|  |  |
| --- | --- |
| Image result for aws picture  Amazon Web Services Associate Certification Exams  Study Notes | objective  A high-level overview of the well-architected framework and accompanying services typically asked in the AWS Associate-level Certification exams.  Akshay Rai |

# **References**

* Official AWS Documentation, whitepapers and FAQs.
  + <https://docs.aws.amazon.com/>
  + <https://aws.amazon.com/whitepapers/>
  + <https://aws.amazon.com/faqs/>
* AWS re:invent videos.
  + <https://www.youtube.com/channel/UCd6MoB9NC6uYN2grvUNT-Zg> (Official YouTube Channel)
* [Stephane Maarek’s](https://www.udemy.com/user/stephane-maarek/)\*\* AWS Certification Courses on Udemy.
  + [Ultimate AWS Certified Developer Associate 2019](https://www.udemy.com/course/aws-certified-developer-associate-dva-c01/)
  + [Ultimate AWS Certified SysOps Adminstrator Associate 2019](https://www.udemy.com/course/ultimate-aws-certified-sysops-administrator-associate/)
* CloudGuru/ [Ryan Kroonenburg’s](https://www.udemy.com/user/ryankroonenburg/) AWS Certification Courses on Udemy.
  + [AWS Certified Solutions Architect – Associate 2019](https://www.udemy.com/course/aws-certified-solutions-architect-associate/)
* Tutorials Dojo/ [Jon Bonso’s](https://www.udemy.com/user/jonjonbonso/) Practice exams on Udemy & their [notes.](https://tutorialsdojo.com/aws-study-path-notes/)

\*\*Special thanks for Stephane Maarek for allowing me to take excepts from his own notes.

# **Index**

**Table of Contents**

[References 1](#_Toc28020922)

[Index 2](#_Toc28020923)

[Cloud Best Practices 12](#_Toc28020924)

[Business benefits of Cloud 12](#_Toc28020925)

[Technical benefits of Cloud 12](#_Toc28020926)

[Planning for failure 13](#_Toc28020927)

[Decoupling components 13](#_Toc28020928)

[Implementing elasticity 13](#_Toc28020929)

[Securing the application 13](#_Toc28020930)

[The Well-Architected Framework 14](#_Toc28020931)

[General Design Principles 15](#_Toc28020932)

[Security 15](#_Toc28020933)

[Reliability 17](#_Toc28020934)

[Performance Efficiency 18](#_Toc28020935)

[Cost Optimization 20](#_Toc28020936)

[Operational Excellence 22](#_Toc28020937)

[Identity Access Management (IAM) 24](#_Toc28020938)

[Definition 24](#_Toc28020939)

[Use Cases 24](#_Toc28020940)

[Key Terminologies 25](#_Toc28020941)

[MFA 25](#_Toc28020942)

[Miscellaneous Features 25](#_Toc28020943)

[Simple Storage Service (S3) 26](#_Toc28020944)

[Definition 26](#_Toc28020945)

[Storage Classes 26](#_Toc28020946)

[Use Cases 27](#_Toc28020947)

[Performance 28](#_Toc28020948)

[Durability and Availability 28](#_Toc28020949)

[Scalability and Elasticity 28](#_Toc28020950)

[Security 28](#_Toc28020951)

[Cost Model 29](#_Toc28020952)

[Miscellaneous Features 29](#_Toc28020953)

[Storage Gateway 31](#_Toc28020954)

[Definition 31](#_Toc28020955)

[Benefits of Storage Gateway 31](#_Toc28020956)

[Storage Solutions 32](#_Toc28020957)

[Security 33](#_Toc28020958)

[Cost Model 33](#_Toc28020959)

[Miscellaneous features 33](#_Toc28020960)

[Snowball 34](#_Toc28020961)

[Definition 34](#_Toc28020962)

[Use Cases 34](#_Toc28020963)

[Important Components 35](#_Toc28020964)

[Performance 35](#_Toc28020965)

[Security 35](#_Toc28020966)

[Cost Model 35](#_Toc28020967)

[Miscellaneous Features 35](#_Toc28020968)

[Snowball Edge 36](#_Toc28020969)

[Definition 36](#_Toc28020970)

[Use Cases 36](#_Toc28020971)

[Elastic Cloud Compute (EC2) 37](#_Toc28020972)

[Definition 37](#_Toc28020973)

[Instance Types 37](#_Toc28020974)

[Purchasing Options 39](#_Toc28020975)

[Amazon Machine Image (AMI) 40](#_Toc28020976)

[Instance Store Volumes 41](#_Toc28020977)

[Placement Groups 41](#_Toc28020978)

[Security Groups 43](#_Toc28020979)

[AutoScaling Groups (ASG) 43](#_Toc28020980)

[Troubleshooting 44](#_Toc28020981)

[Miscellaneous Features 45](#_Toc28020982)

[Elastic Block Store (EBS) 46](#_Toc28020983)

[Definition 46](#_Toc28020984)

[Snapshot Logic 46](#_Toc28020985)

[Usage Patterns 47](#_Toc28020986)

[Performance 47](#_Toc28020987)

[Durability and Availability 48](#_Toc28020988)

[Scalability and Elasticity 48](#_Toc28020989)

[Security 49](#_Toc28020990)

[Cost Model 49](#_Toc28020991)

[Miscellaneous Features 49](#_Toc28020992)

[Elastic File System (EFS) 50](#_Toc28020993)

[Definition 50](#_Toc28020994)

[Use Cases 50](#_Toc28020995)

[Performance 51](#_Toc28020996)

[Durability and Availability 52](#_Toc28020997)

[Scalability and Elasticity 52](#_Toc28020998)

[Security 52](#_Toc28020999)

[Cost Model 53](#_Toc28021000)

[Miscellaneous Features 53](#_Toc28021001)

[Relational Database Service (RDS) 54](#_Toc28021002)

[Definition 54](#_Toc28021003)

[DB Instances 55](#_Toc28021004)

[Performance 55](#_Toc28021005)

[Backup & Recovery 56](#_Toc28021006)

[Security 57](#_Toc28021007)

[Cost Model 57](#_Toc28021008)

[Miscellaneous Features 57](#_Toc28021009)

[DynamoDB 58](#_Toc28021010)

[Definition 58](#_Toc28021011)

[Use Cases 58](#_Toc28021012)

[Key Components 59](#_Toc28021013)

[Performance 60](#_Toc28021014)

[Throughput 60](#_Toc28021015)

[DynamoDB Streams 61](#_Toc28021016)

[DynamoDB Basic APIs 61](#_Toc28021017)

[Miscellaneous Features 63](#_Toc28021018)

[Redshift 64](#_Toc28021019)

[Definition 64](#_Toc28021020)

[Performance 65](#_Toc28021021)

[Deployment Mode 65](#_Toc28021022)

[Backup & Recovery 66](#_Toc28021023)

[Miscellaneous Features 66](#_Toc28021024)

[Aurora 67](#_Toc28021025)

[Definition 67](#_Toc28021026)

[Performance 67](#_Toc28021027)

[Backup & Recovery 68](#_Toc28021028)

[Aurora Serverless 69](#_Toc28021029)

[Miscellaneous Features 69](#_Toc28021030)

[ElastiCache 70](#_Toc28021031)

[Definition 70](#_Toc28021032)

[Caching Engines 71](#_Toc28021033)

[CloudWatch 71](#_Toc28021034)

[Definition 71](#_Toc28021035)

[Data Retention 72](#_Toc28021036)

[Dashboards 72](#_Toc28021037)

[Metrics 72](#_Toc28021038)

[Alarms 73](#_Toc28021039)

[Events 73](#_Toc28021040)

[Logs 73](#_Toc28021041)

[Miscellaneous Features 74](#_Toc28021042)

[CloudFront 74](#_Toc28021043)

[Definition 74](#_Toc28021044)

[CloudFront Distributions 75](#_Toc28021045)

[Performance and Availability 75](#_Toc28021046)

[Origin Access Identity 75](#_Toc28021047)

[Miscellaneous Features 76](#_Toc28021048)

[CloudTrail 76](#_Toc28021049)

[Definition 76](#_Toc28021050)

[Trails 77](#_Toc28021051)

[Route 53 77](#_Toc28021052)

[Definition 77](#_Toc28021053)

[DNS Record Sets 78](#_Toc28021054)

[Routing Policies 79](#_Toc28021055)

[Health Checks 79](#_Toc28021056)

[Miscellaneous Features 80](#_Toc28021057)

[Virtual Private Cloud (VPC) 80](#_Toc28021058)

[Definition 80](#_Toc28021059)

[Subnets 81](#_Toc28021060)

[Internet Gateway (IGW) 81](#_Toc28021061)

[NAT Instances 82](#_Toc28021062)

[NAT Gateways 82](#_Toc28021063)

[Network Access Control List (NACL) 82](#_Toc28021064)

[VPC Flow Logs 83](#_Toc28021065)

[Bastion Hosts 83](#_Toc28021066)

[Direct Connect 84](#_Toc28021067)

[VPC Endpoints 84](#_Toc28021068)

[VPC Peering 84](#_Toc28021069)

[Miscellaneous Features 85](#_Toc28021070)

[Elastic Load Balancing (ELB) 85](#_Toc28021071)

[Definition 85](#_Toc28021072)

[Elastic Load Balancer Types 86](#_Toc28021073)

[Advanced Concepts 87](#_Toc28021074)

[Cost Model 88](#_Toc28021075)

[Load Balancer Scheme 88](#_Toc28021076)

[Miscellaneous Features 88](#_Toc28021077)

[CloudFormation 89](#_Toc28021078)

[Definition 89](#_Toc28021079)

[Templates 90](#_Toc28021080)

[Intrinsic Functions 92](#_Toc28021081)

[Rollback 93](#_Toc28021082)

[Deletion Policy 93](#_Toc28021083)

[Miscellaneous Features 94](#_Toc28021084)

[Simple Queue Service (SQS) 96](#_Toc28021085)

[Definition 96](#_Toc28021086)

[Use Cases 97](#_Toc28021087)

[Queue Types 97](#_Toc28021088)

[Miscellaneous Features 98](#_Toc28021089)

[Simple Workflow Service (SWF) 99](#_Toc28021090)

[Definition 99](#_Toc28021091)

[Actors 99](#_Toc28021092)

[SQF vs SQS 100](#_Toc28021093)

[Elastic Beanstalk 100](#_Toc28021094)

[Definition 100](#_Toc28021095)

[Key Concepts 100](#_Toc28021096)

[Deployment Modes 101](#_Toc28021097)

[Advanced Concepts 103](#_Toc28021098)

[Troubleshooting 103](#_Toc28021099)

[Simple Notification Service (SNS) 104](#_Toc28021100)

[Definition 104](#_Toc28021101)

[Benefits 105](#_Toc28021102)

[Miscellaneous Features 105](#_Toc28021103)

[Elastic Transcoder 106](#_Toc28021104)

[Definition 106](#_Toc28021105)

[API Gateway 107](#_Toc28021106)

[Definition 107](#_Toc28021107)

[Use Cases 108](#_Toc28021108)

[Deployment Stages 108](#_Toc28021109)

[API Gateway Caching 108](#_Toc28021110)

[Cross-Origin Resource Sharing (CORS) 109](#_Toc28021111)

[Security 109](#_Toc28021112)

[Cost Model 109](#_Toc28021113)

[Miscellaneous Features 110](#_Toc28021114)

[Kinesis 110](#_Toc28021115)

[Definition 110](#_Toc28021116)

[Kinesis Types 111](#_Toc28021117)

[Consumer Models 111](#_Toc28021118)

[Miscellaneous Features 112](#_Toc28021119)

[Cognito 113](#_Toc28021120)

[Definition 113](#_Toc28021121)

[Pool types 114](#_Toc28021122)

[Cognito Sync 115](#_Toc28021123)

[Simple Token Service (STS) 115](#_Toc28021124)

[Definition 115](#_Toc28021125)

[How it works 116](#_Toc28021126)

[Lambda 116](#_Toc28021127)

[Definition 116](#_Toc28021128)

[Lambda Configurations 117](#_Toc28021129)

[Service Limits 117](#_Toc28021130)

[Cost Model 118](#_Toc28021131)

[Miscellaneous Features 118](#_Toc28021132)

[AWS X-Ray 119](#_Toc28021133)

[Definition 119](#_Toc28021134)

[Use Cases 120](#_Toc28021135)

[How it works 120](#_Toc28021136)

[Security 120](#_Toc28021137)

[Elastic Container Service (ECS) 120](#_Toc28021138)

[Definition 120](#_Toc28021139)

[Task Definition 121](#_Toc28021140)

[ECS Service 121](#_Toc28021141)

[Elastic Container Registry (ECR) 121](#_Toc28021142)

[Miscellaneous Features 122](#_Toc28021143)

[CodeCommit 122](#_Toc28021144)

[Definition 122](#_Toc28021145)

[Security 123](#_Toc28021146)

[Notifications 123](#_Toc28021147)

[CodePipeline 124](#_Toc28021148)

[Definition 124](#_Toc28021149)

[CodeBuild 125](#_Toc28021150)

[Definition 125](#_Toc28021151)

[Security 126](#_Toc28021152)

[Troubleshooting 126](#_Toc28021153)

[CodeDeploy 126](#_Toc28021154)

[Definition 126](#_Toc28021155)

[Primary Components 126](#_Toc28021156)

[How it works 127](#_Toc28021157)

[AppSpec File 127](#_Toc28021158)

[Deployment Modes 128](#_Toc28021159)

[AWS Simple Systems Manager (SSM) 128](#_Toc28021160)

[Definition 128](#_Toc28021161)

[Run Command 129](#_Toc28021162)

[Inventory 129](#_Toc28021163)

[Patch Manager 129](#_Toc28021164)

[State Manager 129](#_Toc28021165)

[Session Manager 129](#_Toc28021166)

[Parameter Store 129](#_Toc28021167)

[Miscellaneous Services 130](#_Toc28021168)

[AWS Key Management Service (KMS) 130](#_Toc28021169)

[AWS Config 131](#_Toc28021170)

[AWS Organizations 131](#_Toc28021171)

[AWS Cost Explorer 132](#_Toc28021172)

[AWS Budgets 132](#_Toc28021173)

[AWS Certificate Manager 132](#_Toc28021174)

[AWS Shield 133](#_Toc28021175)

[AWS Web Application Firewall (WAF) 133](#_Toc28021176)

[AWS Inspector 133](#_Toc28021177)

[AWS Trusted Advisor 134](#_Toc28021178)

[AWS GuardDuty 134](#_Toc28021179)

[AWS CloudHSM 134](#_Toc28021180)

[AWS Directory Service 135](#_Toc28021181)

[AWS Step Functions 135](#_Toc28021182)

[AWS OpsWorks 135](#_Toc28021183)

[AWS IOT Core 135](#_Toc28021184)

[AWS Data Migration Service 136](#_Toc28021185)

[AWS Serverless Application Model 136](#_Toc28021186)

# Cloud Best Practices

## Business benefits of Cloud

* Almost zero upfront infrastructure investment
* Just-in-time architecture (Autoscaling)
* More efficient resource utilization
* Usage-based costing / utility billing
* Reduced time to market

## Technical benefits of Cloud

* Automation - Scriptable infrastructure
* Auto-scaling - Automated elasticity + scalability
* Proactive scaling
* More efficient development cycle
* Improved testability
* Disaster recovery and business continuity
* "Overflow" the traffic to the cloud

## Planning for failure

* Always assume hardware may fail/ outages may occur/ expected number of requests per second might increase/
* Think about automated recovery from failure strategies during design time, which helps designing an overall system better.

## Decoupling components

* Build components that do not have a tight dependency with each other, so that if one component were to malfunction for some reason, the other components in the system are built so as to continue to work as if no failure is happening.
* In essence loose coupling isolates the various layers and components of an application so that each component interacts asynchronously with the others and treats them as a black box.

## Implementing elasticity

* Proactive Cyclic Scaling
  + Periodic scaling that occurs at fixed intervals (daily, weekly, monthly, quarterly).
* Proactive Event-based Scaling
  + Scaling just when one is expecting a big surge of traffic requests due to a scheduled business event (new product launch, marketing campaigns).
* Auto-scaling based on demand
  + By using a monitoring service, the system can send triggers to take appropriate actions so that it scales up or down based on metrics (utilization of the servers or network I/O, for instance).

