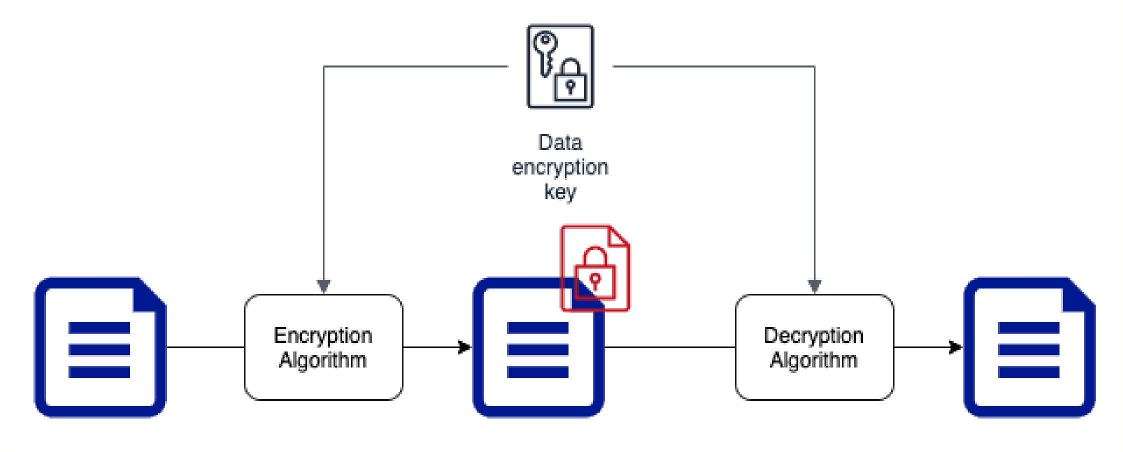


Encryption

- If you store data as is, what would happen if an **unauthorized entity gets access** to it?
  - Imagine losing an unencrypted hard disk
- **First law of security** : Defense in Depth
- Typically, enterprises encrypt all data
  - Data on your hard disks
  - Data in your databases
  - Data on your file servers
- Is it sufficient if you encrypt data at rest?
  - **No. Encrypt data in transit** - between application to database as well.



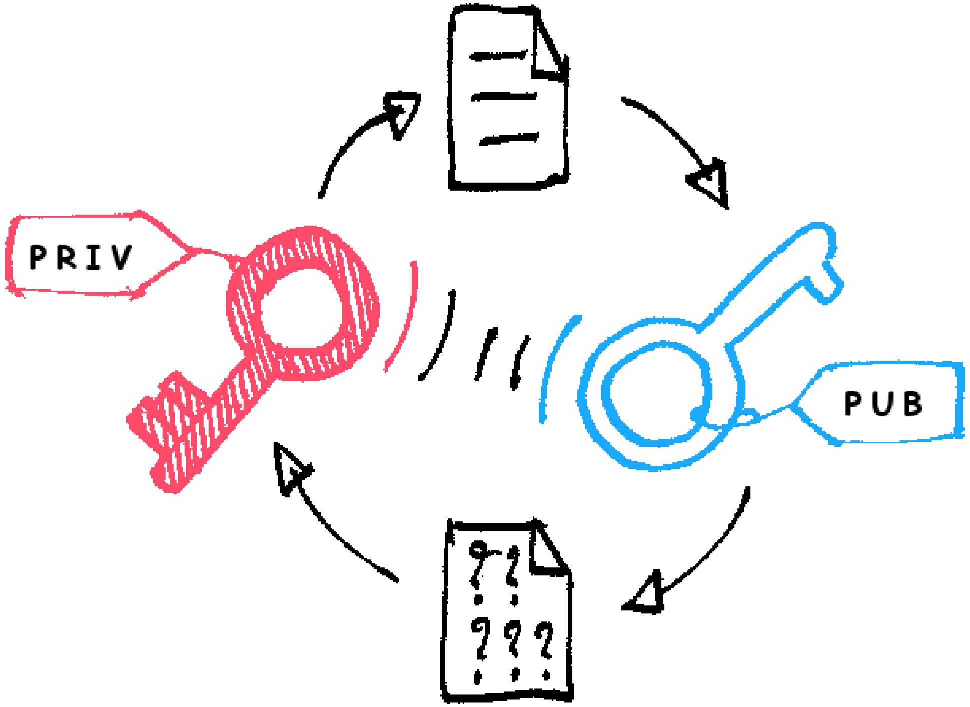
Symmetric Key Encryption



- Symmetric encryption algorithms use the **same key for encryption and decryption**
- Key Factor 1: Choose the **right encryption algorithm**
- Key Factor 2: How do we **secure the encryption key**?
- Key Factor 3: How do we **share the encryption key**?

Asymmetric Key Encryption

- **Two Keys** : Public Key and Private Key
- Also called **Public Key Cyptography**
- Encrypt data with Public Key and decrypt with Private Key
- Share Public Key with everybody and keep the Private Key with you(YEAH, ITS PRIVATE!)
- No crazy questions:
  - Will somebody not figure out private key using the public key?
- How do you create Asymmetric Keys?



https://commons.wikimedia.org/wiki/File:Asymmetric_encryption_(colored).p

KMS and Cloud HSM

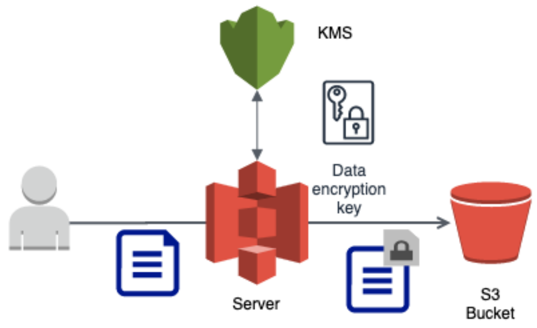
- Generate, store, use and replace your keys(symmetric & asymmetric)
- **KMS**: Multi-tenant Key Management Service
  - **KMS** integrates with all storage and database services in AWS
  - Define key usage permissions (including **cross account** access)
  - **Automatically rotate master keys** once a year
  - **Schedule key deletion** to verify if the key is used
    - Mandatory minimum wait period of 7 days (max-30 days)
- **CloudHSM**: **Dedicated single-tenant** HSM for regulatory compliance
  - AWS **CANNOT access your encryption master keys** in CloudHSM
    - **(Recommendation)** Be ultra safe with your keys. Use two or more HSMs in separate AZs.

