# AWS Identity Access Management (IAM)

* Allows you to manage users and their level of access to the AWS console.
* Universal/global can be viewed by any region

**What does IAM give you?**

* Centralized control of your AWS account
* Shared access to your AWS account
* Granular Permissions
* Identity Federation (including Active Directory, Facebook, linkedin, etc.)
* Multifactor Authentication
* Provides temporary access for users/devices and services as necessary.
* Allows you to set up your own password rotation policy
* Integrates with many different AWS Services
* Supports PCI DSS Compliance

**Core Concepts**

* Users: end users (think people)
* Groups: A collection of users under one set of permissions. (a group of users and apply policies to them collectively
* Roles: You can create roles and then assign them to AWS resources.
* Policies: A document that defines one (or more) permissions (json form key value pairs)

Under security, identity and compliance

**Best Practices**

* Each user has their own account and then create a group
* Root account is simply the account created when first set up you AWS account it has complete admin access. (DO NOT USE THIS account)

Two types of AWS access

* Programmatic access: How applications access AWS and how you access AWS from the command line.
* AWS console Access: How you access the system via AWS console

New users have NO permissions when first created.

New users are assigned Access key and Secret Access Keys when first created

Use Access key and Secret Access Keys for command line and API access

Only view the secret access key and access key once.

# Elastic Compute Cloud (EC2)

* Secure, resizable compute capacity in the Cloud.
* a Virtual machine hosted in AWS instead of on-premise.
* Pay only for what you use/ pay as you go model
* Select the capacity you need and grow and shrink as you need.

**Pricing Options/Models**

* **On-Demand:** Pay by the hour or second depending on the type of instance you run
  + **Flexible:** Low cost and flexibility of Amazon EC2 without any up-Front payment or long-term commitment
  + **Short-term:** Applications with short-term, spiky, or unpredictable workloads that cannot be interrupted
  + **Testing the water:** Applications being developed or tested on Amazon EC2 for the first time.
* **Reserved Instance:** Reserved capacity for **one or three years**. Up to 72% discount on the hourly charge regional.
  + **Predictable usage:** applications with steady state or predictable usage
  + **Specific Capacity requirement:** Applications that require reserved capacity
  + **Pay-up Front:** You can make up-front payments to reduce their total computing costs even further
  + **Standard RI’s:** up to 72% off on-demand price. Cannot change your mind.
  + **Convertible RI’s:** Up to 54% off on-demand price. Has the option to change to a different Reserved instance type of equal or greater value.
  + **Scheduled RI’s launch within the time window you define.** Match your capacity reservation to a predictable recurring schedule that only requires a fraction of a day, week or month.
* **Spot Instance:** Purchase unused capacity at a discount of up to 90% prices fluctuate with supply and demand.
  + Applications with flexible start and end times.
  + Applications that are only feasible at very low compute prices
  + Applications with an urgent need for large amounts of additional computing
  + Used for genomic sequencing or running calculations for algorithmic trading engines (bitcoin mining)
* **Dedicated instance**: A physical EC2 server dedicated for your use. The most expensive option. STRICT compliance requirements.
  + Meeting compliance requirements
  + Meeting licensing requirements
  + Purchase on demand hourly or reserved up to 70% discount

**Savings Plans**

* Commit 1-3 years for a specific amount of compute power (measured in $/hour) for a one-year or 3 year period.
* Super flexible: not only EC2, also includes serverless technologies like lambda and fargate

**EC2 instance Type Dependencies**

1. **Hardware:** when you launch an instance, the instance type determines the hardware of the host computed used for your instance.
2. **Capabilities:** Each instance type offers different compute, memory, and storage capabilities and are grouped in instance families.
3. **Application Requirements:** select an instance type based on the application being hosted on the instance.