## Securing the application

* Only allow appropriate ports for each layer.
  + 80 (HTTP) and 443 (HTTPS) for Web layer for accessing the internet.
  + 22 (SSH) for application layer for developers to log into.
* Only permit Web layer to access Application Layer
* Only permit Application layer to access Database Layer.

# The Well-Architected Framework

The AWS Well-Architected Framework is based on five pillars

* Security
  + The ability to protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.
* Reliability
  + The ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.
* Performance Efficiency
  + The ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve.
* Cost Optimization
  + The ability to run systems to deliver business value at the lowest price point.
* Operational Excellence
  + The ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures.

## General Design Principles

* Do not guess capacity needs
  + One might end up sitting on expensive idle resources or dealing with performance implications of limited capacity. One should scale elastically up and down as required.
* Test systems at production scale
  + One can create a production-scale testing environment on demand, complete the testing, and then decommission the resources.
* Automate to make architectural experimentation easier
  + Allows one to create and replicate systems at low cost and avoid expenses of manual effort.
  + One can track changes to the automation, audit the impact, and revert to previous parameters when necessary.
* Allow for evolutionary architectures
  + The capability to automate and test on demand lowers the risk of impact from design changes.
* Drive architecture using data
  + One can collect data on how architectural choices affect the behavior of a workload.
  + Using this data, one can make fact-driven decisions on how to improve the workload.
* Improve through game days
  + Simulate events in production to test how the architecture and processes perform.

## Security

### Design Principles

* Implement a strong identity foundation
  + Implement the principle of least privilege and enforce separation of duties with appropriate finely-grained authorization for each interaction with AWS resources.
  + Centralize privilege management and reduce or even eliminate reliance on long term credentials.
* Enable traceability:
  + Monitor, alert, and audit actions and changes to the environment in real time.
  + Integrate logs and metrics with systems to automatically respond and take action.
* Apply security at all layers:
  + Rather than just focusing on protection of a single outer layer, apply a defense-in-depth approach with other security controls.
  + Apply to all layers (e.g., edge network, VPC, subnet, load balancer, every instance, operating system, and application).
* Automate security best practices:
  + Automated software-based security mechanisms improve the ability to securely scale more rapidly and cost effectively.
  + Create secure architectures, including the implementation of controls that are defined and managed as code in version-controlled templates.
* Protect data in transit and at rest:
  + Classify your data into sensitivity levels and use mechanisms, such as encryption, tokenization, and access control where appropriate.
* Keep people away from data:
  + Create mechanisms and tools to reduce or eliminate the need for direct access or manual processing of data. This reduces the risk of loss or modification and human error when handling sensitive data.
* Prepare for security events:
  + Prepare for an incident by having an incident management process that aligns to your organizational requirements.
  + Run incident response simulations and use tools with automation to increase your speed for detection, investigation, and recovery.

### Definition

* Data Protection
  + Classify data into segments such as publicly available, available only to members of the organization, only to members of the board, etc.
  + Implement a least privilege access system so that people are only able to access what they need.
  + Tokenization and encryption of data should be done wherever possible, both at rest and at transit.
  + Define data backup, replication, and recovery approach to help protect against deletion or destruction of data.
  + Versioning, as part of a larger data lifecycle-management process, can protect against accidental overwrites, deletes and similar harm.
  + Mission-critical data should be stored on durable storage services designed for exception resiliency.
* Privilege management
  + Only authorized and authenticated users are able to access resources, and only in a manner that is intended.
  + It includes access control lists (ACLs), role-based access controls, and password management (such as powerful rotation policies).
* Infrastructure protection
  + Protecting network and host-level boundaries i.e. the careful management of the network topology and provision of isolation and boundaries for resources within the environment.
* Detective controls
  + They are an essential part of governance frameworks and can be used to support a quality process, a legal or compliance obligation, and for threat identification and response efforts.
  + Logs should be captured and analyzed, especially those pertaining to file access and changes.
  + Auditing controls should be integrated with notification and workflows.

## Reliability

### Design Principles

* Test recovery procedures
  + Create failure scenarios that expose failure pathways so that they can be rectified before a real failure occurs.
* Automatically recover from failure
  + By monitoring system for key performance indicators, automation can be trigger for pre-defined thresholds.
  + This allows for automatic tracking and notification of failures, and possibility anticipate and remediate failures before they occur.
* Scale horizontally to increase aggregate system availability
  + Replace large resources with multiple small resources to reduce the impact of a single point of failure.
* Stop guessing capacity
  + Monitor demand and system utilization and automate addition or removal of resources to maintain optimal level to satisfy demand.
* Manage change in automation
  + Changes to the infrastructure should be done using automation.
  + Changes that need be to be managed are changes in automation.

### Definition

* Foundations
  + Before architecting any solution, foundational requirements that influence reliability should be set in place, such as sufficient network bandwidth to one’s data center, service limits, etc.
* Change Management
  + Being aware of how change affects a system allows you to plan proactively, and monitoring allows you to quickly identify trends that could lead to capacity issues or SLA breaches.
* Failure Management
  + Taking advantage of automation, one can trigger actions due to changes on monitored key performance indicators.
  + One can provision a new source whilst carrying out analysis on failed resource.

## Performance Efficiency

### Design Principles

* Democratize advanced technologies
  + Instead of expending resources to teach on-premise team on how to host and run a new technology, consume it as a service while focusing on product development rather than resource provisioning and management.
* Go global in minutes
  + Deploy system in multiple regions to provide lower latency and a better experience for end-users at minimum cost.
* Use serverless architectures
  + Remove the need to run and maintain servers to carry out traditional compute activities, eliminating the operational burden of managing said servers, whilst reducing cost.
* Experiment more often
  + Carry out comparative testing with different types of instance, storage or configurations.
* Mechanical sympathy
  + Use the technology approach that aligns best with what is trying to be achieved. E.g. Consider data access patterns when selecting database or storage approaches.

### Definition

* Selection
  + Compute
    - Compute is available in three forms
      * Instances Virtualized servers of various families and sizes.
      * Containers Virtualized OS in resource-isolated processes.
      * Functions Abstracting execution environment from code.
    - The optimal compute solution for a system varies based on application design, usage patterns, and configuration settings. Architectures may use different compute solutions for various components and enable different features to improve performance.
    - When architecting use of compute, one should take advantage of the elasticity mechanisms available to ensure sufficient capacity to sustain performance as demand changes.
  + Storage
    - The optimal storage solution for a particular system will vary based on the kind of access method (block, file, or object), patterns of access (random or sequential), throughput required, frequency of access (online, offline, archival), frequency of update (WORM, dynamic), and availability and durability constraints.
    - Well-architected systems use multiple storage solutions and enable different features to improve performance.
  + Database
    - The optimal database solution for a particular system can vary based on requirements for availability, consistency, partition tolerance, latency, durability, scalability, and query capability.
    - Many systems use different database solutions for various subsystems and enable different features to improve performance.
    - As with storage, it is critical to consider the access patterns of your workload, and also to consider if other non-database solutions could solve the problem more efficiently (such as using a search engine or data warehouse).
  + Network
    - The optimal network solution for a particular system will vary based on latency, throughput requirements and so on.
    - Physical constraints such as user or on-premises resources will drive location options, which can be offset using edge techniques or resource placement.
* Review
  + When architecting workloads, there are finite options that one can choose from.
  + However, over time, new technologies and approaches become available that could improve the performance of the workload.
* Monitoring
  + After one has implemented the architecture, they will need to monitor its performance so that any issues can be remediated before end-users are aware.
  + Monitoring metrics should be used to raise alarms when thresholds are breached. The alarm can trigger automated action to work around any badly performing components.
* Tradeoffs
  + When architecting solutions, actively considering tradeoffs enables one to select an optimal approach. Often one can improve performance by trading consistency, durability, and space for time and latency.
  + Tradeoffs can increase the complexity of your architecture and require load testing to ensure that a measurable benefit is obtained.

## Cost Optimization

### Design Principles

* Adopt a consumption model
  + Pay only for the computing resources that you require and increase or decrease usage depending on business requirements, not by using elaborate forecasting
* Measure overall efficiency
  + Measure the business output of the workload and the costs associated with delivering it.
  + Use this measure to know the gains you make from increasing output and reducing costs.
* Stop spending money on data center operations
  + AWS does the heavy lifting of racking, stacking, and powering servers, so one can focus on their customers and organization projects rather than on IT infrastructure.
* Analyze and attribute expenditure
  + The cloud makes it easier to accurately identify the usage and cost of systems, which then allows transparent attribution of IT costs to individual workload owners.
  + This helps measure return on investment (ROI) and gives workload owners an opportunity to optimize their resources and reduce costs.
* Use managed and application level services to reduce cost of ownership
  + Managed and application level services remove the operational burden of maintaining servers for tasks such as sending email or managing databases.
  + As managed services operate at cloud scale, they can offer a lower cost per transaction or service.

### Definition

* Expenditure Awareness
  + Establish policies and mechanisms to ensure that appropriate costs are incurred while objectives are achieved. By employing a checks-and-balances approach, you can innovate without overspending
* Cost-Effective Resources
  + By selecting the appropriate building blocks and managed services, one can optimize workloads for cost.
  + For example, using managed services, one can reduce or remove much of their administrative and operational overhead, freeing them to work on applications and business-related activities.
* Matching supply and demand
  + For a workload that has balanced spend and performance, ensure that everything one pays for is used and avoid significantly underutilizing instances.
  + A skewed utilization metric in either direction has an adverse impact on the organization, in either operational costs (degraded performance due to over-utilization) or wasted AWS expenditures (due to over-provisioning).
  + When designing to match supply against demand, actively think about the patterns of usage and the time it takes to provision new resources.
* Optimizing over time
  + As AWS releases new services and features, it is a best practice to review existing architectural decisions to ensure they continue to be the most cost effective.

## Operational Excellence

### Design Principles

* Perform operations as code
  + One can define their entire workload (applications, infrastructure) as code and update it with code.
  + One can implement your operations procedures as code and automate their execution by triggering them in response to events.
  + By performing operations as code, one limits human error and enable consistent responses to events.
* Annotate documentation
  + One can automate the creation of annotated documentation after every build (or automatically annotate hand-crafted documentation).
  + Use annotations as an input to operations code.
* Make frequent, small, reversible changes
  + Design workloads to allow components to be updated regularly.
  + Make changes in small increments that can be reversed if they fail (without affecting customers when possible).
* Refine operations procedures frequently
  + As you use operations procedures, look for opportunities to improve them. As you evolve your workload, evolve procedures appropriately.
* Anticipate failure
  + Perform “pre-mortem” exercises to identify potential sources of failure so that they can be removed or mitigated.
  + Test failure scenarios and validate the understanding of their impact.
  + Test response procedures to ensure that they are effective, and that teams are familiar with their execution.
* Learn from all operation failures
  + Drive improvement through lessons learned from all operational events and failures.
  + Share what is learned across teams and through the entire organization.

### Definition

* Prepare
  + Effective preparation is required to drive operational excellence. Business success is enabled by shared goals and understanding across the business, development, and operations
  + Design the workload so that it provides the information necessary for you to understand its internal state (for example, metrics, logs, and traces) across all components.
  + Adopt approaches that improve flow of changes into production, that enable refactoring, fast feedback on quality, and bug fixing.
  + Invest in implementing operations activities as code to maximize the productivity of operations personnel, minimize error rates, and enable automated responses.
  + Adopt deployment practices that take advantage of the elasticity of the cloud to facilitate pre-deployment of systems for faster implementations.
* Operate
  + Operational health includes both the health of the workload and the health and success of the operations acting upon the workload (for example, deployment and incident response).
  + Establish baselines from which improvement or degradation of operations will be identified, collect and analyze metrics, and then validate the understanding of operations success and how it changes over time.
  + Use collected metrics to determine if one is satisfying customer and business needs and identify areas for improvement.
  + Use established runbooks for well-understood events and use playbooks to aid in the resolution of other events.
  + Prioritize responses to events based on their business and customer impact.
  + Ensure that if an alert is raised in response to an event, there is an associated process to be executed, with a specifically identified owner.
* Evolve
  + Evolution of operations is required to sustain operational excellence.
  + Dedicate work cycles to making continuous incremental improvements.
  + Regularly evaluate and prioritize opportunities for improvement (for example, feature requests, issue remediation, and compliance requirements), including both the workload and operations procedures.
  + Include feedback loops within your procedures to rapidly identify areas for improvement and capture learnings from the execution of operations.

# Identity Access Management (IAM)

## Definition

* IAM enables one to manage users and their level of access to the AWS Cloud.

## Use Cases

* Provides centralized control of the AWS account.
* Provides shared access to the AWS account.
  + There is no need to share password or access keys.
* Provides granular permissions to users.
  + The notion of giving least privilege to a user should be followed.
* Enables multifactor authentication to the AWS account.
* Provides temporary access for users/devices and services where necessary.
  + Users can use their corporate network or Google/Facebook/other identify federations to access resources.
* Allows one to set up password rotation policies.
* Allows one to monitor resource requests in conjunction with CloudTrail service.

## Key Terminologies

* Users
  + End users such as people, employees of an organization, etc.
  + Users are global entities.
* Groups
  + A collection of users.
  + Each user in the group will inherit the permissions of the group.
* Policies
  + Policies are made up of documents called policy documents.
  + These documents are in the format of JSON and they give permissions to what a user/group/role is able to do.
* Roles
  + One can create a group of permissions and then assign them to specific users or AWS resources at a global-level.

## MFA

* MFA adds an extra level of security when accessing one’s AWS account.
* AWS MFA accepts both virtual as well as hardware MFA devices.
  + Virtual MFA devices include Google Authenticator/ Authy.
* MFA can be configured both through the IAM dashboard as well as through CLI.
* MFA can also be set up for individual users.
* IAM credentials report can be downloaded anytime to view information on all users, including root, for the particular AWS account, including whether or not they have MFA enabled.
  + Good for compliance purposes.

## Miscellaneous Features

* IAM is eventually consistent i.e. changes take some time to reflect.
* New IAM users always start with no permissions.
  + They have a password for console access.
  + They have an access key/secret key to access resources via API, CLI or the SDK.
  + They can be associated with up to 10 groups.
* IAM supports identity providers that are compatible with OpenID Connect (OIDC) or SAML 2.0 (Security Assertion Markup Language 2.0)
  + SAML requires the Identity Provider (IDP) and the Service Provider (SP) to know each other before hand, **pre-configured,**with **static** authentication and authorization.
  + OIDC doesn't have such a requirement.
* To test whether IAM policies are correctly configured, one can
  + Use IAM Policy Simulator to test if certain actions can be performed with given role.
  + Use *‘–dry-run’* command while running commands on CLI which will not perform the action, but simply tell one whether the action would’ve succeeded with current IAM role or not.
  + One can have variables inside the IAM policy using ‘$’ symbol.
    - For example, to grant access to an S3 bucket to users for their own folders only, one can specify the IAM resource as: *‘arn:aws:s3:::<bucket-name>/developer/${aws:username}/\*’*

# Simple Storage Service (S3)

## Definition

* It is a secure, durable, scalable, low-cost storage-service.
* Highly concurrent i.e. multiple points of access.
* Files are redundantly stored as objects, in containers called buckets (created at region-level), with a flat-naming structure. Buckets themselves have universal namespace.

## Storage Classes

Objects in the same bucket can have different storage classes.

* Standard
  + For general-purpose storage of frequently accessed data.
* S3 RRS
  + Reduced redundancy storage for frequently accessed, non-critical data. Redundancy, unlike other storage classes, is only 99.99%.
* Standard - IA
  + For long-lived but less frequently accessed data.
  + If moving data from Standard to Standard-IA, the following rules must be followed:
    - The data must be larger than 128 KB.
    - The data must have been stored in Standard class for at least 30 days.
  + Subcategory One-Zone IA: For data stored in only one AZ. Cheaper but less durable. 99.5% availability. Not resilient to loss of AZ.
* Glacier
  + For low-cost archival of cold data. If archived data is deleted within 90 days, early deletion fee is applied. Data retrieval (from fast to slow) can be of type:
    - Expedited (1-5 minutes)
      * Purchasing provisioned capacity ensures retrieval capacity for expedited retrievals is available when required.
    - Standard (3-5 hours)
    - Bulk (5-12 hours)
  + Data is stored as ‘Archives’ (max size 40 TB, with 32 KB overhead per archive). A group of ‘Archives’ is called ‘Vault’.
    - Each vault has:
      * One Vault Lock Policy
        + To restrict users/account permission.
      * One Vault Lock Policy
        + Immutable policy for compliance purposes.
        + Example: WORM policy – Write Once Read Many.
  + Archives cannot be uploaded to Glacier directly via the console.
    - Direct transfer must be done via AWS CLI or SDK.
* Intelligent Tiering
  + When access patterns are unknown, objects are put in ‘Standard’ storage class, but then automatically moved to IA/Glacier if they have not been accessed for 30+ days. They are moved back to higher tiers when accessed.