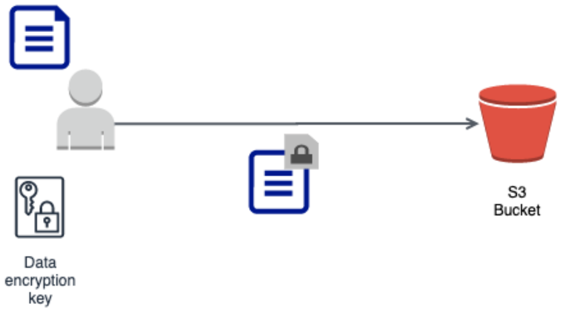
AWS KMS

Cloud HSM

Server Side vs Client Side Encryption

- **Server Side Encryption:** S3<->KMS - encrypt data
  - **SSE-S3**: AWS S3 manages its own keys
  - **SSE-KMS**: Customer manages keys in KMS
  - **SSE-C**: Customer sends key with request
    - S3 performs encryption and decryption without storing the key
  - **Use HTTPS endpoints** (secure data in transit)
    - All AWS services (including S3) provides HTTPS endpoints
- **Client Side Encryption:** Client **owns encryption**
  - Client sends encrypted data to AWS service
    - AWS will not be aware of master key or data key
    - AWS service stores data as is





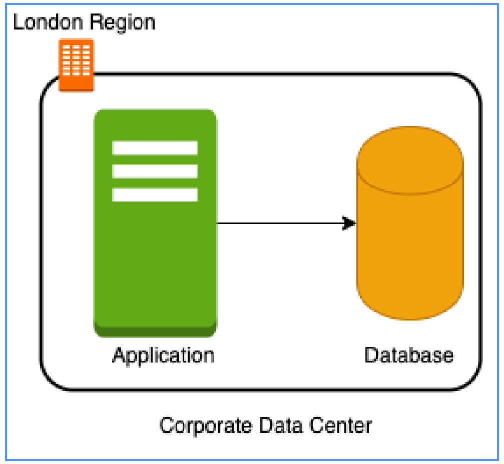
Exploring Security in AWS - Scenarios

Scenario	Solution
Which of these are temporary identities? IAM Roles or IAM Users	IAM Roles
How do you define permissions for IAM Identities in AWS?	IAM Policies
You want to assign permissions for Amazon EC2 instances. Should you use IAM Users or IAM Roles?	IAM Roles
Which of these services can be used to create Symmetric and Asymmetric keys for encryption?	Amazon KMS
You want Dedicated single-tenant HSM for regulatory compliance	Cloud HSM

Virtual Private Cloud (VPC)

Need for Amazon VPC

- In a corporate network or an on-premises data center:
  - Can anyone on the internet **see the data exchange** between the application and the database?
    - **No**
  - Can anyone from internet **directly connect to your database**?
    - Typically **NO**.
    - You need to connect to your corporate network and then access your applications or databases.
- Corporate network provides a **secure internal network** protecting your resources, data and communication from external users
- How do you do create **your own private network** in the cloud?
  - Enter **Virtual Private Cloud (VPC)**

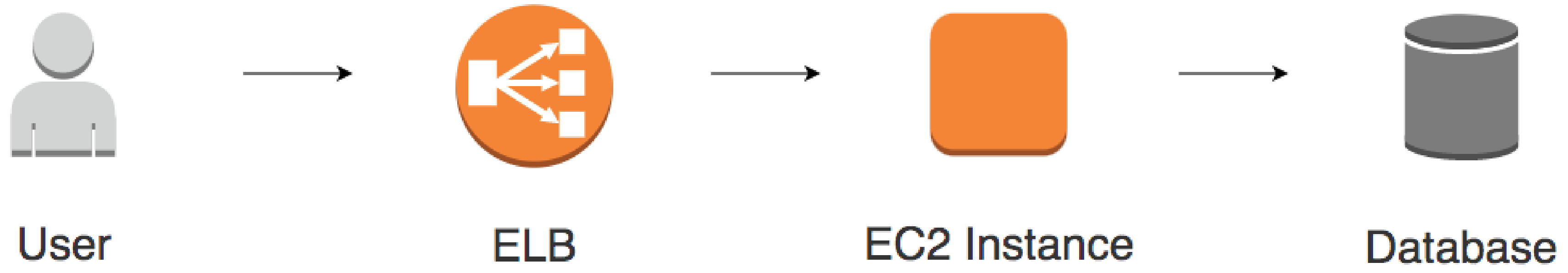


Amazon VPC (Virtual Private Cloud)

- Your **own isolated network** in AWS cloud
  - Network traffic within a VPC is isolated (not visible) from all other Amazon VPCs
- You **control all the traffic** coming in and going outside a VPC
- **(Best Practice)** Create all your AWS resources (compute, storage, databases etc) **within a VPC**
  - Secure resources from unauthorized access AND
  - Enable secure communication between your cloud resources

VPC

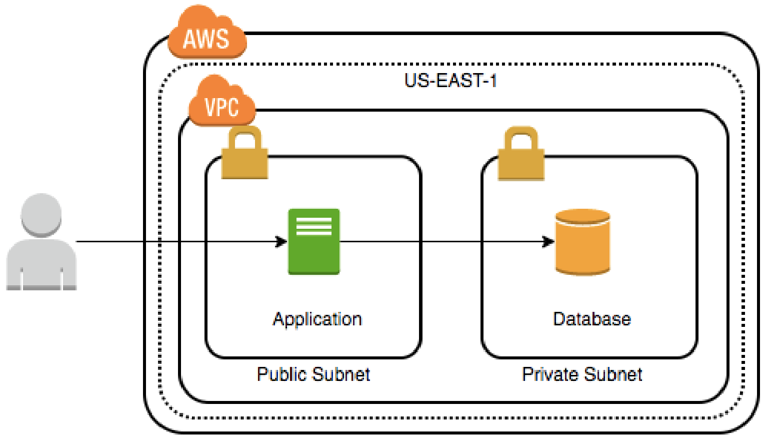
Need for VPC Subnets



- Different resources are created on cloud - databases, compute (EC2) etc
- Each type of resource has **its own access needs**
- Public Elastic Load Balancers are accessible from internet (**public** resources)
- Databases or EC2 instances should NOT be accessible from internet
  - ONLY applications within your network (VPC) should be able to access them(**private** resources)
- How do you **separate public resources from private resources** inside a VPC?

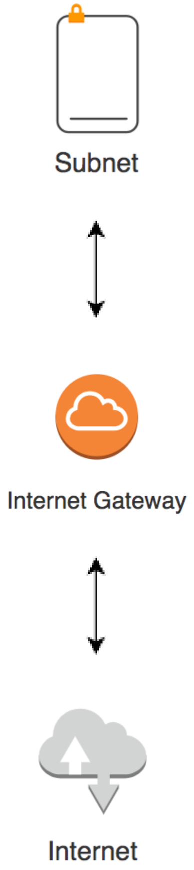
VPC Subnets

- (Solution) **Create different subnets** for public and private resources
  - Resources in a public subnet **CAN** be accessed from internet
  - Resources in a private subnet **CANNOT** be accessed from internet
  - BUT resources in public subnet can talk to resources in private subnet
- Each VPC is created in a Region
- Each Subnet is created in an Availability Zone
- **Example** : VPC - us-east-1 => Subnets - AZs us-east-1a or us-east-1b or ..