**EC2 instance Types**

* **Compute optimized:** higher ratio of CPU to memory than other families. Example web servers, batch processing, distributed analytics. **Names begin with C**
* **Field Programmable Gate Array (FPGA):** a special processor or card which can be programmed to run specific processing for your application. Really good for applications needing massively parallel processing power.Example Genomics, data analytics, video processing and financial computing. **F1**
* **Graphics Programming Unit (GPU):** provide graphics processing units (GPU’s) along with high CPU and network performance for applications benefiting from highly parallelized processing example image rendering, media processing applications. **Begin with the Letter G**
* **Machine learning:** ASIC instances are powered by chips custom built by AWS and are optimized for running machine learning applications. Example image recognition and speech recognition. **INF for inferential**
* **Memory optimized:** lowest cost per GB of Ram example database applications and large deployments of enterprise applications like SAP and Microsoft SharePoint.
* **Storage optimized: allows you to purchase direct storage for your instances.**

# EBS - Elastic Block Store

Summary - Storage volumes that you attach to EC2 instances.

**Uses**:

* the same way you use system disk
* Create a file system
* Run a database
* Run an operating system
* Store data
* Install applications

**Features**:

* Designed for production workloads
* Highly available and automatically replicated within a single Avail zone to protect against failure
* Scalable

**Types (SSD(solid state drive), Magnetic)**

* General Purpose SSD (GP2) - General purpose, <10,000 IOPS(inputoutputs) this is best price for performance. Suitable for boot disks and general applications. Up to 16,000 Iops per volume and 99.999% durability
* Provisioned IOPS SSD (O1) - Optimized for extreme performance, >10,000 IOPs (most expensive) up to 64,000 IOPS per volume. 50 IOPS per GIB. Suitable for OLTP (Online Transactional Processing)
* Use provisioned if you need more than 16,000 IOPS
* Provisioned IOPS SSD(I02) – latest generation. Higher durability and more IOPS max 64,000. More durable but same price as io1. 500 IOPS per GiB, 99.99999% durability. I/O intensive apps, large databases, and latency-sensitive workloads. Applications which need high levels of durability.

**ex. Gigantic DB**

* Throughput Optimized HDD (ST1) - used for big data, can’t be boot volume. Suitable for big data warehouses,ETL. Max 500 MB/s per volume
* Cold HDD (SC1) - Lowest cost, used for data that isn’t accessed often, file server, can’t be boot volume. Lowest cost
* Magnetic (Standard) - Lowest cost that is bootable LEGACY, don’t really use this, maybe in dev

**IOPS:**

* Measures the number of read and write operations per second
* Important metric for quick transactions, low latency apps, transactional workloads.
* The ability to action reads and writes very quickly
* Choose provisioned IOPS SSD (io1 or io2)

**Throughput:**

* Measures the number of bits read or written per second (MB/s)
* Important metric for large datasets, large I/O sizes, complex queries.
* The ability to deal with large data sets
* Choose Throughput optimized HDD (St1)

Exam tips:

**EBS Volumes are AVAILABILITY ZONE locked!** One of the only things that are AZ specific

# Elastic Load Balancers

**3** **Types**

* Application Load Balancers: make clever decisions, can see up to the application layer and make clever routing or rooting decisions based on known information.
* Network Load Balancer: Layer 4, super-fast performance and speed. (most expensive)
* Classic Load Balancer: not recommended but there for legacy.

## Application Load Balancers

* Best suited for load balancing of HTTP and HTTPS traffic. They operate at Layer 7 and are application aware. They are intelligent, and you can create advanced request routing, sending specified requests to specific web servers.

## Network Load Balancers

* Best suited for loagfd balancing of TCP traffic where extreme performance is required. Operating at the connection level (layer 4), Network Load balancer are capable of handling millions of requests per second, while maintaining ultra-low latencies. **Use for extreme performance! Most costly**
* Classic Load Balancers are the legacy load balancers. You can load balance HTTP/HTTPS applications and use layer 7 specific features, such as x-Forwarded and sticky sessions. You can also use strict layer 4 load balancing for applications that rely purely on the TCP protocol.