## Use Cases

* Store and distribute static web content and media.
  + Delivered directly from Amazon S3 because each object in Amazon S3 has a unique HTTP URL.
  + These URLs can be pre-signed to give short-term access.
* Host entire static websites.
  + Storage for static HTML files, images, videos, and client-side scripts in formats such as JavaScript.
* Data store for computation and large-scale analytics.
  + No constraints of single connection.
* Durable, scalable, and secure solution for backup and archiving of critical data.
  + Cross-region replication copies objects to different buckets asynchronously (Requires versioning to be enables).

## Performance

* Multipart Upload
  + Increases throughput, bandwidth utilization, and better recovery in case of network issues due to uploading objects in parts.
* Transfer Acceleration
  + Leverages Amazon CloudFront distribution network to transfer data over Amazon-optimized paths.
* Pairing with Database
  + Since S3 can’t be queried, pair with DB to store metadata.
* Partitioning data by adding a random prefix to the key names of objects can increase the PUT requests per second for an S3 bucket. This is useful when the number of PUT requests per second exceeds 100.

## Durability and Availability

* Automatically and synchronously storing data across multiple devices within region (>=3 AZ).
* Built in error correction. Verifies integrity of data using checksums. (Content-MD5 + CRC)
* No single point of failure.
* Designed for 99.999999999% durability and 99.99% availability.

## Scalability and Elasticity

* Unlimited number of files and bytes.
  + Each file can store up to 5 TB. One PUT request’s max size is 5 GB.
  + Scaling and distribution of data handled by AWS’s infrastructure.

## Security

* Grant permission to users to perform resource operations via access policy.
* Encryption:
  + Client-Side encryption: Client encrypts data at their end before uploading onto S3.
  + Server-Side encryption: AWS encrypts data at their end before uploading it onto S3 and decrypts it when it is being accessed.
    - SSE-S3: Objects encrypted with unique key, which is also encrypted by a master key, which keeps changing. Block cipher AES-256 used.
    - SSE-KMS: Separate permissions for envelope key. Audit trail provided for key usage.
    - SSE-C: Key is provided by client and AWS handles encryption. As the key is transferred over the network, only HTTPS endpoints can be used.
  + Data in transit protected by Secure Sockets Layer (SSL)/TSL or client-side encryption.
* Versioning can be enabled to preserve, retrieve and restore every version of object stored.
  + Multi-Factor Authentication (MFA) can be associated to buckets for versioning state change requests and to permanently delete an object’s version.
    - Only the root account owner can enable/disable MFA delete.
  + Every time an object is encrypted in a versioning enabled bucket, a new version is created. The unencrypted object remains as an old version.
  + Old hidden versions can cause seemingly-empty buckets to fail upon bucket deletion.
* Access logging can be enables to track all S3 activity associated to a bucket. Useful for security, access audits and monitor billing.
  + S3 access logs can be stored in another S3 bucket. (To avoid recursion)
  + API calls can be logged in AWS CloudTrail.
* Object lock can block object version deletion during a customer-defined retention period to enforce retention policies as an added security measure.
* When S3 receives a request, it must evaluate all the user policies, bucket policies and ACLs to determine whether to authorize or deny the request.
* Signed URLS can be used to provide access to resources for a limited amount of time.
* S3 CORS (Cross Origin Resource Sharing) allows one to limit the origins/websites from one domain to access objects from another domain.

## Cost Model

* Amount of data stored (by the GB) (~0.023$/month on Standard, ~0.125$ on IA, 0.004$ on Glacier)
* Retrieving data (Per 1000 requests + data returned)
* Lifecycle transition requests.
* Cross-region replication.
* Transfer acceleration.
* Storage management pricing (objects and tags)

## Miscellaneous Features

* S3 Select
  + Retrieve a smaller, targeted data set from an object using simple SQL statements.
  + Simplifies and improves performance of scanning and filtering the contents of an object by up to 400%.
* S3 Event Notifications
  + Enables one to run workflows, send alerts, or perform other actions in response to changes in objects stored on S3.
  + Possible actions include transcoding media upon upload, processing data files as they become available, and synchronizing objects with other data stores.
  + Possible targets for notifications include SQS, SNS and Lambda.
* S3 Object Tagging
  + Tags are key-value pairs which can be created (up to 10), updated or deleted at any time during lifetime of object.
  + Used for labelling, access control, selectively replicate, etc.
* S3 Inventory
  + Provide metadata of stored objects.
  + Can be used to audit and report on the replication and encryption status of objects.
* S3 Consistency Model
  + Read after Write consistency for PUTs of new objects.
  + Eventual consistency for overwrite PUTs and DELETEs (can take time to propagate).
* S3 Object Lifecycle Policies
  + The object lifecycle management allows for two types of behaviors:
    - Transition: Storage class of object changes
    - Expiration: When objects are permanently deleted.
  + Transition of objects to Glacier class is one-way.
    - From Glacier class, objects can be transitioned only to Deep-Archive class.
  + Transition of objects from any class to Standard/RRS class is not allowed.
  + Transition of objects from Standard or RRS class to Intelligent Tiering or Standard IA class requires the object to be present in the original class for a minimum of 30 days and the object size to be more than 128 KB.
* Cross-Region Replication
  + One can asynchronously replicate data from one S3 bucket to another bucket in another region, provided both the buckets have versioning enabled and proper IAM permissions are in place.

# Storage Gateway

## Definition

* Service that enables hybrid storage between on-premises environments and the AWS Cloud.
* Can be used via the management console and a gateway that is available as a VM, or a physical hardware appliance.

## Benefits of Storage Gateway

* Enables effective leveraging of AWS storage with existing on-premise applications and workflows by using a standard set of protocols such as NFS, ISCCI, etc.
* Through its local cache, provides low-latency access to recently used datasets.
* The gateway optimizes data transfer through intelligent buffering, upload management to address network variations, and bandwidth management.
* Provides an effective mechanism to store data on S3 across a range of storage services most suitable to use case (hybrid cloud backup).
* Frees up space of on-premise storage infrastructure.
* Integrates natively with AWS CloudWatch, AWS KMS, AWS IAM, etc.

## Storage Solutions

* File Gateway
  + Supports file interface into S3 with existing applications/workflows.
  + Can store and retrieve objects in S3 using industry standard file protocols such as Network File System (NFS) and Server Message Block (SMB).
  + Manages data transfer to and from AWS, buffer applications from network congestion, optimize and stream data in parallel and manage bandwidth consumption.
  + As a further optimization, only file metadata is stored by file share. The actual data is only loaded when the NFS client specifically requests it.
  + R/O operation performance optimizations include
    - Read: Gateway first checks for cache before requesting data from S3.
    - Write: Gateway first commits on cache and sends acknowledgment of write success to NFS client, enabling low latency on writes. The file is then asynchronously put into S3 to increase local performance of internet transfers.
    - Update: Gateway transfers only newly written bytes over the network. S3 creates new object with previous version of object and new bytes.
* Volume Gateway
  + Provides cloud-backed storage volumes that one can mount as Internet Small Computer System Interface (ISCSI) devices from on-premises application servers.
  + Gateway supports two volume configurations:
    - Cached Volumes:
      * Most of the data is stored on S3 but a copy of frequently accessed datasets is retained locally.
      * Minimizes the need for on-premise storage while retaining low-latency for frequently accessed data.
    - Stored Volumes
      * Entire data is stored locally while asynchronously backing up point-in-time EBS snapshots to S3, enabling one to make space-efficient versioned copies of data volumes for protection, recovery, migration, etc.
      * Provides durable and inexpensive offsite backups.
* Tape Gateway
  + Eliminates the operational burden of provisioning, scaling and maintaining a physical tape infrastructure as the gateway provides a virtual tape infrastructure (via it’s Virtual Tape Library i.e. VTL interface) that scales seamlessly with business requirements.
  + Used for cost-effective yet durable archive backup of data into Glacier.
  + VTL interface is compatible with backup and archival applications that use industry standard ISCSI-based tape library interface.

## Security

* All data transferred between the gateway and AWS storage is encrypted with SSL.
* All data stored in S3 by gateway is by default encrypted with SSE-S3. SSE-KMS can be configured.

## Cost Model

* Volume storage is not pre-provisioned. Billing is applied on amount of stored data on volume, not the size of the volume.
* Data at rest and transit is compressed, reducing charges. Gateway only uploads changed data.
* For all gateway types, 0.01$ per GB of data written to AWS, among other charges.

## Miscellaneous features

* Caches Storage
  + As applications write data to storage volumes in AWS, the gateway first stores the data on on-premise disks which forms the cache storage.
  + It acts as a durable store for data waiting to be uploaded to S3 from the upload buffer.
  + Provides low-latency access for recently accessed data as gateway checks in the caches storage before checking S3.
  + The cache storage is recommended to be about 20% of total storage, and at least larger than the upload buffer.
  + The storage removes data based on Least-Recently-Used (LRU) algorithm.
* Upload Buffer
  + To prepare for upload to S3, gateway stores incoming data in a staging area, i.e. the upload buffer.
* Storage Gateway Limits
  + Each volume gateway can store up to 32 volumes.
    - In caches mode, each volume can store up to 32 TB, making the gateway capacity 1 PB.
    - In stored volumes, each volume can store up to 16 TB, making the gateway capacity 512 TB.
  + Each tape gateway can have up to 1500 tapes. Size of each tape can be of 100, 200, 400, 800, 1500 and 2500 GB.
* Snapshots
  + Only changed data is captured with incremental snapshots.
  + Snapshots can be used as a starting point for new EBS volumes, versioning and backup.

# Snowball

## Definition

* Service that transfers large amounts of data into and out of AWS Cloud using physical storage devices, bypassing the internet.
* Also known as AWS Import/Export service.

## Use Cases

* Securely and quickly transfer TBs/PBs of data to AWS.
* When one doesn’t want to make expensive upgrade of network infrastructure.
* When one frequently experiences large backlogs of data.
* When one is in a physically isolated environment.
* When high-bandwidth connections internet connections are not available or cost-prohibitive.

## Important Components

* Snowball Client: Software tool used to transfer data to-and-fro between on-premises and device, depending on if it is an import job (to AWS) or export job (from AWS).
* Job Manifest File: An encrypted metadata file that is used to uniquely identify the data transfer job.
* Job Manifest Unlock Code: A 25-character code to unlock manifest file.

## Performance

* Devices have 10 Gpbs network interfaces.
* Under normal circumstances, data transfer in a day reaches up to 48TB. This can be increased by using multiple snowball devices in parallel.
* The Snowball Client improves the transfer performance for small files by batching them into larger *.snowballarchives* files automatically.
* When the compressed batches are imported into Amazon S3, they are automatically expanded, so there’s no need for one to expand the files themselves.

## Security

* All data is encrypted with 256-bit encryption.
* Industry-standard Trusted Platform Module (TPM) is used with a dedicated processor designed to detect any unauthorized modifications to software, firmware or hardware.

## Cost Model

* Flat-free for device handling and import/export operations at AWS data centers. (~250$)
* Snowball is free for 10 days at client site. Day of receiving from AWS and day of shipping back to AWS are not counted. Every extra day costs 15$.
* Data being transferred into AWS storage services is free. Data being transferred out has normal fees.

## Miscellaneous Features

* Snowball devices come in two variants: 80TB and 50TB (USA only)
* Data transfers must be completed in 90 days.
* Data transfers must be within the same region.
* Data stored in Glacier must be upgraded to S3 before exporting it.
* Snowmobile
  + Used when massive data transfer (exabyte-level) is required.
  + Each snowmobile can store up to 100PB of data.

# Snowball Edge

## Definition

* Snowball Edge is a data transfer device with on-board storage and compute power that provides select AWS services, such as Lambda and EC2-based applications.
* Snowball Edge comes in two options, Storage Optimized and Compute Optimized, to support local data processing and collection in disconnected environments such as ships, wind mills, and remote factories.

## Use Cases

* AWS Snowball Edge Storage Optimized
  + For use cases that require secure and quick transfer of terabytes to petabytes of data to AWS.
  + It is also a good fit for running general purpose analysis such as IoT data aggregation and transformation.
  + It provides up to 100TB of storage, 24 vCPUs, 1 TB SSD, and up to 40Gb network connectivity to address large scale data transfer and pre-processing use cases.
  + Multiple Snowball Edge devices can be clustered together.
* AWS Snowball Edge Compute Optimized
  + For use cases that require access to powerful compute and high-speed storage for data process before transferring it into AWS.
  + It is good fit for applications such as high-resolution video processing, advanced IoT data analytics, and real-time optimization of machine learning models in environments with limited connectivity.
  + It features 52 vCPUs, 7.68 TB of NVMe SSD, and up to 100Gb networking to run applications.

# Elastic Cloud Compute (EC2)

## Definition

* Service that provides resizable compute capacity in the cloud, via virtual computing environments known as instances.
* Provides complete control of computing resources in terms of instance types and scalability.

## Instance Types

* General Purpose
  + A1
    - Provides significant cost savings since AWS’s own ARM-based processors are used.
    - Use cases include scale-out workloads such as web servers, containerized microservices, caching fleets, distributed data stores as well as development environments.
    - Also suited for ARM-based workloads that are supported by extensive ARM ecosystems.
  + T3
    - Provides baseline CPU performance with the ability to burst CPU usage at any time indefinitely.
    - Offers a balance of compute, memory and network resources designed for applications with moderate CPU usage that experience temporary spike.
    - CPU credits are generated when workloads are below the baseline which can be later utilized for CPU bursts.
    - Use cases include micro-services, low-latency interactive applications, small and medium databases, virtual desktops, development environments, code repositories, and business-critical applications.
  + M5
    - Latest general-purpose instance type.
    - Provides a balance of compute, memory, and network resources for applications which have a balanced workload.
    - Small and mid-size databases, data processing tasks that require additional memory, caching fleets, and for running backend servers for SAP, Microsoft SharePoint, cluster computing, and other enterprise applications.
* Compute Optimized
  + C5
    - Optimized for compute-intensive workloads and deliver cost-effective high performance at lower price per compute ratio.
    - Use cases include high performance web servers, scientific modelling, batch processing, distributed analytics, high-performance computing (HPC), machine/deep learning inference, ad serving, highly scalable multiplayer gaming, and video encoding.
* Accelerated computing
  + P3
    - They are the latest generation of general-purpose GPU instances.
    - Use cases include machine/deep learning, high performance computing, computational fluid dynamics, computational finance, seismic analysis, speech recognition, autonomous vehicles, drug discovery.
  + G3
    - Optimized for graphic-intensive applications.
    - Use cases include 3D visualizations, graphics-intensive remote workstation, 3D rendering, application streaming, video encoding, and other server-side graphics workloads.
  + F1
    - Offers customizable hardware acceleration with field programmable gate arrays (FPGAs).
    - Use cases include genomics research, financial analytics, real-time video processing, big data search and analysis, and security.
* Memory Optimized
  + R5
    - Use cases include memory intensive applications such as high-performance databases, distributed web scale in-memory caches, mid-size in-memory databases, real time big data analytics, and other enterprise applications.
  + X1
    - Use cases include In-memory databases (e.g. SAP HANA), big data processing engines (e.g. Apache Spark or Presto), high performance computing (HPC).
* Storage Optimized
  + H1
    - Feature up to 16 TB of HDD-based local storage, deliver high disk throughput, and a balance of compute and memory.
    - Use cases include MapReduce-based workloads, distributed file systems such as HDFS and MapR-FS, network file systems, log or data processing applications such as Apache Kafka, and big data workload clusters.
  + I3
    - Provides Non-Volatile Memory Express (NVMe) SSD-backed instance storage optimized for low latency, very high random I/O performance, high sequential read throughput and provide high IOPS at a low cost.
    - Use cases include NoSQL databases (e.g. Cassandra, MongoDB, Redis), in-memory databases (e.g. Aerospike), scale-out transactional databases, data warehousing, Elasticsearch, analytics workloads.
  + D2
    - Feature up to 48 TB of HDD-based local storage, deliver high disk throughput, and offer the lowest price per disk throughput performance on Amazon EC2.
    - Use cases include Massively Parallel Processing (MPP) data warehousing, MapReduce and Hadoop distributed computing, distributed file systems, network file systems, log or data-processing applications.