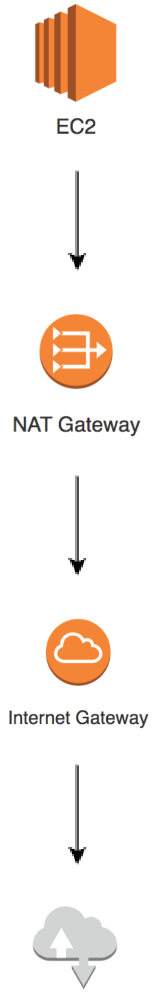
Public Subnet vs Private Subnet

- **Public Subnet:** Communication allowed - Subnet <-> Internet
- An **Internet Gateway** enables internet communication for subnets
  - **Public Subnet:** Subnet having a route to an internet gateway
  - **Private Subnet:** Subnet **DOES NOT** have route to an internet gateway
- **Internet Gateway:** Between subnet resources & internet
  - **One to one mapping** with a VPC
  - Supports IPv4 and IPv6
  - Translate private IP address to public IP address and vice-versa

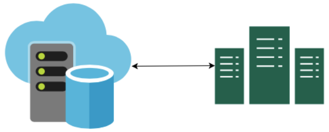


Private Subnet - NAT Devices - Download Patches

- What if you want to allow ONLY **outbound traffic to internet** from your private subnet?
  - While DENYing all inbound traffic
- **NAT Devices:** Allow outbound internet access for private subnets
  - **Allow instances in a private subnet to download software patches** while denying inbound traffic from internet
  - **Three Options:**
    - **NAT Instance:** Create your own EC2 instance with specific NAT AMI
    - **NAT Gateway:** Managed Service (PREFERRED - No maintenance, more availability & high bandwidth)
      - NAT Gateway supports **IPv4 ONLY**
    - **Egress-Only Internet Gateways:** For IPv6 subnets



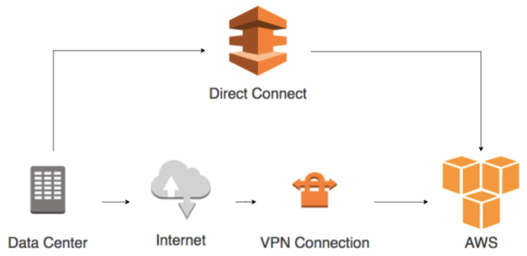
Cloud Computing: Public vs Private vs Hybrid clouds



- Cloud Computing
  - **Public Cloud**
    - You host everything in the cloud (You DO NOT need a data center anymore)
      - No Capital Expenditure required
    - Hardware resources are owned by cloud platform
      - Hardware failures and security of the data center are managed by cloud platform
    - Summary: Hardware owned by cloud platform and shared between multiple tenants
      - Tenants: Customers who rent infrastructure (You, Me and other enterprises)
  - **Private Cloud**
    - You host everything in your own data center
      - Needs Capital Expenditure
      - Incur staffing and maintenance expenses for infrastructure
    - Delivers higher level of security and privacy
  - **Hybrid Cloud:**
    - Combination of both (Public & Private)
      - Use Public Cloud for some workloads and Private cloud for others
      - Example: Connecting an on-premise app to a cloud database
    - Provides you with flexibility: Go on-premises or cloud based on specific requirement

AWS and On-Premises - Overview

- **AWS Managed VPN:** Tunnels from VPC to on premises
  - Traffic over internet - encrypted using IPsec protocol
  - VPN gateway to connect one VPC to customer network
  - Customer gateway installed in customer network
    - You need a Internet-routable IP address of customer gateway
- **AWS Direct Connect (DX):** Private dedicated network connection to on premises
  - (Advantage) Reduce your (ISP) bandwidth costs
  - (Advantage) Consistent Network performance (private network)
  - (Caution) Establishing DC connection takes a month
  - (Caution) Establish a redundant DC for maximum reliability
  - (Caution) Data is NOT encrypted (Private Connection ONLY)



VPC - Review

- **VPC**: Virtual Network to protect resources and communication from outside world.
- **Subnet**: Seperate private resources from public resources
- **Internet Gateway**: Allows Public Subnets to connect/accept traffic to/from internet
- **NAT Gateway/Egress-Only Internet Gateways**: Allow internet traffic from private subnets
- **AWS Direct Connect**: Private pipe from AWS to on-premises
- **AWS VPN**: Encrypted (IPsec) tunnel over internet to on-premises

DevOps

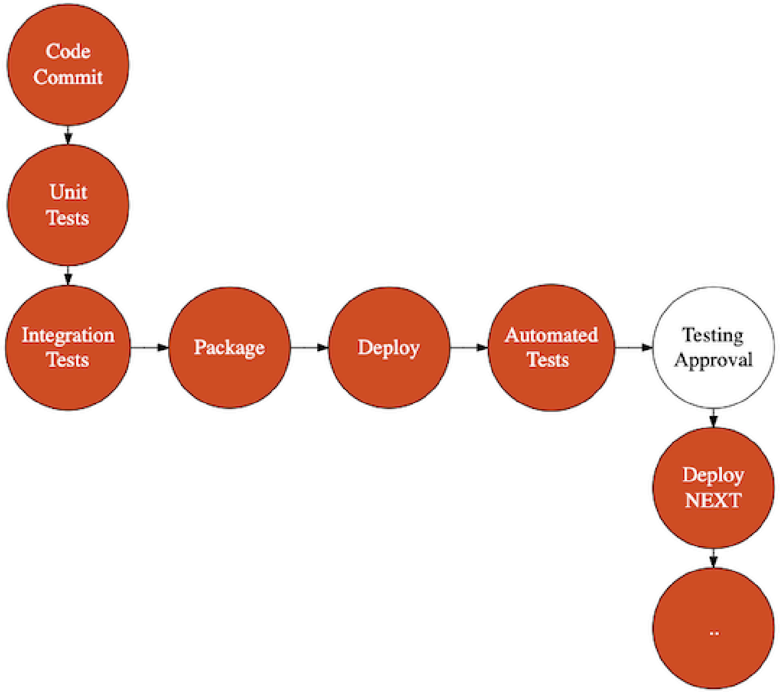
DevOps

Business
Development
Operations

- Getting Better at "**Three Elements of Great Software Teams**"
  - **Communication** - Get teams together
  - **Feedback** - Earlier you find a problem, easier it is to fix
  - **Automation** - Automate testing, infrastructure provisioning, deployment, and monitoring