**Errors**

* If your application stops responding, the ELB (classic load balancer) responds with a **504 error**. Means application is having issues. Could be the web server layer or the database layer. Identify where the application is failing and scale it up or out where possible.
* X-forwarded-For Header allows you to see public ipv4 address

**RDS 101(Relational Database Service)**

# DynamoDB

A fast and flexible NoSQL database service for all applications that need consistent, **single-digit millisecond** latency at any scale. Fully managed and supports document and key-value data models. Its flexible data model and reliable performance make it a great fit for mobile, web, gaming, ad-tech, IOT, and many other applications.

Summary:

* Serverless
* **Stored on SSD**
* Conditions on IAM Policy to restrict data that isn’t your own using
* dynamodb: LeadingKeys: allows users access to only the items where the partition key value matches their user ID.

Two consistency Models

* Eventual Consistent Reads (Default): consistency across all reads all copies of data within a second. Repeating a read
* Strongly Consistent Reads

Consists of

* Tables
* Items (Think a row of data in a table)
* Attributes (Think of a column of data in a table)
* Supports key-value and document data structures
* Key = The name of the data, Value = the data itself
* Documents can be written in JSON, HTML or XML
* You can use a special IAM condition to restrict user access to only their own records. (ALL Dynamo access is IAM)
* IAM conditions start with statement identifier, Actions: defines the actions that the policy allows, condition, last, attributes: defines the attributes that the policy applies to.

Stores and retrieves data based on a Primary Key.

2 types of Primary Keys

* Partition Key:
  + Unique key is input to an internal (hash function(takes in input such as 1234 and gives an output such as \*5#9253(a ton of random characters).) which determines the partition or physical location on which the data is stored.

\*\* if you are using the partition Key as your Primary key, then no two items can have the same partition Key.

* Composite Key (Partition Key + Sort Key) in combination
  + When you have two partition keys which are the same you would need a sort key.
  + Allows you to store multiple items with the same partition Key

LSI (Local Secondary Indexes)

* Has to be created as you’re creating table
* Has same primary key but can have different sort key

GSI (Global Secondary Indexes)

* Use these when you want to query by something that isn’t the partition key
* Keys in a GSI don’t need to be unique
* The tradeoff to projecting every attribute into an index is the cost to store and write
* If you need to access most of the non-key attributes often project the entire base table, but storage cost will double

Queries:

* Default is eventually consistent but can be strongly consistent
* ProjectionExpression to refine results

Scan:

* Hits whole table
* Can set **smaller page size**
* Use **parallel scans**

Read/Write Capacity Modes:

* On Demand - autoscaling
  + Use if workloads are unknown, unpredictable traffic, want to pay as you go
* Provisioned (free-tier)

Calculating Throughput

* Read (4 KB)
  + One Read Request Unit = 1 4KB strongly consistent read or 2 4KB eventually consistent reads
  + Transactional read requests require 2 Read Request Units
* Write (1 KB)
  + One Write Request Unit = 1 1 KB write
  + Transactional write requests require 2 Write Request Units

Limits:

* 256 tables per region

TTL:

* Can set data to expire
* Expressed in epoch time
* Will be deleted within 48 hours of being marked for deletion

Transactions:

* ACID transactions, all or nothing
* Finances

Tips

* **DAX -** DynamoDB Accelerator (caching service): delivers fast response times for accessing eventually consistent data. Reduces response time from milliseconds to microseconds. Useful for read heavy or bursty workloads. Supports encryption. Don’t use if app requires strongly consistent reads, write intensive, or are caching a different way. (Write through)
* **Atomic Counter** - Allows all write requests to be applied in order to increment/decrement value
* Hot partitions are a thing
* By default writes are **unconditional**, but you can have **conditional** writes, which can be idempotent
* Streams - Logs are stored for 24 hours, tune-ordered seq of modifications
* ProvisionedThroughputExceedException - Use exponential backoff, reduce request frequency. Used in every **AWS SDK**

# SQS - Simple Queueing Service

Summary:

* Message oriented queueing service that allows decoupling of microservices, distributed systems and serverless apps
* NO ORDER - SQS messages can be delivered multiple times in any order
* Design - you can have 2 priority queues for priority-based message one for higher and other for lower priority
* EC2 instances always poll for messages from the queue (**pull** from the queue and not push)
* Long polling can reduce costs, it waits for a message to come in before returning. **Maximum long polling timeout 20 seconds**. This reduces # of empty ReceiveMessageResponse(s) returned
* FIFO queues end in fifo - offer exactly once processing, support multiple producers but not multiple consumers