## Purchasing Options

* On-Demand Instances
  + Enables one to reserve capacity for your Amazon EC2 instances in a specific Availability Zone for any duration.
* Reserved Instances
  + Purchase, at a significant discount, instances that are always available, for a term from one to three years, in a specific region.
  + The discount also depends on whether the payment is fully upfront, partially upfront, or if there is no upfront, in decreasing order of discount.
  + One can choose between two classes:
    - Standard
      * Applies to a single instance family, platform, scope, and tenancy over a term.
      * Can be sold in the Reserved Instance Marketplace.
    - Convertible
      * Can be exchanged during the term for another Convertible Reserved Instance with new attributes including instance family, instance type, platform, scope, or tenancy.
      * Cannot be sold in the Reserved Instance Marketplace.
  + Instance size flexibility is determined by the normalization factor of the instance size. A bigger instance reservation discount can apply for multiple smaller instances. Similarly, a smaller instance can apply partial discount for a bigger instance.
* Scheduled Instances
  + Enables one to purchase capacity reservations that recur on a daily, weekly, or monthly basis, with a specified start time and duration, for a one-year term.
* Spot Instances
  + Request unused EC2 instances, which can lower costs significantly.
  + Bid a price lower that On-Demand Price. If there is capacity available and the current spot price is lower than bid price, EC2 instance is provided.
  + If the spot price exceeds bid price, the instance is terminated.
    - If the instance is terminated by AWS within an hour of launch, no charge is billed.
  + Spot Instances are a cost-effective choice if you can be flexible about when your applications run and if your applications can be interrupted.
* Dedicated Instances
  + Instances that run in a virtual private cloud (VPC) on hardware that's dedicated to a single customer.
  + May share hardware with other instances in same account.
  + No control over instance placement.
* Dedicated Hosts
  + A physical server with EC2 instance capacity fully dedicated to one’s use.
  + Allows one to use existing per-socket, per-core, or per-VM software licenses, including Windows Server, Microsoft SQL Server, SUSE, Linux Enterprise Server, and so on.

## Amazon Machine Image (AMI)

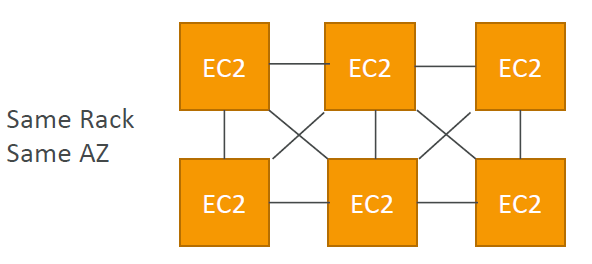
* An Amazon Machine Image (AMI) provides the information required to launch an instance. You must specify an AMI when you launch an instance.
* AMIs are region-specific. To use an AMI is another region, it must be copied to that region.
* AMI can be created from snapshots or directly from the EBS volumes.
* An AMI includes the following:
  + A template for the root volume for the instance (for example, an operating system, an architecture, an application server, and applications).
  + Launch permissions that control which AWS accounts can use the AMI to launch instances.
  + A block device mapping that specifies the volumes to attach to the instance when it's launched. The volumes can either be Instance Store Volumes or EBS volumes.
* There are two types of AMI virtualization types:
  + Hardware Virtual Machine (HVM)
    - HVM guests are fully virtualized i.e. VMs running on top of their hypervisors are not aware that they are sharing processing time with other clients on the same hardware.
    - HVMs can use hardware extensions, such as enhanced networking or GPU processing which provide very fast access to underlying hardware on the host system.
    - PV drivers are available for HVM.
  + Paravirtual (PV)
    - Lighter form of virtualization. The Guest OS is modified to interact directly with hypervisor.
    - Paravirtual guests traditionally performed better with storage and network operations than HVM guests because they could leverage special drivers for I/O that avoided the overhead of emulating network and disk hardware, whereas HVM guests had to translate these instructions to emulated hardware.
    - One major disadvantage with Paravirtualization is that one needs a region-specific kernel object for each Linux instance.
* When copying an AMI to another account:
  + Sharing the AMI does not change the owner.
  + Copying the shared AMI makes one the owner of the AMI.
  + The original owner of the AMI should also share permissions to access the storage used to back the AMI, either the associated EBS snapshot or S3 bucket.
  + One cannot directly copy an AMI shared from another account if it is encrypted, rather it needs to be decrypted and re-encrypted with a new key and registered as a new AMI.
  + If the AMI is associated with a ‘*billingProduct’* code, it cannot be copied directly. An instance needs to be started with that AMI, and then a new AMI must be created from that instance.

## Instance Store Volumes

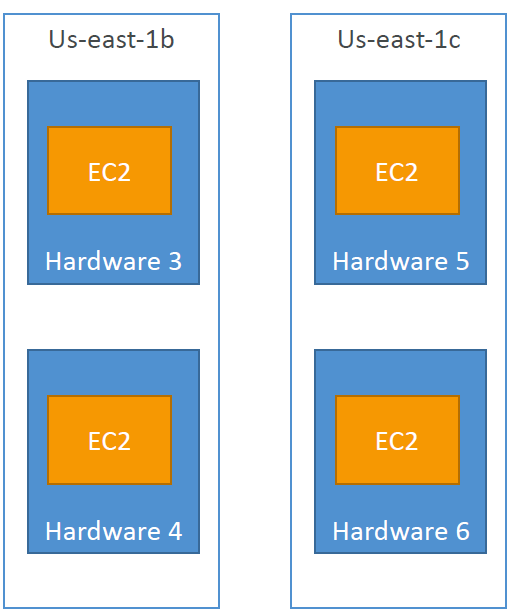
* They are also known as Ephemeral storage, as the data is not persisted if the instance is stopped or terminated, unless a snapshot is taken. However, if the instance is rebooted, data persists.
* If the hypervisor fails, the EC2 can be stopped and started again to change hypervisor. While the EBS volumes retain data, the data on instance store volumes is lost.
* Although the instance store volumes are scalable on-the-fly like EBS volumes, additional instance store volumes cannot be attached to an EC2 instance once it has launched.
* Since instore store volumes are hard drives that are physically attached to the EC2 hosts running the instance, they provide much higher IOPS (100,000 IOPS for some instance types).
* Generally used for data that needs to be accessed quickly but temporarily like swap or paging files, or for data that is regularly replicated at other locations.

## Placement Groups

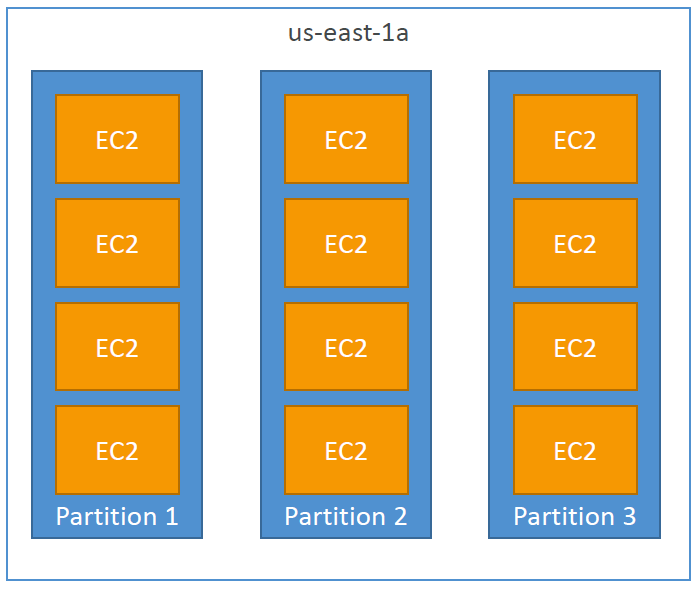
* There are two kinds of placement groups:
  + Clustered Placement Group
    - A grouping of instances within a single AZ, usually within a single rack. Instances cannot span multiple AZ.
    - Only certain instances can be placed into a such a group.
    - Recommended for applications that need low network latency, high network throughput or both.



* + Spread Placement Group
    - A grouping of instances that are placed on distinct underlying hardware. Instances can span multiple AZ.
    - A maximum of 7 instances per AZ per group.
    - Recommended for applications that have a small number of critical instances that should be kept away from each other, reducing the risk of simultaneous failure.



* + Partition Placement Group
    - Spread across up to 7 different partitions (which rely on different sets of racks) within an AZ.
    - Scales up to 100s of EC2 instances per group.



* The name specified for a placement group must be unique within one’s AWS account.
* AWS recommends homogeneous instances within a placement group.
* Placement groups cannot be merged.
* An existing instance cannot be brought into a placement group. Instead, an AMI can be created from the instance and a new instance can be launched into the placement group with that AMI, although it might be difficult to add more instances to an existing placement group to due insufficient capacity.

## Security Groups

* It is a virtual firewall for the instances defined at region level.
* Security groups are associated with network interfaces. Changing an instance’s security groups changes the security groups associated with the primary network interface (eth0).
* An instance can have multiple security groups, and a security group can be attached to multiple instances.
* All inbound rules are copied as outbound rules i.e. they are stateful.
* Any changes in the rules for security groups are instantly propagated.
* One cannot deny traffic using SG.
* Inbound Ports are disabled or blocked by default. Outbound ports are open by default.

## AutoScaling Groups (ASG)

* One can create groups where EC2 instance fleets can be launched based on configured templates and scale it out based on performance thresholds or pre-defined schedules.
  + In addition to normal metrics, custom metrics can be defined to determine if scaling needs to occur.
* ASGs have the following attributes:
  + A launch configuration
    - AMI + Instance Type
    - EC2 User Data
    - EBS Volumes
    - Security Groups
    - SSH Key Pair
  + The minimum/ maximum/ initial capacity.
  + Network + Subnet information.
  + Load balancer information.
  + Scaling Policies (Could be average CPU, Network in/out, based on a schedule, etc).
* The default health check for an autoscaling group are EC2 status checks. When these checks fail, the autoscaling group replaces the instance.
  + ASGs can also be configured to use an ELB’s (if present) health checks in addition to EC2 status checks to determine if an instance should be replaced or not.
* The cooldown period helps to ensure that the ASG doesn't launch or terminate additional instances before the previous scaling activity takes effect.
  + This is useful when it takes time for newly scaled instances to start accepting requests due to say, a software installation, and CloudWatch keeps detecting that more instances are required as the resource utilization is still above the threshold.
* When an ASG needs to be scaled in, the protocol to select which EC2 gets terminated is as follows:
  + Choose AZ with most instances. If there is an unprotected instance (allowed to scale-in), then terminate it. If multiple, choose the one with oldest launch configuration.
  + If multiple such instances match above criteria, terminate the instance closest to the next billing hour. If there are multiple such instances, pick one at random.
* One can add lifecycle hooks to an ASG to perform custom actions when instances launch or terminate.
* ASGs automatically register new instances to load balancers if present.
* One can suspend certain processes in ASGs such as ‘*Launch*’, ‘*Terminate*’, ‘*HealthCheck*’, ‘*AZRebalance*’, etc.
  + Suspending ‘*Launch*’ and/or ‘*Terminate*’ will impact AZ rebalancing. (ASG can grow 10% more than its desired size but will not terminate old instances).
* If the ASG fails to launch an instance for over 24 hours, it will automatically suspend the process (administrative suspension).

## Troubleshooting

* During EC2 launch:
  + If one exceeds the EC2 limit for the region they are launching instances in, they receive ‘*InstanceLimitExceeded*’ error. One must contact AWS support to have the limit increased.
  + If one receives the ‘*InsufficientInstanceCapacity*’ error, it means that AWS does not have that much on-demand capacity for the particular AZ in which the instance is being launched. One must request fewer instances, wait for some time, or use a different instance type to launch instances in that AZ.
  + If during launch, an instance goes from pending to terminated state, it can be due to one of the following reasons (The exact reason can be found by using the description tab):
    - The EBS volume limit has been reached.
    - The EBS snapshot is corrupt.
    - The EBS volume is encrypted and the necessary permissions to access the KMS key for decryption are missing.
    - The instance store-backed AMI being used to launch the instance is missing a part.
* During SSH:
  + The private key (.pem file) on the UNIX machine might not be having 400 permissions, resulting in *‘Unprotected Private Key File error*’.
  + The username for the OS might not be correct, or else one might get *‘Host key not found’* error.
  + If the connection times out, then:
    - Security groups might not be configured properly for SSH.
    - The CPU load on the instance might be too high to process connection request.

## Miscellaneous Features

* While on EC2 terminal, if one wishes to view the bootstrap script used while booting the instance, or if one wishes to view the instance metadata, they can use curl command.
  + curl http://169.254.169.254/latest/meta-data/
  + curl http://169.254.169.254/latest/user-data/
* Enhanced networking provides higher bandwidth, higher packet per second (PPS) performance, lower latency, consistency, scalability and lower jitter.
  + Instances must be in a VPC.
  + Instances must be backed by an HVM virtualization type AMI.
* One can configure instances to stop or terminate upon shutdown from the OS.
* By default, one can launch only 20 EC2 instances per region for new accounts (Soft limit).
* Using Elastic IP, one can mask the failure of underlying instance or software by rapidly remapping the address to another instance.

# Elastic Block Store (EBS)

## Definition

* Storage service that provides provide durable block-level storage for use with EC2 instances.
* They are network-attached storage that persists independently from the running life of a single EC2 instance.
* The EBS volume and the EC2 instance it is attached to must be in the same AZ to reduce latency.
* Multiple EBS volumes can be attached to a single EC2 instance. However, a single EBS volume cannot be attached to more than one instance.
* EBS also provides the ability to create point-in-time snapshots of volumes, which are stored in Amazon S3. These snapshots can be used as the starting point for new EBS volumes and to protect data for long-term durability.
* Size of a single volume can range from 1 GB to 16 TB and are allocated in 1 GB increments. They can be changed on-the-fly except for standard i.e. magnetic volumes.

## Snapshot Logic

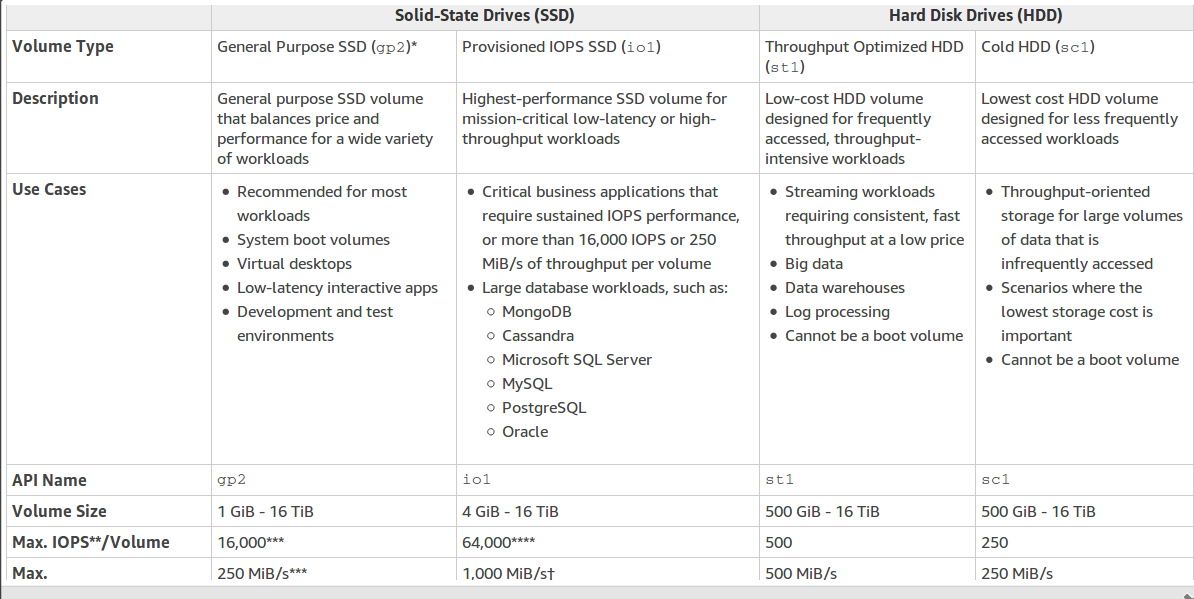
* EBS volumes are backed up to Amazon S3 by taking point-in-time snapshots.
* Snapshots are incremental backups, which means that only the blocks on the device that have changed after the most recent snapshot are saved, minimizing the time to create the snapshot and reducing costs by not duplicating data.
* As EBS-volumes are AZ-specific, to move them from one AZ to another AZ/Region, a snapshot can be created and moved to target location and the EBS volume can be created thereafter.
* Volumes created from encrypted snapshots and encrypted by default. Similarly, snapshots created from encrypted volumes are encrypted by default.
* Only unencrypted snapshots can be made public. Encrypted snapshots can be shared with certain configured AWS accounts.
* To delete a snapshot of the root device of an EBS volume used by a registered AMI, one must first deregister the AMI.
* One can use the snapshot lifecycle manager feature to automate backups of EBS volumes by taking snapshots at configured time frames.