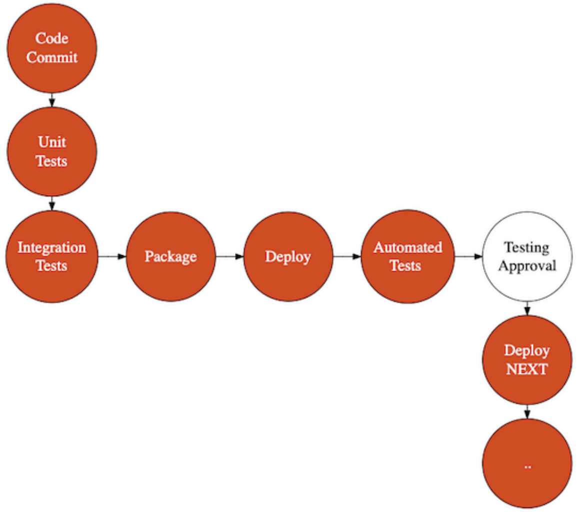
DevOps - CI, CD

- How do you get quick feedback?
  - Do things continuously in small iterations!
- **Continuous Integration**
  - Continuously run your tests and packaging
- **Continuous Delivery**
  - Continuously deploy to test environments
- **Continuous Deployment**
  - Continuously deploy to production

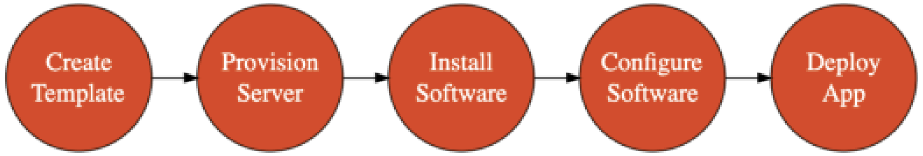


DevOps - CI, CD Tools in AWS

- **AWS CodeCommit:** Fully-featured, private Git repository
  - Similar to Github
- **AWS CodePipeline:** Orchestrate CI/CD pipelines
- **AWS CodeBuild** - Build and Test Code (packages and containers)
- **AWS CodeDeploy** - Automate Deployment(ECS, Lambda etc)
- **Amazon Elastic Container Registry (ECR):** Store your Docker images

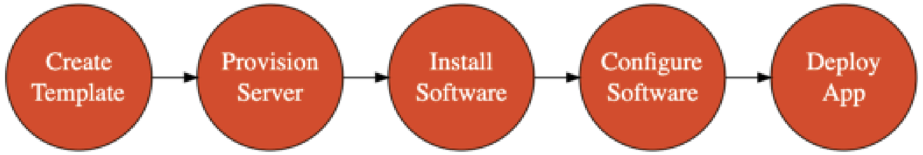


DevOps - Infrastructure as Code



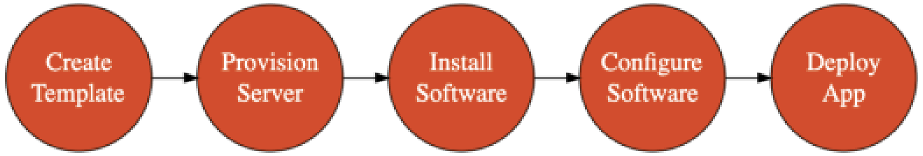
- **Lets consider an example:**
  - I would want to create a new VPC and a subnet
  - I want to provision a ELB, ASG with 5 EC2 instances & RDS database
  - I would want to install Python one EC2 instances
  - AND I would want to create 4 environments
    - Dev, QA, Stage and Production!
- Doing this manually:
  - Will take time and there is every chance you will make a mistake
- How about automating this?
  - **Infrastructure as Code**

DevOps - Infrastructure as Code - 2



- **Infrastructure as Code**: Automate Provisioning & Configuration of Resources (in cloud, most of the times)
- Treat **infrastructure the same way as application code**
  - Track infrastructure **changes over time** (versioning)
  - Bring **repeatability** into your infrastructure
  - Advantages:
    - Automate deployment of resources in a controlled, predictable way
    - Avoid mistakes with manual configuration

DevOps - Infrastructure as Code - Two Parts

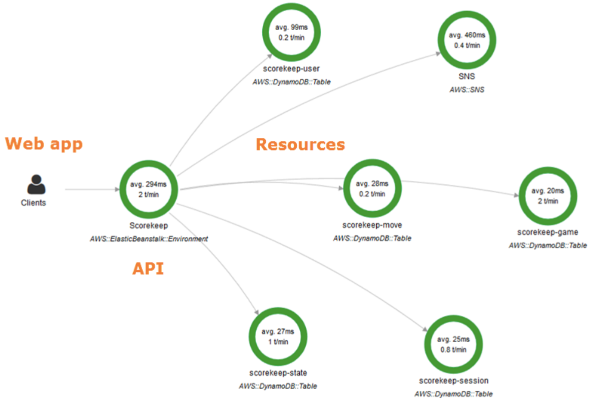


- **1: Infrastructure Provisioning**: Provision AWS resources
  - **Cloud neutral**: Terraform (templates), Pulumi (code)
  - **AWS**:
    - **AWS CloudFormation** - Provision Resources using JSON/YAML templates
    - **AWS Cloud Development Kit (AWS CDK)** - Provision resources using popular programming languages
    - **AWS Serverless Application Model (SAM)** - Provision Serverless Resources
- **2: Configuration Management**: Install right software and tools
  - **Open Source Tools** - Chef, Puppet, Ansible and SaltStack
  - **AWS Service**: OpsWorks (Chef, Puppet in AWS)

Monitoring & Debugging with CloudWatch & X-Ray

- **Amazon CloudWatch** - Monitor Resources & Apps
  - **CloudWatch Metrics**: Metrics for your apps & resources
  - **CloudWatch Logs**: Monitor and troubleshoot using system, application and custom log files
  - **CloudWatch Alarms**: Create alarms based on CloudWatch Metrics and take immediate action (Send an email, Autoscale...)
  - **CloudWatch Events (NOW EventBridge)**: **Take immediate action** based on **events on AWS resources**
    - Call a AWS Lambda function when an EC2 instance starts
    - Notify an Amazon SNS topic when an Auto Scaling event happens
- **X-Ray** - Analyze and Debug Your Applications
  - Trace request across microservices/AWS services