Limits

* **NO LIMIT** for number of messages in a SQS queue
* MAX of **120,000** in-flight messages (FIFO queue 20,000)
* Message can contain **1-256KB of tex**t, billed at 64KB chunks (For messages bigger than this use SQS Extended Client)
* Single consumers can have **1 to 10** messages unto maximum of 256KB payload
* Messages in the Queue can be retained from **1 min** up to **14 days**
* 3,000 messages per second with batching
* Messages can have any format

Billing:

* First 1 million request are free, then $0.50 PER EVERY MILLION REQUESTS

DLQ - Dead Letter Queues:

* Other queues can put messages here when they aren’t processed successfully
* Main purpose is handling failures

Visibility Timeout

* It is a period of time where it prevents other consumers for processing the message again
* Visibility timeout always start from when the application instance polled the message.
* Visibility timeout expires that means there is a failure somewhere since that message was polled but not processed and hence not deleted so other some other process will poll the message again and visibility timeout starts again.
* Visibility timeout by **default is 30 Seconds** min is **0 seconds**, up to **12 hour** maximum (ChangeMessageVisibility) / maximum visibility
* ChangeMessageVisibility - Changes the timeout of a message in a queue to a new value.

# Kinesis

Increase number of shards and allow partition key to take many attributes in order to scale

A kinesis stream is made up of a set of multiple shards and each shard is a sequence of data records, each with their own sequence number.

The data capacity of your stream is the sum total capacity of its shards.

Per Shard:

* 5 read transactions per second, up to a max of 2MB per second
* 1,000 write records per second, up to a max of 1MB per second.
* As the data rate increases, you increase the number of shards. (resharding)
* Consumers are usually ec2 instances consuming data from your streams

Kinesis Client Library

The kinesis client library running on your consumers creates a record processor for each shard that is being consumed by your instance

If you increase the number of shards, the KCL will add more record processors on your consumers.

CPU utilization is what should drive the quantity of consumer instances you have, NOT the number of shards in your kinesis stream.

Use an auto scaling group, an base scaling decisions on CPU load on your consumers.

**Graphical user interface

Description automatically generated**

# Elastic Beanstalk

Summary

* Service for deploying and scaling web applications and services
* Supports Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Tomcat, Nginx, Passenger, and IIS
* Makes it so developer don’t have to worry about underlying resources
* Compute Service

Deployments

* **All at once** - default, Deploy the new version to all instances at the same time. All instances are out of service for a short time (**Downtime**)
* **Rolling** - Elastic Beanstalk splits the environment into batches and updates them one batch at a time. Some instances will serve old version and some new until deployment is complete
* **Rolling deployment with an additional batch** - Will launch a new batch of instances prior to taking any instances out of service. Use if you want to maintain full capacity during deployments. After deployment additional batch is terminated
* **Immutable** - Launches full set of new instances running new version alongside old instances. If EC2 instances dont pass health checks they get terminated and originals stay. If you want to allow instances passing with lower status you can modify the **Healthy threshold**. Optionally you can override with **ignore health check** option
* Blue/Green - Use if you want to manually do immutable
* **Traffic Splitting** – Installs the new version on a new set of instances, like an immutable deployment. Forwards a percentage of incoming client traffic to the new application version for evaluation. To test the new deployment while leaving your original up. Enabling canary testing. Canary Testing – Installs the new version on a new set of instances just like an immutable launch.

.ebextensions/

* **.ebextensions/settings.config** - This file allows you to configure your environment. (.config file)
* Any resources created in ebextensions will be deleted if environment is terminated
* If you want them to last you must define them externally
* Written in Yaml or Json

Tips:

* Uses Cloud Formation to deploy applications.
* Worker environments can be used for task that take a long time to complete
* For RDS, deploy it by itself first, then connect to Beanstalk. Must create security group and connection string