## Usage Patterns

* Meant for data that changes relatively frequently and needs to persist beyond the life of EC2 instance.
* Well-suited for use as the primary storage for a database or file system, or for any application or instance (operating system) that requires direct access to raw block-level storage.
* Provides a range of options that allow you to optimize storage performance and cost for your workload.

## Performance

* Provides a range of volume types that are divided into two major categories
  + SSD-backed storage volumes
    - Offers great price/performance characteristics for random small block workloads, such as transactional applications.
  + HDD-backed storage volumes
    - Offers the best price/performance characteristics for large block sequential workloads.
* It is possible to attach and stripe data across multiple volumes of any type to increase the I/O performance available to your Amazon EC2 applications.
* Newly created EBS volumes receive their maximum performance the moment they are available, and they don’t require initialization.
* There is a significant increase in latency on the first access of each block of data on a new EBS volume that was restored from a snapshot. One can avoid this performance hit by accessing each block prior to putting the volume into production (warming).
* Best price/performance balanced workloads take advantage of different volume types on a single EC2 instance.
* One can stripe data across multiple similarly provisioned EBS volumes using RAID 0 (disk striping) or logical volume manager software, thus aggregating available IOPS, total volume throughput, and total volume size.
* Instances can be EBS-optimized to increase network traffic between them and the EBS volumes.



## Durability and Availability

* Designed to be highly available and reliable.
* EBS volume data is replicated across multiple servers in a single Availability Zone to prevent the loss of data from the failure of any single component.
* Creating snapshots of EBS volumes frequently can improve the durability of the data. These snapshots have 99.99% availability.
* The snapshots are at region-level. They can be used to re-create a volume in any AZ of that region or copied to other regions/shared between users.
* One can use RAID 1 configuration for additional data redundancy. Essentially, every write will be written to two EBS volumes (masked by one logic volume).
  + While this increases redundancy in case of one of underlying volume fails, as every write is done twice, network bandwidth is consumed twice as much.

## Scalability and Elasticity

* One can easily and rapidly provision and mount EBS volumes to scale in and out with increasing storage demands.
* EBS volumes can be modified on-the-fly including volume type upgrade, volume size increase/decrease and IOPS adjustment.
  + The volume size can only be increased, not decreased.
    - After increasing the size, the volume will be in the optimization phase. It will still be usable, but with lower-than-normal performance.
    - One would need to repartition the volume post-increase to use the added storage.
  + The IOPS can only be increased for io1 volume types.

## Security

* IAM enables access control for your EBS volumes, allowing one to specify who can access which EBS volumes.
* EBS encryption enables data-at-rest and data-in-motion security. It offers seamless encryption of both EBS boot volumes and data volumes as well as snapshots, eliminating the need to build and manage a secure key management infrastructure.
  + AWS KMS (AE-256) is used for encryption.
* To encrypt an EBS volume:
  + Create a snapshot of from the volume.
    - If the volume is currently attached to an instance, then stop the instance.
  + Make a copy of the snapshot in the same region and encrypt it as the copy takes place.
  + Finally, create the new, encrypted volume from the encrypted snapshot.

## Cost Model

* Amazon EBS pricing has three components:
  + Provisioned storage (regardless if it is used or not)
  + I/O requests
  + Snapshot storage.
* There is no charge for transferring information among the various AWS storage offerings as long as it is not a cross-region transfer.

## Miscellaneous Features

* The root volume of a default AMI provided by AWS cannot be encrypted.
* When an instance is terminated, additional volumes aside from root volume are not deleted by default.
* The maximum ratio of provisioned IOPS to requested volume size in GB is:
  + Volume type gp2: 3:1, up to ~5.33TB+ & 16000 IOPS.
    - We can also conclude that increasing volume beyond 5.33TB will not increase IOPS.
  + Volume type io1: 50:1, up to 640 GB+ & 32000 IOPS. (64000 for Nitro instances)
* The maximum ratio of throughput to request volume size per TB is:
  + Volume type st1: 40 MB/s, up to 500 MB/s.
  + Volume type sc1: 12 MB/s, up to 250 MB/s.
* When creating snapshots of EBS volumes that are configured in a RAID array, it is critical that there is no data I/O to or from the volumes when the snapshots are created. RAID arrays introduce data interdependencies and a level of complexity not present in a single EBS volume configuration.
* To detach an EBS volume which is not the root volume while the instance is running, one must first unmount the volume. If the volume is the root volume, then the instance must be stopped before it can be detached.
* Throughput in MB/s = (Volume size in GB) \* (IOPS per GB) \* (I/O size in KiB)
  + The maximum I/O size in EBS is 256 KB.
  + Example: For, for 100 GB gp2 EBS volume, we get: 100\*3\*256 = 75 MB/s

# Elastic File System (EFS)

## Definition

* It is a managed storage service that delivers a simple, scalable, elastic, highly available, and highly durable network file system as a service to EC2 instances.
* Can create file systems that are accessible to Amazon EC2 instances via a file system interface using standard operating system file I/O APIs and supports full file system access semantics, such as strong consistency and file locking.
* Can an automatically scale from gigabytes to petabytes of data without needing to provision storage.
* Tens, hundreds, or even thousands of EC2 instances can access an Amazon EFS file system at the same time, and Amazon EFS provides consistent performance to each instance.
* No minimum fee or setup cost—One simply pays for what they use.

## Use Cases

* Designed to meet the needs of multi-threaded applications and applications that concurrently access data from multiple EC2 instances and that require substantial levels of aggregate throughput and input/output operations per second (IOPS).
* Its distributed design enables high levels of availability, durability, and scalability, which results in a small latency overhead for each file operation.
* Because of this per-operation overhead, overall throughput generally increases as the average input/output (I/O) size increases since the overhead is amortized over a larger amount of data. This makes Amazon EFS ideal for growing datasets consisting of larger files that need both high performance and multi-client access.
* Supports highly parallelized workloads and is designed to meet the performance needs of big data and analytics, media processing, content management, web serving, and home directories.

## Performance

* Amazon EFS file systems are distributed across an unconstrained number of storage servers, enabling file systems to grow elastically to petabyte-scale and allowing massively parallel access from EC2 instances within a region. This distributed data storage design means that multi-threaded applications and applications that concurrently access data from multiple EC2 instances can drive substantial levels of aggregate throughput and IOPS.
* There are two different performance modes available for Amazon EFS:
  + General Purpose
    - General Purpose performance mode is the default mode and is appropriate for most file systems which are latency-sensitive like web serving environments, content management systems, home directories, etc.
  + Max I/O.
    - Max I/O performance mode is optimized for applications where tens, hundreds, or thousands of EC2 instances are accessing the file system.
    - With this mode, file systems scale to higher levels of aggregate throughput and operations per second with a tradeoff of slightly higher latencies for file operations.
    - Suitable for highly parallelized applications and workloads, such as big data analysis, media processing, etc.
* Due to the spiky nature of file-based workloads, Amazon EFS is optimized to burst at high-throughput levels for short periods of time, while delivering low levels of throughput the rest of the time.
  + A credit system determines when an Amazon EFS file system can burst.
  + A file system can drive throughput continuously at its baseline rate. It accumulates credits during periods of inactivity or when throughput is below its baseline rate.
* If the application can handle asynchronous writes to the file system, and one is able to trade off consistency for speed, enabling asynchronous writes may improve performance.

## Durability and Availability

* Amazon EFS is designed to be highly durable and highly available.
* Each Amazon EFS file system object (such as a directory, file, or link) is redundantly stored across multiple Availability Zones within a Region.

## Scalability and Elasticity

* Amazon EFS automatically scales one’s file system storage capacity up or down as one adds or removes files without disrupting one’s applications, giving one just the storage required, when it’s required, and while eliminating time-consuming administration tasks associated with traditional storage management (such as planning, buying, provisioning, and monitoring).
* EFS file system can grow from an empty file system to multiple petabytes automatically, and there is no provisioning, allocating, or administration.

## Security

There are three levels of access control to consider when planning your EFS file system security

* IAM permissions for API calls.
  + Identity-based policies, like IAM policies, are used to assign permissions to IAM identities to manage the EFS resources and sub resources.
* Security groups for EC2 instances and mount targets.
  + Groups play a critical role in establishing network connectivity between EC2 instances and EFS file systems.
  + One associates one security group with an EC2 instance and another security group with an EFS mount target associated with the file system. These security groups act as firewalls and enforce rules that define the traffic flow between EC2 instances and EFS file systems.
  + One must add inbound rules in the EFS Security group to allow traffic from EC2’s Security Group.
* File System-level users, groups, and permissions.
  + EFS file system objects work in a Unix-style mode, which defines permissions needed to perform actions on objects.
  + Users and groups are mapped to numeric identifiers, which are mapped to EFS users to represent file ownership.
  + Files and directories within Amazon EFS are owned by a single owner and a single group. Amazon EFS uses these numeric IDs to check permissions when a user attempts to access a file system object.
* One can also encrypt the data in EFS using KMS.

## Cost Model

* One only pays for the amount of storage put into the file system.
* As files are added & the EFS file system dynamically grows, one only pays for the amount of storage used.
* As files are removed & EFS file system dynamically shrinks, one stops paying for the data deleted.
* There are no charges for bandwidth or requests, and there are no minimum commitments or up-front fees.

## Miscellaneous Features

* By default, EFS has two kinds of throughput mode
  + Bursting throughput mode
    - The default throughput mode, it scales as the file system grow.
  + Provisioned throughput mode
    - The required throughput for the file system is provisioned independent of the amount of data stored.
* AWS DataSync is an online data transfer service that makes it faster and simpler to move data between on-premises storage and Amazon EFS.

# Relational Database Service (RDS)

## Definition

* Managed service that makes it easy to set up, operate, and scale a relational database in the cloud.
* It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching, maintenance, backups and security, freeing one to focus on applications and business requirements.
* It supports the following SQL engines:
  + Amazon Aurora
  + PostgreSQL
  + MySQL
  + MariaDB
  + Oracle Database
  + Microsoft SQL Server

## DB Instances

* A DB instance is an isolated database environment running in the cloud. It is the basic building block of Amazon RDS.
* They are essentially the database environment in the cloud with the compute and storage resources specified.
* A single user can have up to 40 DB instances.
* There are primary three types of DB instances in terms of specs:
  + M5: General purpose
  + X1e/X1/R5: Memory optimized
  + T3: Burstable performance
* There are two purchasing options for DB instances:
  + On-Demand Instances
  + Reserved Instances
* There are three storage types for DB instances:
  + General-Purpose (gp2): SSD-backed, general purpose with burstable performance.
  + Provisioned IOPS (io1): SSD-backed, for high performance and consistency.
  + Magnetic: For support for legacy databases.
* If one wants their DB Instance to run with custom-specified engine configuration values, they can simply create a new DB Parameter Group.

## Performance

* There are two ways though which RDS is optimized
  + Elastic Cache
    - One can use Elastic Cache to cache frequently accessed information/frequent identical queries results in-memory.
    - Two types:
      * Memcached
      * Redis
  + Read Replication
    - Read replicas are essentially read-only copies of the primary DB.
    - Writes are done to primary DB which is propagated to read replicas asynchronously.
      * Writes are not allowed on read-replicas themselves except for MySQL and MariaDB.
    - Read replicas can have read replicas (this can create read latency issues). These read replicas can then be used to serve read-requests, sharing the load from the primary DB.
    - Each read replica will have its own DNS endpoint.
    - The DB must have automatic backups turned on to use read replicas.
    - Read replicas can be in another region.
    - Read replicas can be promoted to their own databases. This breaks the replication.
    - Read replicas can be Multi-AZ enabled.
    - Useful for read-intensive workloads and scaling, not for data recovery.
    - One can have Aurora or MySQL read replicas.
* One can also enable ‘RDS Performance Insights’ on their DB instances to get a visual view of how the database resources are being utilized and to identify potential bottlenecks.
  + By Waits: Find the resource which is the bottleneck (CPU, IO, Lock, etc.)
  + By SQL Statements: Find the SQL statements which take a lot of resource.
  + By Hosts: Find the server that is using most of the DB.
  + By User: Find the user that is using most of the DB.
  + DBLoad: Find the number of active sessions for the DB engine.

## Backup & Recovery

* There are two backup methodologies supported by RDS:
  + Automated Backups:
    - Recover database to any point in time within a configurable retention period (1 ~ 35 days). Automated backup will take a full daily snapshot and will also store transaction logs throughout the day.
    - During recovery, latest snapshot is taken, and transactions logs relevant to that day are applied. This allows point in time recovery down to a second within the retention period.
    - They are enabled by default. The backup data is stored in S3 for free. E.g. 10GB DB instance storage gives free 10GB worth of storage on S3.
    - Backups are taken within a defined window period, during which storage I/O might be suspended and elevated latency might be experienced.
  + Manual Database Snapshots:
    - Backups are done manually i.e. they are user initiated.
      * Snapshots are stored even after RDS is deleted, unlike automated backups.
    - Useful for checkpointing database before making big changes.
* In case of either automatic or manual snapshots, one can:
  + Restore the database from the snapshot
  + Copy the snapshot to another region/AZ (While possibly encrypting it)
  + Migrate the snapshot to another DB engine.
* Restored version of database will always be on new instance with new DNS, regardless of backup methodology.
* Multi-AZ deployments
  + For disaster recovery purposes, configuring Multi-AZ for a database creates an exact copy of the database on stand-by in another AZ synchronously.
  + AWS automatically changes RD’s DNS to stand by database if primary database fails i.e. failover is automated so that operations can resume without administrative intervention.
  + Stand-by databases will always be in different AZ of same region as primary, as inactive until failover.
  + DB snapshots can be taken from the stand-by to avoid I/O suspension in the source DB instance.

## Security

* Encryption at rest is supported for all engines.
  + Encryption is done using AWS KMS.
* Once the DB instance is encrypted, the data store at rest in the underlying storage is encrypted, as are its automated backups, read replicas, and snapshots.

## Cost Model

* DB Instance hours based on instance class. Partial hours are fully billed.
  + DB Instances can be reserved to reduce long-term costs.
* Storage (per GB per month) + Backup (which exceeds 100% of storage)
* Standard AWS data transfer pricing.
* I/O requests per month & Provisioned IOPS (Amazon Aurora only).

## Miscellaneous Features

* Parameter Groups act as a ‘container’ for engine configuration values that can be applied to one or more DB instances.
  + Dynamic parameters are applied immediately.
  + Static parameters are applied after instance reboot.
* Performance degradation of a DB instance can happen due to multiple reasons such as snapshot creation (if multi-AZ is not enabled), backups, multi-AZ peer creation, read replica creation and scaling storage.
* During maintenance window in a multi-AZ deployment, maintenance is done on the standby, after which it is promoted to the primary. Lastly, maintenance is done on the old primary which becomes the new standby.

# DynamoDB

## Definition

* Non-relational distributed database service that is fast and flexible for any scale with single-digit millisecond latency.
* It is a fully-managed, highly available AZ replicated across three AZs.
* Enables customers to offload the administrative burdens of operating and scaling distributed databases to AWS so that they don’t have to worry about hardware provisioning, setup and configuration, throughput capacity planning, replication, software patching, or cluster scaling.