Cloudwatch



Exploring DevOps in AWS - Scenarios

Scenario	Solution
What is the difference between Continuous Delivery and Continuous Deployment?	**Continuous Delivery**: Continuously deploy to test environments **Continuous Deployment**: Continuously deploy to production
Popular Tools for Automating Infrastructure Provisioning in AWS	AWS CloudFormation, AWS CDK, AWS SAM, Terraform, Pulumi
Recommended Managed Service for Configuration Management in AWS	OpsWorks (Chef, Puppet in AWS)
Which managed service can you use to trace your request across microservices/AWS services?	Amazon X-Ray
How can you troubleshoot problems using system, application and custom log files in AWS?	CloudWatch Logs

Cost Management

Total Cost of Ownership(TCO)

- **Total Cost of Ownership(TCO)** includes:
  - Infrastructure Costs
    - Procuring Servers, Databases, Storage, Routers ..
    - Infrastructure maintenance costs
  - Licensing Costs (Software + Hardware)
  - Networking Costs (Connection cost + Data Ingress + Data Egress)
  - Personnel Costs (Dev + Test + Ops + Business + ..)
  - Other Costs:
    - Penalties for missed SLAs or Compliance needs
    - Third Party APIs
    - Electricity costs
- **When designing solutions (or migrating to cloud)**, ensure that you take Total Cost of Ownership(TCO) into account!
  - Compare Apples to Apples!

Expenditure Models: CapEx vs OpEx

- **Capital Expenditure (CapEx):** Money spent to buy infrastructure
  - Additional cost to maintain infrastructure with time
  - You might need a team to manage the infrastructure
  - **Example**: Deploying your own data center with physical servers
  - **Example**: Purchasing Reservations
  - **Example**: Leasing Software
- **Operational Expenditure (OpEx):** Money spent to use a service or a product
  - **Zero upfront costs**
  - You Pay for services as you use them (Pay-as-you-go model)
  - **Example**: Provisioning VMs as you need them
  - **Example**: Using Serverless FaaS and paying for invocations

Pricing Models - Consumption-based vs Fixed

- **Consumption-based** - You are billed for only what you use
  - **Example**: Serverless FaaS - You pay for no of invocations!
- **Fixed-price** - You are billed for instances irrespective of whether they are used or not
  - **Example**: You provision a VM instance
    - You pay for its lifetime irrespective of whether you use it or NOT
  - **Example**: You provision a Kubernetes cluster
    - You are billed irrespective of whether you use it or not

Billing and Cost Management Services/Tools

Service	Description
AWS Billing and Cost Management	Pay your AWS bill, monitor your usage **Cost Explorer** - View your AWS cost data as a graph (Filter by Region, AZ, tags etc. See future cost projection.) **AWS Budgets** - Create a budget (Create alerts (SNS)) Recommendation: Enable Cost allocation tags. Helps you categorize your resource costs in Cost Management.
AWS Compute Optimizer	Recommends compute optimizations to reduce costs (Ex: Right-sizing - EC2 instance type, Auto Scaling group configuration)
AWS Pricing Calculator (NEW)	Estimate cost of your architecture solution
AWS Simple Monthly Calculator (OLD)	Estimate charges for AWS services
TCO - Total Cost of Ownership Calculator (OLD)	Compare Cost of running applications in AWS vs On Premise

Managing Costs - Best Practices

- **Group resources** based on cost ownership
  - Tags etc.
- **Regular cost reviews** (at least weekly)
  - CapEx (Ahead of time planning) -> OpEx (regular reviews)
  - Involve all teams - executive, management, business, technology & finance
- **Estimate costs** before you deploy (Pricing Calculator)
- Use **Cost Management features**
  - Budgets and Budgets alerts etc.
- **Others**:
  - Stop Resources when you don't need them
  - Use Managed Services (PaaS >>> IaaS)
  - Use Spot instances for fault tolerant non-critical workloads
  - Reserve compute for 1 or 3 years

Digital Transformation

What has changed in last decade or so?

- How consumers make purchase decisions? (**Social**)
- How we do things? (**Mobile**)
- How much data we have? (**Big Data**)
  - How much intelligence we can get? (**AI/ML**)
- How much access startups have to technology at scale? (**Cloud**)

Enterprises have to adapt (or get disrupted)

- **Enterprises can ADAPT** by:
  - Providing awesome (omni-channel - social, mobile) customer experiences
  - Getting intelligence from data (Big Data, AI/ML)
    - Example: Personalize consumer offerings
  - Enabling themselves to make changes faster
    - Cultural change from "traditional Datacenter, SDLC, manual IT Ops" to "Cloud, Containers, DevOps/SRE, Automation"
- **Digital Transformation**: Using modern technologies to create (or modify) business processes & customer experiences by innovating with technology and team culture
  - Focus on WHY (NOT HOW)
    - Increase pace of change
    - Revenue Growth
    - Cost Savings
    - Higher customer engagement/retention

Cloud - Enabler for Digital Transformation

- Cloud can **ENABLE** Digital Transformations
  - Lower cost
  - Reduced responsibilities
  - Higher capabilities
  - Increased speed to market
- **BUT needs a change** in skills, mindset and culture
  - Modern Architectures (Microservices, Serverless, Containers, Kubernetes)
  - More Agile Processes (DevOps)
  - Right Talent
  - Right Culture (of data driven experimentation and innovation)

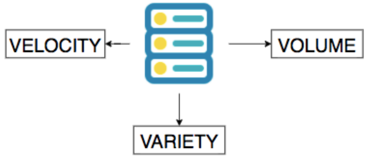
Cloud Mindset

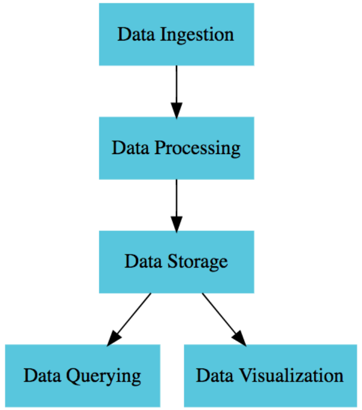
Factor	Data Center	Cloud
Infrastructure	Buy	Rent
Planning	Ahead of time	Provision when you need it
Deployment	VMs	PaaS or Containers or Serverless
Team	Specialized skills	T-shaped skills
Releases	Manual	CI/CD with flexible release options (Canary, A/B Testing, ...)
Infrastructure Creation	Manual	Infrastructure as Code
Attitude	Avoid Failures	Move Fast by Reducing Cost of Failure (Automation of testing, releases, infrastructure creation and monitoring)