## Use Cases

* It supports both document and key value data models.
* It's flexible data model and reliable performance make it great a great fit for mobile web gaming, ad-tech, IOT and many other applications.

## Key Components

* Table
  + Collection of items/data with no pre-defined schema.
* Item
  + Collection of attributes.
* Attributes
  + Fundamental data element.
  + Nested attributes can be 32 layers deep.
* Primary Key
  + Attribute that can uniquely identify each item in a table.
  + Must be scalar.
  + It can be of two types:
    - Partition key: A simple primary key, composed of one attribute.
    - Partition key and sort key: Composed of two attributes.
  + DynamoDB uses partition key value as an input to an internal hash function, the output of which determines the partition where the item will be stored.
    - A partition key should distribute the data well-enough.
    - All items with the same partition key are stored together, in sorted order of the sort key value.
    - If no sort key is used, no two items can have the same partition key.
* Secondary indexes
  + Allows one to query the data using an alternate key, in addition to queries against the primary key.
  + One can create more than one secondary indexes on a table.
  + Two kinds of indexes:
    - Global secondary index (GSI)
      * An index with a partition key and sort key that can be different from those on the table.
      * It creates a whole new table with the newly defined primary key.
        + The new table must consist of the original table’s partition and sort key.
        + Additional attributes can be included from the original table.
        + The new table must have its throughput configured again in the form of Read/Write Capacity Units.
      * It need not be defined at table creation time.
        + It can be added and modified on-the fly, unlike LSI.
    - Local secondary index (LSI)
      * An index that has the same partition key as the table, but a different sort key.
      * It must be defined at the table creation time.
      * The sort key must consistent of exactly one scalar attribute.
        + The attribute must be a scalar String, Number or Binary.
      * One can have up to 5 LSI per table.

## Performance

* It’s backed by SSD-storage for performance improvement and is available across three distinct geographical data centers.
* Its consistency model consists of:
  + Eventually consistent reads
    - This option maximizes read throughput.
    - However, an eventually consistent read might not reflect the results of a recently completed write.
    - All copies of data usually reach consistency within a second. Repeating a read after a short time should return the updated data.
  + Strongly consistent reads
    - DynamoDB also gives you the flexibility and control to request a strongly consistent read if your application, or an element of your application, requires it.
    - A strongly consistent read returns a result that reflects all writes that received a successful response before the read.
  + DynamoDB supports ‘optimistic locking’ consistency model.
    - One can specify conditions for updates or deletes to ensure an item hasn’t changed before altering it.
    - E.g. Update an item only if it’s version number 1, etc.
* Amazon DynamoDB Accelerator (DAX) is a managed in-memory cache service for DynamoDB.
  + One can assign up to 10 nodes to a DAX cluster.
  + DAX supports multi-AZ.
  + The default TTL is 5 minutes.
  + It is launched inside a VPC subnet with appropriate IAM roles and security groups configured.
* DynamoDB auto scaling uses the AWS Application Auto Scaling service to dynamically adjust provisioned throughput capacity on one’s behalf, in response to actual traffic patterns.
  + This enables a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic, without throttling.
  + When the workload decreases, Application Auto Scaling decreases the throughput so that one doesn't pay for unused provisioned capacity.

## Throughput

* A DynamoDB table must have read and write capacity units.
* Read Capacity Units (RCU)
  + Throughput for reads.
  + One RCU represents one strongly consistent read per second , or two eventually consistent reads per second, for an item up to 4 KB in size
    - If items are larger than 4 KB, more RCU are consumed.
  + Items are rounded off to nearest upper 4KB increment.
  + Example: 10 eventually consistent reads per second of objects of size 6 KB requires (10/2) \* (8/4) = 10 RCU.
    - One can have 2 eventually consistent reads per second for an object.
    - 6 KB is rounded off to 8 KB.
* Write Capacity Units (WCU)
  + Throughput for writes.
  + One WCU represents one write per second for an item up to 1 KB in size.
  + If items are larger than 1 KB, more WCU are consumed.
    - Items around rounded off to nearest upper 1 KB.
  + Example: Writing 120 objects per minute of 1.5 KB each requires (120/60) \* 2 = 4 WCU.
    - WCU is calculated at second level, not minute. Hence divided by 60.
    - Objects are rounded off to nearest upper KB.
* Throughput can exceed temporarily to meet demand using ‘burst credit’.
  + If ‘burst credit’ is empty, one gets a ‘*ProvisionedThroughputException*’ error.
  + This can happen if:
    - One deals with very large items (more RCU/WCU required).
    - Hot partitions i.e. one partition key is being read too many times (WCU/RCU is spread across partitions).
  + Exponential back-off is advised for such cases.
  + If it’s an RCU issue, DAX can be enabled.
* Autoscaling can be enabled to scale as per demand.

## DynamoDB Streams

* An optional feature that captures data modifications events in DynamoDB tables.
  + An event is represented by a stream record, a group of which are called shards.
  + A stream record contains the name of the table, the event timestamp and other metadata.
* It can capture events like the image of a new item being added to a table, before-and-after image of the attributes of an updated item, or an image of an entire deleted item.
* It can be paired with AWS Lambda to trigger events based on changes to a table.
* Stream records have a lifetime of 24 hours before they are deleted.

## DynamoDB Basic APIs

For writing data, we have:

* *PutItem*
  + Write data to DynamoDB (create or fully replace).
  + Consumes WCU.
* *UpdateItem*
  + Update data in DynamoDB (partial update of attributes).
  + Possibility to use Atomic Counters and increase them.
* *BatchWriteItem*
  + One can put up to 25 PutItem and/or DeleteItem in one call.
  + There is a limit of 16 MB of data written/ or 400 KB per item.
  + Allows one to save in latency by reducing the number of API calls done against DynamoDB.
  + Operations are done in parallel for better efficiency.
  + Exponential back-off can be used if part of a batch fails due to some reasons (capacity exceed).
* Condition Writes
  + Accept a write/update only if conditions are respected, otherwise reject.
  + Helps with concurrent access to items.
  + There is no performance impact.

For deleting data, we have:

* *DeleteItem*
  + Delete an individual row.
  + One can perform a conditional delete.
* *DeleteTable*
  + Delete a whole table and all its items.
  + Much faster than calling DeleteItem on all items.

For reading data, we have:

* *GetItem*
  + Read based on primary key (HASH or HASH-RANGE).
  + By default, it performs eventually consistent reads.
  + One can opt for strongly consistent reads (which will take twice as much RCU).
  + *‘ProjectionExpression’* can be specified to include only certain attributes & save network bandwidth.
* *BatchGetItem*
  + Fetch up 100 items/ 16 MB of data.
  + Items are retrieved in parallel to minimize latency.

For querying data, we have:

* *Query*
  + It returns items based on:
    - Partition Key value (must be an ‘=’ operator).
    - Sort Key value (must be ‘= , <, <=, >, >=, BETWEEN, BEGIN) – optional.
    - *‘FilterExpression’* to further filter data. (client-side filtering)
  + Returns up to 1 MB of data or number of items specified in ‘*Limit’*.
  + One can do pagination of results.
  + One can query a table from a global/local secondary index.
* *Scan*
  + It returns the entire table and then filters the data, making it highly inefficient.
  + It returns up to 1 MB of data and uses pagination to keep on reading.
  + Consumes a lot of RCU (Can be reduced by using ‘*Limit*’).
  + One can use parallel scans for faster performance.
    - Multiple instances scan multiple partitions at the same time.
    - Increases throughput and RCU consumed.
  + One can use ‘*ProjectExpression*’ and ‘*FilterExpression*’ (no change to RCU).

## Miscellaneous Features

* Throttling can be enabled on DynamoDB for a table or an index.
* One can specify a Time-To-Live (TTL) for items in a table so that they can be deleted automatically from the database.
* One can do point-in-time restores in DynamoDB like RDS with no impact in performance.
* Amazon DynamoDB global tables provide a fully managed solution for deploying a multi-region, multi-master database, without having to build and maintain a replication solution, making it ideal for massively scaled applications, with globally dispersed users.
* Maximum size of an item is 400 KB.
* Data types supported include:
  + Scalar Types: String, Number, Binary, Boolean, NULL
  + Document Types: List, Map
  + Set Types: String set, Number set, Binary set

# Redshift

## Definition

* Fast, fully managed data warehouse service that makes it simple and cost-effective to analyze data using standard SQL and existing Business Intelligence (BI) tools.
* It allows you to run complex analytic queries against petabytes of structured data, using sophisticated query optimization, columnar storage on high-performance local disks, and massively parallel query execution.
* Includes Amazon Redshift Spectrum, allowing you to directly run SQL queries against exabytes of unstructured data in Amazon S3.
  + No loading or transformation is required, and one can use open data formats, including Avro, CSV, Grok, Ion, JSON, ORC, Parquet, RCFile, RegexSerDe, SequenceFile, TextFile, and TSV.
  + It automatically scales query compute capacity based on the data being retrieved, so queries run fast regardless of dataset size.

## Performance

* Columnar Data Storage
  + Amazon Redshift organizes the data by column, instead of rows.
  + Unlike row-based systems, which are ideal for transaction processing, column-based systems are ideal for data warehousing and analytics, where queries often involve aggregates performed over large data sets.
  + Since only the columns involved in the queries are processed and columnar data is stored sequentially on the storage media, column-based systems require far fewer I/Os, greatly improving query performance.
* Advanced Compression
  + Columnar data stores can be compressed much more than row-based data stores because similar data is stored sequentially on disk.
  + Amazon Redshift employs multiple compression techniques and can often achieve significant compression relative to traditional relational data stores.
* Massively Parallel Processing (MPP):
  + Amazon Redshift automatically distributes data and query load across all nodes.
  + It makes It makes easy to add nodes to the data warehouse and enables one to maintain fast query performance as the data warehouse grows.
* Redshift Spectrum:
  + Redshift Spectrum enables you to run queries against exabytes of data in Amazon S3 without requiring an ETL pipeline.
  + When a query is issued, a query plan is generated. The data present in local vs the data required from S3 is determined to minimize unnecessary reads.

## Deployment Mode

* Redshift currently supports single-AZ deployment.
* Redshift cluster can be deployed in two modes:
  + Single-Node configuration
    - Start with a single node, 160GB data warehouse and scale all the way to a petabyte or more.
    - Enables one to get started with Amazon Redshift quickly and cost-effectively and scale up to a multi-node configuration as requirements grow.
  + Multi-Node configuration
    - Leader node
      * Receives queries from client applications, parses the queries and develops execution plans, which are an ordered set of steps to process these queries.
      * The leader node then coordinates the parallel execution of these plans with the compute nodes, aggregates the intermediate results from these nodes and finally returns the results back to the client applications.
      * One is not billed for leader node.
    - Compute node
      * Executes the steps specified in the execution plans and transmit data among themselves to serve these queries.
      * The intermediate results are sent back to the leader node for aggregation before being sent back to the client applications.
      * A single Redshift cluster can have up to 128 compute nodes.
* One can have two types of nodes on Redshift:
  + DC
    - Dense storage node types optimized for large data workloads.
    - Uses HDD storage.
  + DS
    - Dense compute node types optimized for performance-intensive workloads
    - Uses SSD storage.
    - Less storage but much faster I/O operations.

## Backup & Recovery

* Redshift enables automated backups of the data warehouse cluster with a 1-day retention period, by default, which can be extended to a maximum of 35 days.
* Redshift always attempts to maintain at least three copies of the data: the original and replica on the compute nodes and a backup in S3.
* Redshift can also asynchronously replicate the snapshots to S3 in another region for disaster recovery.

## Miscellaneous Features

* In Amazon Redshift, one can use workload management (WLM) to define the number of query queues that are available, and how queries are routed to those queues for processing.
  + WLM is part of parameter group configuration.
  + A cluster uses the WLM configuration that is specified in its associated parameter group.
  + A custom WLM would require the creation of a new parameter group, which would then be associated to the Redshift clusters.
* With Amazon Redshift Enhanced VPC Routing enabled, Redshift forces all COPY and UNLOAD traffic between the cluster and data repositories through Amazon VPC, allowing one to use standard VPC features such as VPC security groups, NACLs, VPC endpoints, etc.

# Aurora

## Definition

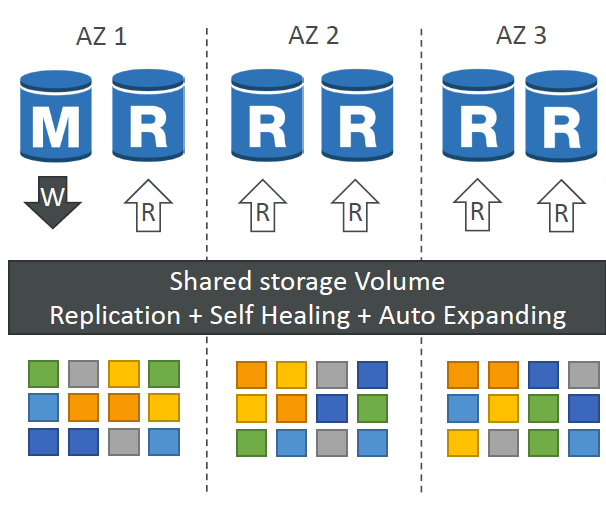
* Relational database engine that is MySQL/PostgreSQL compatible.
* Combines the speed and availability of high-end commercial databases with the simplicity and cost-effectiveness of open source databases.
* Utilizes RDS platform to automate administrative tasks.
* Provides five times better performance than MySQL at a price point one tenth of a commercial database.
* Starting from 10GB storage, it can scale to up to 64TB in 10GB increments.
* Leverages cloud ecosystem by integrating services like Lambda, S3, IAM, CloudWatch, etc.

## Performance

* By tightly integrating the database engine with an SSD-based virtualized storage layer purpose-built for database workloads, Aurora reduces writes to the storage system, minimizes lock contention and eliminates delays created by database process threads.
  + The storage and logging layer have been combined together to build a log-structured distributed storage layer.
  + Storage volume is striped across hundreds of storage nodes distributed over 3 AZs.
  + Multiple read scanners can access the same lock chain.
* Up to five times better performance than MySQL and three times better than PostgreSQL. Some of the reasons can be attributed to:
  + Fewer I/Os processed.
  + Minimized network packets.
  + Caching of prior results.
* Aurora improves upon traditional engines with its highly concurrent workloads.
* Aurora provides ‘Parallel Query’ feature that enables distribution of the computational load of a single query across thousands of CPU’s in Aurora’s storage layer, albeit at a higher I/O cost.
* Aurora can horizontally scale to up to 15 read replicas.
* Fast-DDL is a service provided which can speed up DDL statements to a large degree under some circumstances.
* It has self-healing property wherein if some data gets corrupted, it can repair itself via peer-to-peer replication.

## Backup & Recovery

* Automated backups are always enabled on Amazon Aurora DB Instances. Backups do not impact database performance, neither does taking snapshots.
* Snapshots can be shared to other AWS accounts.
* Aurora automatically maintains 6 copies of the data across 3 Availability Zones and will automatically attempt to recover the database in a healthy AZ with no data loss.
* Aurora is designed to transparently handle the loss of up to two copies of data without affecting database write availability and up to three copies without affecting read availability.
* Automated failover is available only for Aurora replicas, not MySQL replicas.



## Aurora Serverless

* Amazon Aurora Serverless is an on-demand, autoscaling configuration for the MySQL-compatible edition of Amazon Aurora, providing a relatively simple, cost-effective option for infrequent, intermittent, or unpredictable workloads i.e. where the peak to average workload ratio is high.
* An Aurora Serverless DB cluster automatically starts up, shuts down, and scales capacity up or down based on an application's needs.
* While it saves on costs, some of the disadvantages include:
  + Accessible only within the VPC. VPN Connection/VPC Peering is not supported.
  + Cold starts when the database is paused.
  + Only MYSQL compatible with fixed port.

## Miscellaneous Features

* Aurora endpoints:
  + Aurora has a writer endpoint which client uses to write data. The endpoint points to the Master node.
  + Aurora has a single reader endpoint which points to one or more read replicas, much like a connection load balancer. (This is due to the fact that it can be difficult to track individual read replica endpoints as the cluster auto-scales)
* Patches for DB engine occur in a new process. Once the engine is available, then the switch occurs from old engine to the new one. This avoids patch up downtime.
* Advanced logging and auditing do not impact performance.
* Performance Insights can provide query-level insights, helping identify source of bottlenecks, top SQLs, etc.

# ElastiCache

## Definition

* Web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud.
* The service improves the performance of web applications by allowing one to retrieve information from fast, managed, in-memory caches, instead of relying entirely on slower disk-based databases.
* A popular use case is applications that require distributed session management.

## Caching Engines

* Memcached (in-memory object store)
  + Should be used when simple offload from database is required.
  + Has the ability to scale horizontally by adding or remove nodes.
  + Multi-threaded performance is provided.
  + Data is lost if node fails/AZ experiences outage.
* Redis (in-memory key-value store)
  + Should be used when
    - when working with advanced data types is required.
    - when ranking/sorting datasets is required.
    - when persistence or multi-AZ support is required.
    - when backup & restore capabilities are required.
    - when publish/subscribe capabilities are required.
  + Supports read replicas.

# CloudWatch

## Definition

* Monitoring service to monitor AWS resources, as well as the applications that are run on AWS.
* Can be used to collect and track metrics, collect and monitor log files, and set alarms, enabling one to gain system-wide visibility into resource utilization, application performance, and operational health.
* It can monitor services like
  + Compute
    - EC2 instances
    - Autoscaling Groups
    - Elastic Load Balancers
    - Route 53 Health Checks
  + Storage & Content Delivery
    - EBS Volumes
    - Storage Gateways
    - CloudFront

## Data Retention

* Data points with a period of less than 60 seconds are available for 3 hours. These data points are high-resolution custom metrics.
* Data points with a period of 60 seconds (1 minute) are available for 15 days.
* Data points with a period of 300 seconds (5 minute) are available for 63 days.
* Data points with a period of 3600 seconds (1 hour) are available for 455 days (15 months)

## Dashboards

* Dashboards can be created to see what is happening in the AWS environment.
* Dashboards are global and can include graphs from different regions.
* One can set up automatic refresh (10s, 1m, 2m, 5m, 15m).

## Metrics

* Metric is a variable to monitor.
  + Metrics can have up to 10 attributes known as dimensions, such as instance id, environment, etc.
    - These dimensions can be used to segment metrics.
  + Metrics have timestamps.
* Host level metrics (Collected at 5 minutes to 1-minute interval) consist of:
  + CPU
    - CPU Utilization of EC2 instance
    - For burstable instance types, Credit Usage/Balance
  + Network
    - Network packets in & out of an EC2 instance
  + Disk
    - Disk Reads activity of an EC2 instance. (only for instance store)
  + Status Check
    - Instance status: Check the EC2 VM.
    - System status: Check the underlying hardware.
  + Hypervisor
* Custom metrics must be created for the following items (Collected at 1-minute to 1-second intervals):
  + Memory utilization
  + Disk Swap utilization
  + Disk Space utilization
  + Page File utilization
  + Log collection

## Alarms

* Alarms can be set to notify one when particular resource thresholds are hit.
* Alarm states include:
  + OK
  + INSUFFICIENT DATA
  + ALARM

## Events

* CloudWatch Events helps one respond to state changes in one’s AWS resources.
* It can either be event-driven or scheduled regularly using CRON.
  + Events can originate from almost all the services AWS offers. E.g. Deletion of object in a S3 bucket.
* Based on the event/schedule, CloudWatch events can perform a number of tasks such as:
  + Stop/Reboot/Terminate an instance or taking a snapshot.
  + Invoke a Lambda function.
  + Boot up a Step Function workflow.
  + SNS, SQS, etc.
* It creates a small JSON document to give information about the change.

## Logs

* CloudWatch Logs helps one aggregate, monitor and store logs.
* Logs can be imported from a variety of services like S3 Access Logs, VPC Flow Logs, Lambda function logs, etc.
* Logs can be exported to S3 for archival or ElasticSearch cluster for further analytics.
* Log storage structure:
  + Log Groups: Arbitrary names, usually representing an application.
  + Log Stream: Instances within an application / log files / containers.
* Encryption of logs is at the group level using KMS.
* To push logs from EC2 machine or on-premise servers, one can run CloudWatch Agents on them.
* CloudWatch Logs can be filter expressions.
  + Metric filters can be used to trigger alarms.
* CloudWatch Logs Insights can be used to query logs and add queries to CloudWatch dashboards.

## Miscellaneous Features

* Custom metrics can have a minimum resolution of 1 second.
* The alarm on a high-resolution custom metric can be triggered as often as 10 seconds.

# CloudFront

## Definition

* It is a web service that gives businesses and web application developers an easy and cost-effective way to distribute content with low latency and high data transfer speed using a network of data centers called edge locations around the world.
* It is a self-service, pay-per-use offering, requiring no long-term commitments or minimum fees.

## CloudFront Distributions

It tells CloudFront where one wants content to be delivered from, and the details on how to track and manage content delivery.

Its various configurations include:

* Content origin
  + The S3 bucket, MediaPackage channel, or HTTP server from which CloudFront gets the files to distribute.
* Access
  + Whether the files are to be made to available to everyone or certain users.
  + Geo-restrictions can be applied here.
* Security
  + Whether one wants CloudFront to require its users to use HTTPS to access the content.
* Cookies
  + Whether one wants CloudFront to forward cookies or query strings to one’s origin.
* Access Logs
  + Whether one wants CloudFront to create access logs that show viewer activity.

## Performance and Availability

* CloudFront allows one to setup multiple origins to enable redundancy with Origin Failover.
  + One must have a distribution with at least two origins, setting one as the primary.
  + When defining cache behavior, configure CloudFront to failover to second origin when the primary returns specific status codes.
* CloudFront is optimized for both dynamic and static content, providing extensive flexibility for optimizing cache behavior, coupled with network-layer optimization for latency and throughput.

## Origin Access Identity

* With S3 as origin, objects must be granted public read permissions and hence the objects are accessible both from S3 as well as CloudFront.
* Users could directly access objects on S3. This would be detrimental since:
  + Users could bypass the controls provided by CloudFront signed URLs or signed cookies.
  + CloudFront access logs would be incomplete.
* Origin Access Identity (a special user) can be used to prevent users from directly accessing objects in S3.
* S3 buckets/objects can be configured to only provide access to OAI, which can then fetch & service content on general user’s behalf.

## Miscellaneous Features

* CloudFront uses these cache control headers to determine how frequently it needs to check the origin for an updated version of that file. For expiration period set to 0 seconds, CloudFront will revalidate every request with the origin server.

# CloudTrail

## Definition

* It is a service that enables governance, compliance, operational auditing, and risk auditing of one’s AWS account.
* It provides visibility into user activity by recording actions taken on one’s account.
  + It records important information about each action, including who made the request, the services used, the actions performed, parameters for the actions, and the response elements returned by the AWS service.
  + This information helps one track changes made to AWS resources and troubleshoot operational issues.
* CloudTrail makes it easier to ensure compliance with internal policies and regulatory standards.

## Trails

* One can set up trails for to get a detailed list of API calls done.
  + The trail could be applicable to only one region or be configured for all regions.
* These trails can be directly sent to a S3 bucket.
  + The default encryption methodology is SSE-S3.

# Route 53

## Definition

* Provides highly available and scalable Domain Name System (DNS), domain name registration, and health-checking web services.
* Designed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating names like example.com into the numeric IP addresses, such as 192.0.2.1, that computers use to connect to each other.
* Route 53 record sets are not region-specific.

## DNS Record Sets

* Record sets come in the following types:
  + SOA Records
    - Start-Of-Authority record set.
    - Contains name of the server that has supplied the data for the zone, administration of the zone, current version of data file and default number of seconds for the TTL file on resource records.
  + NS Records
    - They are used by top-level domain servers to direct traffic to the Content DNS server which contains the authoritative DNS records.
  + A Records
    - “Address” records are the most fundamental type of record.
    - Used by computer to translate the name of the domain to an IP address.
    - For IPv6, it is known as ‘AAAA’ records.
  + CNAMES
    - A canonical name can be used to resolve one domain name to another.
  + Alias Records
    - Used to map resource record sets in hosted zone to ELBs, CloudFront distributions, or S3 buckets that are configured as websites.
    - Alias records work like CNAME, wherein one can map a DNS name to another.
    - A key difference is that CNAME cannot be used map to naked domain names i.e. zone apex record.
    - Has better network performance than CNAME but loses Geolocation information.
  + MX Records
    - Used for mail exchange.
  + PTR Records
    - Reverse of an ‘A’ record i.e. used to look up a domain name based on IP address.
  + CAA Records (Certificate Authority Authorization)
    - Let’s one specify which certificate authorities (CAs) are allowed to issue certificates for a domain or subdomain.
  + NAPTR Records (Name Authority Pointer Record)
  + SPF Records (Sender Policy Framework)
  + SRV Records (Service Locator)
  + TXT Records (Text Record)
* The length that the DNS record is cached on either the resolving server or the users own local PC is equal to the value of TTL in seconds
* Individual record sets can be associated with health checks.
  + If a record set fails a health check, it will be removed from Route 53 until it passes the health check.
  + SNS notifications can be set up in case of failed health checks.

## Routing Policies

* Simple Routing
  + DNS can have one record with multiple addresses.
  + If multiple values are specified, Route 53 returns values in a random order.
  + Health checks cannot be attached.
* Weighted Routing
  + Allows one to split incoming traffic based on different weights assigned.
  + Health checks can be attached.
* Latency-base Routing
  + Allows one to your traffic based on the lowest network latency for the end user.
  + Physically closer servers need not guarantee lowest latency.
* Failover Routing
  + Allows one to create an active/passive set-up.
  + If primary server fails a health check, failover is automated to the passive secondary.
* Geolocation Routing
  + Allows one to direct traffic based on the geographical location of end-users.
  + One should also specify a default IP.
* Geoproximity Routing
  + Allows one to route traffic to resources based on the geographical location of the end-users as well as the resources themselves.
  + One can optionally choose to route more or less traffic to a given resource by specifying a value known as bias.
  + One can only use geoproximity routing using Route 53 traffic flow.
* Multivalued Answer Routing
  + Allows one to specify multiple values for almost any record (similar to simple routing), but Route 53 only returns values for healthy resources.

## Health Checks

* One can configure health checks in Route 53, such as to monito the health of an AWS endpoint.
* The health checks can be configured to occur every X seconds and can be deemed to fail if Y health checks fail.
  + By default, health checks occur every 30 seconds. For additional cost, one can configure health checks to occur every 10 seconds.
* The health checks can be integrated with CloudWatch.

## Miscellaneous Features

* If Route 53 is used to map to a static website hosted on S3, the bucket name must be the same as the domain name.
* Each DNS record must have its TTL specified.
* One can buy domains from a 3rd party website but can still use Route 53.
  + One would need to create a public hosted zone on Route 53.
  + Then, NS records in the 3rd party website would need to be updated to use Route 53’s name servers.

# Virtual Private Cloud (VPC)

## Definition

* Amazon VPC lets one provision a logically isolated section of AWS Cloud i.e. a virtual data center where resources can be launched in a virtual network that one defines.
* VPCs are defined at region-level.
* One has complete control over the virtual networking environment, including selection of IP address range, creation of subnets, and configuration of route tables and network gateways.
* One can place public-facing subnet for webservers that have access to the internet and place backend systems such as databases and application servers in a private-facing subnet with no internet access.
* One can create a Hardware Virtual Private Network (VPN) connection between the corporate datacenter and the VPC and leverage the AWS Cloud as an extension of the corporate datacenter.
* One can have VPC Peering i.e. connect one VPC to another via a direct network route using private IP addresses.
  + Instances behave as if they were on the same private network.
  + VPCs can be peered to other AWS accounts and across regions.
  + Peering is not transitive in nature.
* The VPC consists of Virtual Private Gateways, Route Tables, Network Access Control Lists, Subnets and Security Groups.
* When a VPC is created, a default route table, Network Access Control List and a security group is also created. Subnets and Internet Gateways are not created.

## Subnets

* When a VPC is created, a subnet is not created by default.
* Subnets are created at AZ-level.
* AWS reserves 5 IP addresses per subnet. E.g. If the Subnet CIDR block is 10.0.0.0/24, then
  + 10.0.0.0 – Reserved for network address
  + 10.0.0.1 – Reserved for the VPC router
  + 10.0.0.2 – Reserved for the DNS server of the VPC
  + 10.0.0.3 – Reserved for future use
  + 10.0.0.255 – Reserved for network broadcast, which is not allowed in a VPC.
* AWS recommends
  + CIDR /16 for VPC (65,536 addresses) and /24 for subnets (251 addresses) for IPv4.
  + CIDR /56 for VPC and /64 subnets for IPv6.
  + Use multiple AZs per VPC via subnets.

## Internet Gateway (IGW)

* IGW helps one’s VPC instances to connect with the internet.
* IGW scale horizontally and are highly available.
* They must be created separate from VPC.
  + Each VPC can only be attached to one IGW and vice-versa.

## NAT Instances

* They are individual EC2 instances which are always in a public subnet behind a security group.
* By default, EC2 instances do source/destination checks where it verifies that it is either a source or destination for a networking process. NAT instances have this check disabled.
* They are used to provide outbound internet access to instances in a private subnet which do not have a public IP address. There must be a route out of a private subnet to the NAT instance for the subnet to access the internet.
* The amount of traffic that can be handled by the NAT instance is bottlenecked by its size.
* To achieve high availability, one can use Autoscaling groups, multiple subnets in different AZs, and script to automate failover.

## NAT Gateways

* They are highly available virtual device that are redundant inside an AZ i.e. they are not a single EC2 machine and can survive failover of the instances that power the NAT Gateway.
* There can only be one NAT Gateway in one AZ (in a public subnet).
* Preferred by enterprises since it’s a scalable solution compared to NAT instances.
* Networking speeds start at 5 Gbps and can go up to 45 Gbps.
* Unlike NAT instances,
  + Patching is not required.
  + Source/destination checks need not be disabled.
  + Security groups are not associated.
* Route tables still need to be configured for the private subnets to access NAT gateways.
* If resources span multiple AZ, multiple NAT Gateways should be configured for their respective AZs to achieve AZ-independent architecture in case an AZ goes down.
* The IPv6 equivalent of NAT gateways are Egress-only internet gateways.

## Network Access Control List (NACL)

* Every subnet created in a VPC is associated with a default NACL for that VPC. If a custom NACL is not explicitly created, the default NACL is used for that subnet.
* It allows one to filter traffic at subnet level, unlike Security Groups that filter traffic at instance/application level.
* Default NACL allow all inbound and outbound traffic by default, whereas custom VPC denies all traffic by default.
* An NACL can be associated to multiple subnets, however, a subnet can only have a single NACL associated with it. If a new NACL is associated with a subnet, the previous association is removed.
* NACL act before Security Groups and can block specified IP addresses from accessing the subnet.
* NACL have separate inbound/outbound rules and are stateless i.e. responses to allowed inbound traffic are subject to the rules of outbound traffic and vice-versa, unlike Security Groups.
* Rules in NACL are evaluated in chronological order, with the lowest rule number being evaluated first. A lower numbered ALLOW rule can trump a higher-numbered DENY rule for the same traffic type.

## VPC Flow Logs

* It is a feature that enables one to capture information about the IP traffic going to and from network interfaces in the VPC.
* Flow log data is stored using CloudWatch Logs.
* The flow log can be created at three levels
  + VPC level
  + Subnet level
  + Network interface level
* Configuration of a flow log cannot be changed such as IAM role.
* One cannot enable flow logs for VPCs that are peered with one’s own VPC unless the peer VPC is in the same AWS account.
* Some traffic is not monitored such as:
  + Traffic generated by instances when they contact Amazon’s DNS server. If one’s own DNS server is used, it is logged.
  + Traffic generated by Windows instance for Amazon Windows License Activation
  + Traffic to and from 169.254.169.254 for instance metadata.
  + DHCP traffic.
  + Traffic to the reserved IP address for the default VPC router.

## Bastion Hosts

* A bastion host is a special purpose computer on a network specifically designed and configured to withstand attacks.
* A Bastion is used to securely administer EC2 instances (Using SSH/RDP).
* NAT Gateway cannot be used as a bastion Host.