Data Analytics

Big Data - Terminology and Evolution

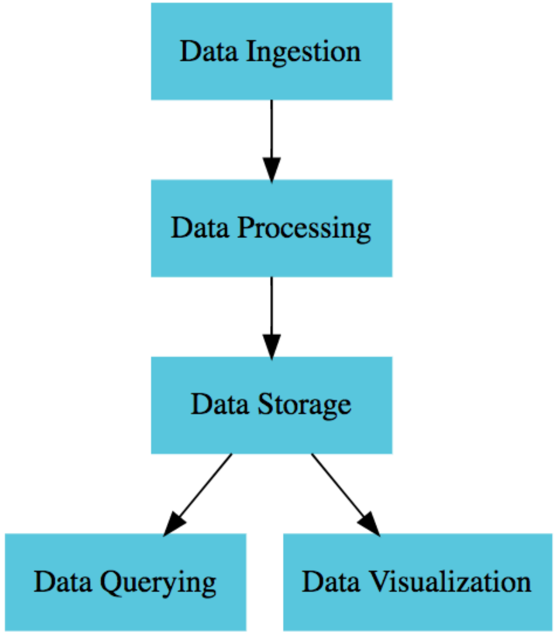
- **3Vs** of Big Data
  - **Volume**: Terabytes to Petabytes to Exabytes
  - **Variety**: Structured, Semi structured, Unstructured
  - **Velocity**: Batch, Streaming ..
- **Terminology**: Data warehouse vs Data lake
  - **Data warehouse**: Execute queries processing petabytes of data in seconds
    - Data stored in a format ready for specific analysis! (processed data)
      - Examples: Teradata, BigQuery(GCP), Redshift(AWS), Azure Synapse Analytics
  - **Data lake**: Typically retains all raw data (compressed)
    - Typically object storage is used as data lake
      - Amazon S3, Google Cloud Storage, Azure Data Lake Storage Gen2 etc..
    - Flexibility while saving cost
    - Perform ad-hoc analysis on demand
    - Analytics & intelligence services (even data warehouses) can directly read from data lake
      - Azure Synapse Analytics, BigQuery(GCP), Amazon Athena etc..





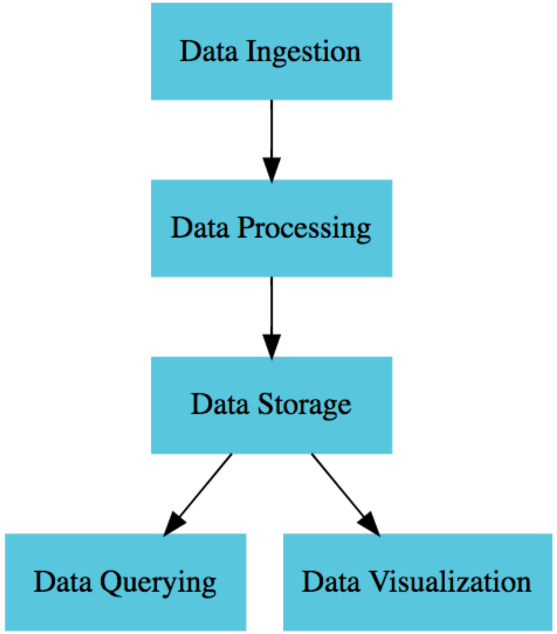
Data Analytics

- **Goal**: Convert raw data to intelligence
  - Uncover trends and discover meaningful information
  - Find new opportunities and identify weaknesses
  - Increase efficiency and improve customer satisfaction
  - Make appropriate business decisions
- Raw data can be **from different sources**:
  - Customer purchases, bank transactions, stock prices, weather data, monitoring devices etc
- **Example**: Decide future sales using past customer behavior
- **Example**: Faster diagnosis & treatment using patient history



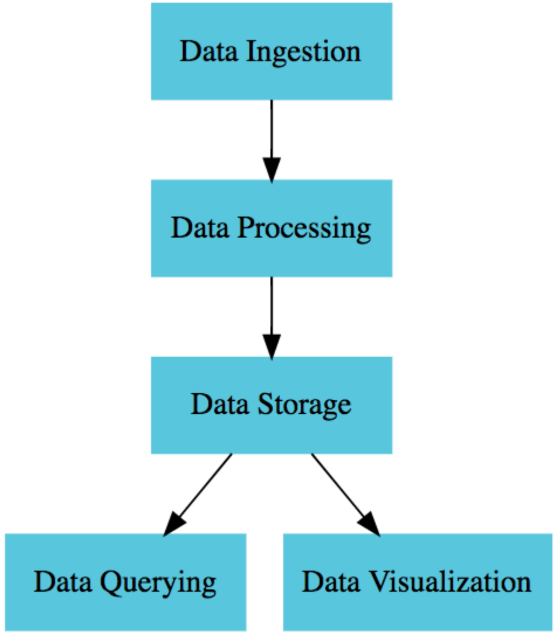
Data Analytics Work Flow

- **Data Ingestion**: Capture raw data
  - From various sources (stream or batch)
    - Example: Weather data, sales records, user actions - websites ..
- **Data Processing**: Process data
  - Raw data is not suitable for querying
    - Clean (remove duplicates), filter (remove anomalies) and/or aggregate data
    - Transform data to required format (Transformation)
- **Data Storage**: Store to data warehouse or data lake
- **Data Querying**: Run queries to analyze data
- **Data Visualization**: Create visualizations to make it easier to understand data and make better decisions
  - Create dashboards, charts and reports (capture trends)
  - Help business spot trends, outliers, and hidden patterns in data



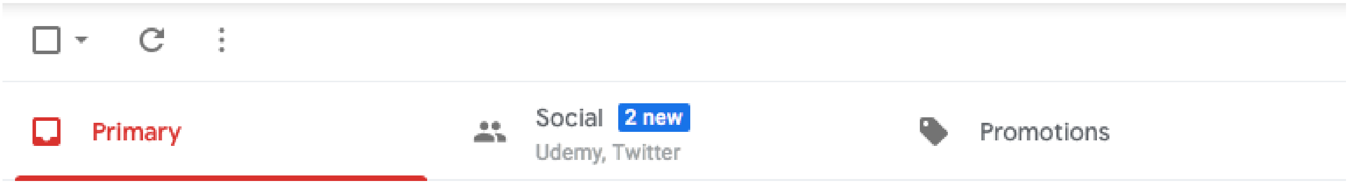
Important AWS Analytics Services

- **AWS Analytics Services:**
  - **Kinesis** - Work with Real-Time Streaming Data
  - **MSK** - Fully managed service for Apache Kafka
  - **Amazon Redshift** - Fast Data Warehousing
  - **EMR** - Managed Hadoop Framework
  - **AWS Glue** - Fully managed ETL
  - **AWS Lake Formation** - Easily set up a secure data lake
  - **Amazon S3** - Data lake storage
  - **Athena** - Query Data in S3 using SQL
  - **QuickSight** - Fast, easy to use business analytics
- **Sample Architectures:**
  - **Streaming** - Kinesis > S3 > Athena
  - **Batch** - Database > AWS Glue > Redshift > QuickSight



Machine Learning

Artificial Intelligence - All around you



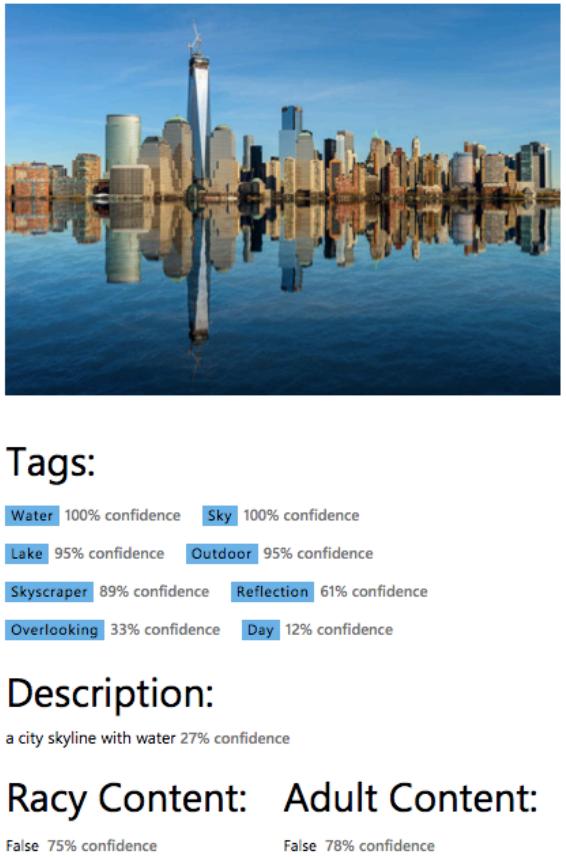
- Self-driving cars
- Spam Filters
- Email Classification
- Fraud Detection

What is AI? (Oxford Dictionary)

The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages

Understanding Types of AI

- **Strong artificial intelligence (or general AI):** Intelligence of machine = Intelligence of human
  - A machine that can solve problems, learn, and plan for the future
  - An expert at everything (including learning to play all sports and games!)
  - Learns like a child, building on it's own experiences
  - We are far away from achieving this! (Estimates: few decades to never)
- **Narrow AI (or weak AI):** Focuses on specific task
  - Examples: Self-driving cars and virtual assistants
  - **Machine learning**: Learn from data (examples)



Tags:

Water 100% confidence Sky 100% confidence

Lake 95% confidence Outdoor 95% confidence

Skyscraper 89% confidence Reflection 61% confidence

Overlooking 33% confidence Day 12% confidence

Description:

a city skyline with water 27% confidence

Racy Content:

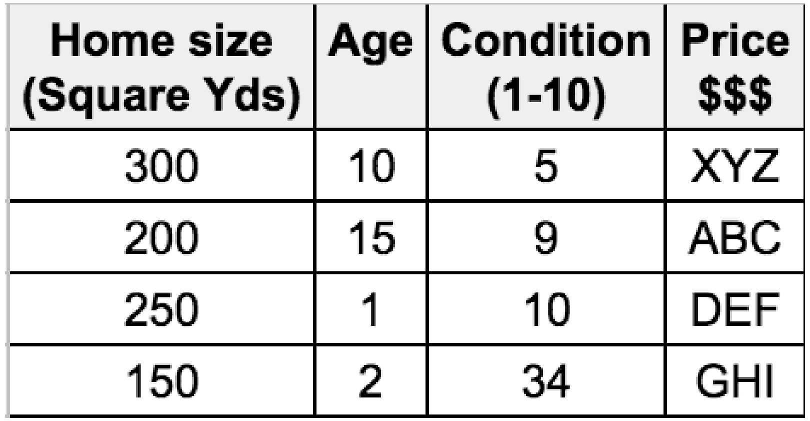
False 75% confidence

Adult Content:

False 78% confidence

Exploring Machine Learning vs Traditional Programming

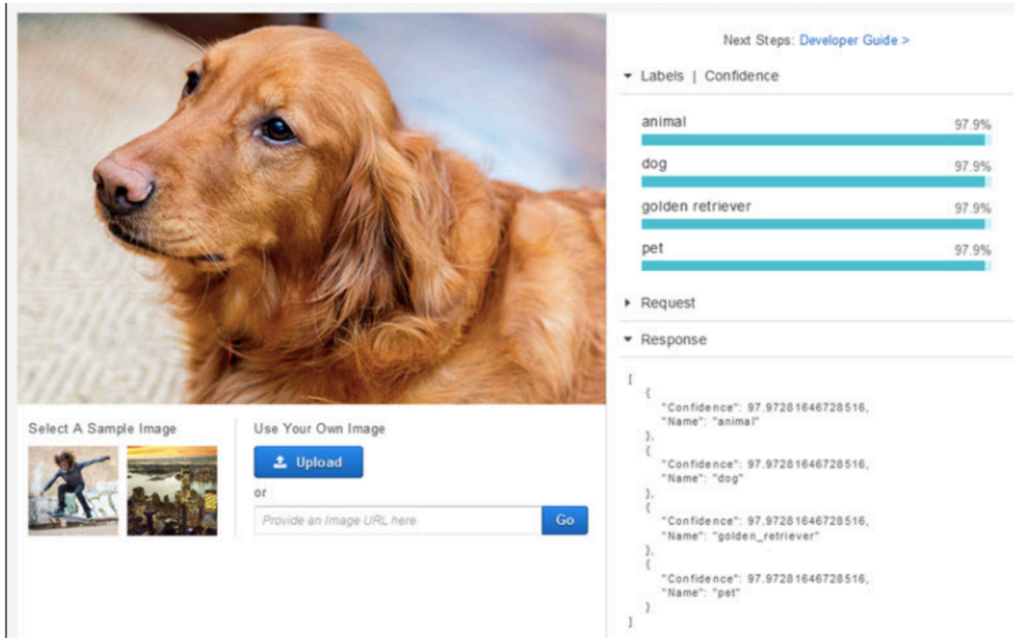
- **Traditional Programming**: Based on Rules
  - IF this DO that
  - Example: Predict price of a home
    - Design an algorithm taking all factors into consideration:
      - Location, Home size, Age, Condition, Market, Economy etc
- **Machine Learning**: Learning from Examples (NOT Rules)
  - Give millions of examples
  - Create a Model
  - Use the model to make predictions!
- **Challenges**:
  - No of examples needed
  - Availability of skilled personnel
  - Complexity in implementing MLOps



Home size (Square Yds)	Age	Condition (1-10)	Price $$$
300	10	5	XYZ
200	15	9	ABC
250	1	10	DEF
150	2	34	GHI

Machine Learning - 3 Approaches

- **Use Pre-Trained Models**
  - Get intelligence from text, images, audio, video
  - Amazon Comprehend, Rekognition,..
- **Build simple models:** Without needing data scientists
  - Limited/no-code experience
  - Example: Amazon SageMaker Auto ML
- **Build complex models:** Using data scientists and team
  - Build Your Own ML Models from ZERO (code-experienced)
  - Example: Amazon SageMaker



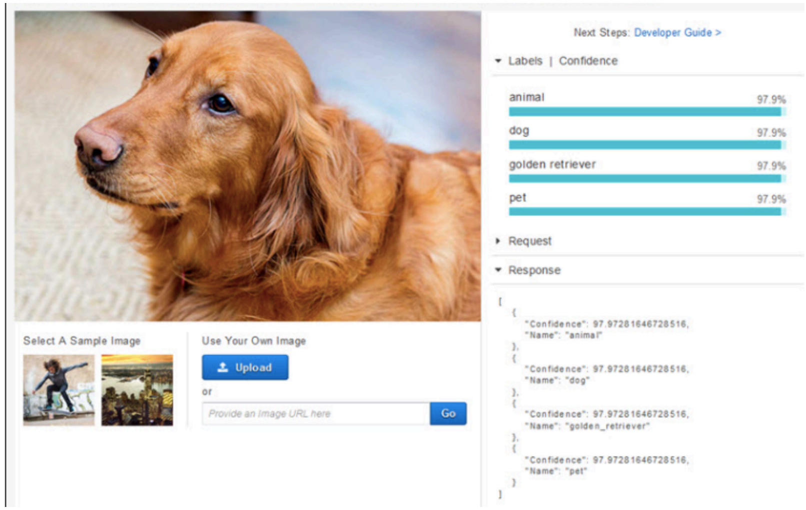
Labels	Confidence
animal	97.9%
dog	97.9%
golden retriever	97.9%
pet	97.9%

Response

```
[
  {
    "Confidence": 97.9728164672851б,
    "Name": "animal"
  },
  {
    "Confidence": 97.9728164672851б,
    "Name": "dog"
  },
  {
    "Confidence": 97.9728164672851б,
    "Name": "golden_retriever"
  },
  {
    "Confidence": 97.9728164672851б,
    "Name": "pet"
  }
]
```

Pre-Trained Models in AWS

- **Amazon Comprehend** - Analyze Unstructured Text
- **Amazon Rekognition** - Search and Analyze Images and Videos
- **Amazon Transcribe** - Powerful Speech Recognition
- **Amazon Polly** - Turn Text into Lifelike Speech
- **Amazon Translate** - Language Translation
- **Amazon Personalize** - Add real-time recommendations to your apps
- **Amazon Fraud Detector** - Detect online fraud faster using machine learning

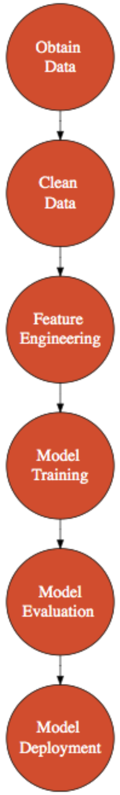


Labels	Confidence
animal	97.9%
dog	97.9%
golden retriever	97.9%
pet	97.9%

```
[
  {
    "Confidence": 97.9728164672851б,
    "Name": "animal"
  },
  {
    "Confidence": 97.97281646728516,
    "Name": "dog"
  },
  {
    "Confidence": 97.97281646728516,
    "Name": "golden_retriever"
  },
  {
    "Confidence": 97.97281646728516,
    "Name": "pet"
  }
]
```

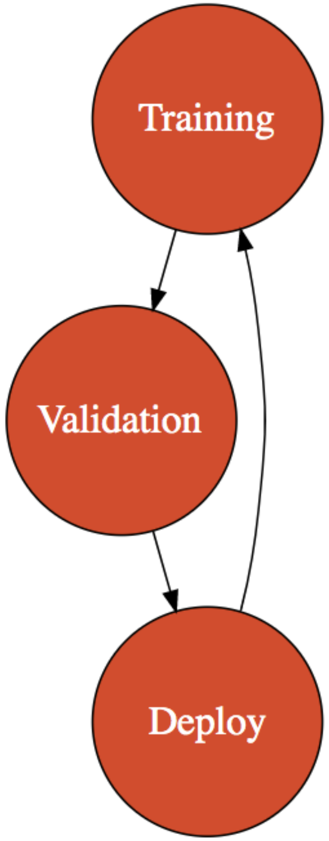
Creating Machine Learning Models - Steps

- **1:** Obtain Data
- **2:** Clean Data
- **3:** Feature Engineering: Identify Features and Label
- **4:** Create a Model using the Dataset and the ML algorithm
- **5:** Evaluate the accuracy of the model
- **6:** Deploy the model for use



Amazon SageMaker

- **Amazon SageMaker**: Simplifies creation of your models
  - Manage data, code, compute, models etc
  - Prepare data
  - Train models
  - Publish models
  - Monitor models
- **Multiple options to create models**
  - **Autopilot**: Build custom models with minimum ML expertise
  - **Build Your Own Models**: Data Scientists
    - Support for deep-learning frameworks such as TensorFlow, Apache MXNet, PyTorch, and more (use them within the built-in containers)
    - Data and compute management, pipelines



Machine Learning Fundamentals - Scenarios

Scenario	Solution
Categorize: Building a computer system as intelligent as a human. An expert at everything (all sports and games!)	Strong AI
Categorize: Building a computer system that focuses on specific task (Self-driving cars, virtual assistants, object detection from images)	Narrow AI (or weak AI)
Category of AI that focuses on learning from data (examples)	Machine learning
Which AWS service helps you analyze if user reviews are positive?	Amazon Comprehend
Which AWS service helps you identify objects in an image?	Amazon Rekognition
Which AWS services helps you build simple models without needing data scientists or AI/ML skills?	Amazon SageMaker Autopilot
Which AWS service helps you build complex ML models?	Amazon SageMaker

You are all set!

Let's clap for you!

- You have a lot of patience! **Congratulations**
- You have put your best foot forward **to learn AWS**
- Don't stop your learning journey!
  - Keep Learning Every Day!
- Good Luck!

Do Not Forget!

- **Recommend** the course to your friends!
  - Do not forget to **review**!
- **Your Success = My Success**
  - Share your success story with me on LinkedIn (Ranga Karanam)
  - Share your success story and lessons learnt in Q&A with other learners!

What Next?

In28 Minutes

FASTEST ROADMAPS

in28minutes.com

Google Cloud Certifications

Azure Certifications

AWS Certifications

DevOps

Java Full Stack

Java Microservices