## Direct Connect

* Enables one to directly connects on-premise data center to AWS Cloud.
* On-premise data centers connect their routers with Amazon Direct connect routers (cross connect) in a physical location.
* Direct connect routers then use AWS's backbone network to connect to AWS cloud.
* AWS Direct should be used for
  + High throughput workloads i.e. lots of network traffic.
  + When stable and reliable connection is required.
* Redundancy can be increased by establishing a second Direct Connect connection.

## VPC Endpoints

* A VPC endpoint enables one to privately connect one's VPC to supported AWS services without requiring an internet gateway, NAT device, VPN connection or AWS Direct Connect connection.
* Instances in one's VPC do not require public IP address to communicate with resources in the service.
* Traffic between VPC and the other services do not leave the Amazon network.
* They are horizontally scaled, redundant and highly available VPC components that allow communication between instances in the VPC and services without imposing availability risks or bandwidth constraints on the network traffic.
* There are two types of VPC endpoints
  + Interface endpoints
    - An interface endpoint is an elastic network interface with a private IP address that servers as an entry point for traffic destined to a supported service.
  + Gateway endpoints
    - Essentially like NAT gateways, they support services like Amazon S3 and DynamoDB.

## VPC Peering

* Enables full private IP connectivity between two VPCs.
* One can peer VPCs across regions and accounts.
* VPC CIDR ranges must not overlap.
* VPC peering relationship does not support edge-to-edge routing for the following connection types:
  + A VPN connection or an AWS Direct Connect connection to a corporate network
  + An internet connection through an internet gateway.
  + An internet connection in a private subnet through a NAT device.
  + A VPC endpoint to an AWS service; for example, an endpoint to Amazon S3.
  + A ClassicLink connection. One can enable IPv4 communication between a linked EC2-Classic instance and instances in a VPC on the other side of a VPC peering connection. However, IPv6 is not supported in EC2-Classic, hence this connection cannot be extended for IPv6 communication.

## Miscellaneous Features

* Egress-only internet gateways for IPv6 allow outbound internet access to private subnets, like NAT Gateways do for IPv4.
* Virtual Private Gateways, together with Customer Gateways can be used to extend on-premise VPN to AWS VPC.

# Elastic Load Balancing (ELB)

## Definition

* It is a managed load-balancing service for AWS deployments.
* It operates on a logical grouping of homogeneous targets known as a ‘Target Group’.
  + Targets include EC2 instances, IP addresses and containers.
  + For Application Load balancers, targets may also include Lambda functions for some regions.
* It automatically distributes incoming application traffic across multiple target groups, and scales resources to meet traffic demands.
* One can configure rules for each of the listeners present on the load balancer. The rules include conditions and corresponding actions if the conditions are satisfied.
  + The supported conditions are Host header, path, HTTP headers, methods, query parameters, and source IP CIDRs.
  + The supported actions are redirect, fixed response, authenticate, and forward.

Once this is set up, the load balancer will use the rules to determine how a particular HTTP request should be routed.

One can use multiple conditions and actions in a rule and in each condition can specify a match on multiple values.

* It is defined over multiple AZs in a particular VPC/region.
  + Only one subnet per AZ can be attached to the load balancer.
  + Cross-Zone Load Balancing must be enabled to load balance over different AZs.

## Elastic Load Balancer Types

* Application Load Balancer
  + Best suited for load balancing of HTTP/HTTPS traffic, microservices and containerized applications.
  + They operate at layer 7 and are feature rich & application aware.
  + They are intelligent (E.g. If one switches language from English->French on browser, Load balancer will start balancing French servers), and one can create advanced request routing, sending specified requests to specific web servers.
  + Some other improvements over the classic load balancer:
    - Support for WebSockets and HTTP/2.
    - Improved health checks and CloudWatch metrics.
    - Improved performance for real-time and streaming applications.
    - Ability to redirect from one URL to another.
      * HTTP to HTTP E.g. http://this.com to http://this.com:8080
      * HTTP to HTTPS E.g. http://this.com to https://this.com (Security boost)
      * HTTPS to HTTPS E.g. https://this.com:443 to https://this.com:40443
    - Ability to auto response to HTTP requests based on any criteria supported by content-based routing rules, without needing to reach out to the application. E.g. error messages.
    - Ability to warm up new targets before receiving their fair share of requests for applications that require low-latency.
  + They must be enabled over multiple AZs.
* Network Load Balancer
  + Best suited for load balancing of TCP traffic where extreme performance is required.
  + Operating at the connection layer 4, it is capable of handling millions of requests per second, while maintaining ultra-low latency and high throughput.
  + Some additional features include:
    - It automatically gets assigned a single IP (Elastic) per AZ to help with white-listing for firewalls, zero-dollar billing and other use case.
    - Ability to create network TCP health checks which enable fast-failover.
* Classic Load Balancer
  + Legacy Load Balancers. One can load balance HTTP/HTTPS applications and use Layer-7 specific features, such as X-forwarded and sticky sessions.
  + One can also use strict Layer 4 load balancing for applications that rely purely on TCP protocol.

## Advanced Concepts

* Sticky Sessions
  + Classic Load Balancers routes each request independent to the registered EC2s with the smallest load.
  + Sticky sessions allow one to bind a user’s session to a specific EC2 instance. This ensures that all requests from the user during the session are sent to the same instance.
  + Application Load Balancers can have sticky sessions enabled as well. However, the traffic will be sent at the target group level.
    - When to use: When an application is writing to disk on an EC2 instance i.e. storing locally
    - When not to use: When all the traffic is going to one single instance.
* Cross Zone Load Balancing
  + Enables one to load balance across multiple AZs.
  + Without cross zone load balancing, ELBs can only route traffic to the EC2 instances in its own AZ.
  + If there are other instances in other AZ which can receive traffic, then enabling cross zone load balancing can route traffic to them for better distribution.
  + It is disabled by default on Network ELBs but always enabled for Application ELBs. For Classic ELBs, it is enabled by default if created via console.
* Content-based routing
  + Allows one to direct traffic to different target groups based on the path or host filed contained in the HTTP header.
  + A listen is created with rules to forward requests based on the URL path.
  + Before content-based routing was available, one would need to create separate ELBs for applications to direct traffic based on path. The ELB to be used was determined by DNS.

## Cost Model

* One is charged for each hour (full or partial) that the load balancer runs.
* One is also charged for the number of Load Balancer Capacity Units (LCU) used per hour.
  + LCU is a new metric that defines the maximum resource consumed in any one of pre-determined dimensions such as new connections/active connections, bandwidth or rule evaluation.
* Unless it is a Network Load Balancer, one is not charged for regional data transfer between AZs when cross-zone load balancing is enabled.

## Load Balancer Scheme

* When creating a load balancer, one can choose to make the load balancer one of the following
  + Internet-facing load balancer
    - The nodes of the load balancer have public IP addresses.
    - The DNS name is publicly resolvable to the public IP addresses of the nodes.
    - It can route requests from clients over the internet.
  + Internal load balancer
    - The nodes of the load balancer only have private IP addresses.
    - The DNS name is publicly resolvable to the private IP addresses of the nodes.
    - It can route requests from clients within the VPC of the load balancer.
* Both types of load balancers route traffic to the targets using private IP addresses, making the targets not requiring a public IP address.
* Both types of load balancers can be used together in conjunction with a multi-tiered architecture to serve requests from senders both outside and inside the VPC.

## Miscellaneous Features

* If application stops responding, the ELB responds with a 504 Error (Gateway timeout). Most likely, the application is having issues at the Web Server Layer or at the Database Layer.
* Health Checks check the instance health by talking to it.
  + Instances monitored by ELB are reported as: InService, or OutofService.
  + If the load balancer detects an unhealthy target, it stops sending traffic to it until it detects that the target is healthy again.
* Load Balancers have their own DNS name. The IP address is never provided.
* When an end user hits the ELB, the ELB passes its own internal IP address to the EC2, which logs it as the end user's IP address. If the original IP address of the end user is required, it can be obtained from the ‘X-Forwarded-For’ header. Similarly, protocol (HTTP/HTTPS) and port of the client can be obtained from ‘X-Forward-Proto’ and ‘X-Forwarded-Port’ headers respectively.
* ELB Connection Draining is used to stop sending new requests to a target which is being de-registered whilst continuing in-flight requests for a specified time period, defaulting to 300 seconds.
* Elastic Load Balancing provides access logs that capture detailed information about requests sent to the load balancer and store them in S3.
  + Each log contains information such as the time the request was received, the client's IP address, latencies, request paths, and server responses. One can use these access logs to analyze traffic patterns and troubleshoot issues.
  + Access logs are disabled by default.
  + Access logs, if enables, are automatically encrypted.
* Application Load balancers provide SSL termination.
  + Basically, the load balancer accepts incoming HTTPS traffic, and redirects it to the internal AWS backend via private HTTP connections.
  + This allows one to only have the SSL certificate be attached to the load balancer instead of individual servers.
  + If reduces the CPU cost at server end as the servers need not waste time decrypting traffic.
* All load balancer metrics are directly pushed to CloudWatch metrics, such as:
  + BackendConnectionErrors
  + HealthyHostCount/ UnhealthyHostCount
  + HTTPCode\_Backend\_2XX: Successful request
  + HTTPCode\_Backend\_3XX: Redirected request
  + HTTPCode\_ELB\_4XX: Client error codes
  + HTTPCode\_ELB\_5XX: Server error codes (generated by load balancer).
  + Latency
  + RequestCount
  + SurgeQueueLength: The total number of requests (HTTP listeners) or connections (TCP listeners) that are pending routing to a healthy instance.
    - This helps in scaling ASG.
    - The maximum value is 1024.
  + SpilloverCount: The total number of requests that were rejected because the surge queue was full.
* Application Load balancers attach a custom header ‘X-Amzn-Trace-Id’ to every HTTP request.
  + This is useful in logs/ distributed tracing platform to track a single request.

# CloudFormation

## Definition

* It is a deployment service that enables one to completely script an entire cloud environment via templates.
* Terminating a CloudFormation terminates all resources attached to that CloudFormation.

## Templates

* The templates are generally JSON or YAML-formatted text file.
* The templates are not region-specific.
* The templates can consist of multiple sections, all of which are optional except for the ‘Resources’ section.
  + Format version
    - For version control
  + Description
  + Metadata
  + Parameters
    - To provide dynamic inputs for the template.
    - It enhances reusability & should be used when resource configurations are likely to change in the future.
    - They can be referenced in the template using the ‘!Ref’ (Reference) function.
    - Example:

*DBSubent1:*

*Type: AWS::EC2::Subnet*

*Properties:*

*VpcID: !Ref MyVPC*

* + Mappings
    - Static variables for the template that are hard-coded.
    - They are very handy to differentiate between different environments (dev vs prod), regions, AMI types, etc.
    - They should be used when one knows in advance all the values that can be deduced from variables such as regions, AZ, environment, etc.
    - They can be referenced in the template using ‘!FindInMap’ function that returns a value from a specific key.
      * The syntax is !FindInMap [MapName, TopLevelKey, SecondLevelKey]
    - Example:

*AWSTemplateFormatVersion: “2019-09-09”*

*Mappings:*

*RegionMap:*

*us-east-1:*

*“32”:”ami-6411e20d”*

*“64”:”ami-7a11e213”*

*us-west-1:*

*“32”:”ami-c9c7978c”*

*“64”:”ami-31c2f645”*

*Resources:*

*MyEC2Instance:*

*Type: “AWS::EC2::Instance”*

*Properties:*

*ImageId: !FindInMap [RegionMap, !Ref “AWS::Region”, 32]*

*InstanceType: m1.small*

* + Conditions
    - They are used to control the creation of resources based on a condition.
    - Common conditions include environment/region/parameter value check.
    - Each condition can reference another condition, parameter value or mapping.
    - Other conditions include AND, IF, NOT and OR.
    - Example:

*Conditions:*

*CreateProdResources: !Equals [ !Ref EnvType, prod ]*

*Resources:*

*Mountpoint:*

*Type: “AWS::EC2::VolumeAttachment”*

*Condition: CreateProdResources*

* + Transform
    - Used to specific one or more macros that CloudFormation can use to process the template.
    - For example, AWS::Include transform works with template snippets that are stored separately from the main CloudFormation template.
  + Resources
    - Core of the CloudFormation template.
    - They represent the different components that will be created and configured.
    - Resources are declared and can reference each other.
    - Their type identifiers are in the form: AWS::aws-product-name::data-type-name
    - Each resource must have a type and a set of properties.
      * Some of the properties are mandatory.
      * Some properties might replace existing resources if they are updated.
  + Outputs:
    - Declares optional output values that can be imported into other stacks.
    - Useful for outputting variables such as VPC ID/Subnet ID, etc.
    - One cannot delete a CloudFormation Stack if its outputs are being referenced by another stack.
    - Export property must be present in the source stack with the output resource, and the [!ImportValue] function is sued in the destination stack to import said resource.
    - Example:

*Outputs:*

*StackSSHSecurityGroup:*

*Description: The SSH Securtiy Group for our Project*

*Value: !Ref MyProjectWideSSHSecurityGroup*

*Export:*

*Name:* ***SSHSecurityGroup***

*--Some other CF Stack –*

*Resources:*

*MySecureInstance:*

*Type: AWS::EC2::Instance*

*Properties:*

*AvailabilityZones: us-east-1a*

*ImageId: ami-a4c7edb2*

*InstanceType: t2.micro*

*SecurityGroups:*

*- !ImportValue* ***SSHSecurityGroup***

## Intrinsic Functions

* Fn::Ref
  + It can be leveraged to reference
    - Parameters: Returns the value of the parameter
    - Resources: Returns the physical ID of the underlying resource
* Fn::GetAtt
  + Attributes are attached to any resource created.
  + Example: !GetAtt EC2Insance.AvailabiltyZone
* Fn::FindInMap
  + Used to return a named value from a specific key.
  + Syntax: !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]
* Fn::ImportValue
  + Import values exported in other functions.
  + Example: !ImportValue SSHSecurityGroup
* Fn::Join
  + Join values with a delimiter.
  + Example: !Join [“:”,[a,b,c]] creates “a:b:c”
* Fn::Sub
  + Used to substitute variables from a text.
  + Example: { "Fn::Sub": [ "www.${Domain}", { "Domain": {"Ref" : "RootDomainName" }} ]}
* Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc.)

## Rollback

* During Stack Creation:
  + Default: Everything rolls back (gets deleted).
    - One can look at the log *‘OnFailure=ROLLBACK’*
  + Troubleshoot: Option to disable the rollback and manually troubleshoot.
    - One can look at the log *‘OnFailure=DO\_NOTHING’*
  + Delete: Option to delete the stack entirely.
    - One can look at the log *‘OnFailure=DELETE’*
* During Stack Update:
  + The stack automatically rolls back to the previous known working state.
  + One can look at the logs to see error messages for troubleshooting purposes.

## Deletion Policy

One can specify deletion policies on any resource to control what happens when the CloudFormation stack is deleted.

* DeletionPolicy = Retain:
  + Specify the resource to preserve/backup if the CloudFormation stack gets deleted.
  + To keep a resource, specify Retain: (works for any resource/ nested stacks).
* DeletionPolicy = Snapshot:
  + Create a snapshot before deletion.
  + Applicable to resources like EBS Volume, ElastiCache Cluster, RDS DB Instances, Redshift, etc.
* DeletionPolicy = Delete (default)
  + NOTE: The default deletion policy for DBCluster resources is ‘snapshot’
  + NOTE: S3 buckets must be emptied before deletion.

To avoid accidental deletes of entire Stack, one can enable ‘*Termination Protection*’ for the entire stack.

## Miscellaneous Features

* CloudFormation Rollbacks
  + During Stack creation
    - Default: Everything rolls back (gets deleted). One can view the log.
    - Option to disable rollback and troubleshoot the problem.
  + During Stack updates
    - The stack automatically rolls back to the previous known working state.
    - Ability to view the log as to what happened via error messages.
* All CloudFormation templates must be placed in S3.
* Exported output names must be unique in a given region.
* One can base user data to launched instances using base64-coded strings.
* Example:

*Resources:*

*MyInstances:*

*Type: AWS::EC2::Instance*

*Properties:*

*AvailabilityxZone: us-east-1a*

*ImageId: ami-009d6802948d06e52*

*InstanceType: t2.micro*

*KeyName: !Ref SSHKey*

*SecurityGroups:*

*- !Ref SSHSecurityGroup*

*UserData:*

***Fn::Base64:*** *|*

*#!/bin/bash -xe*

*sudo yum update -y*

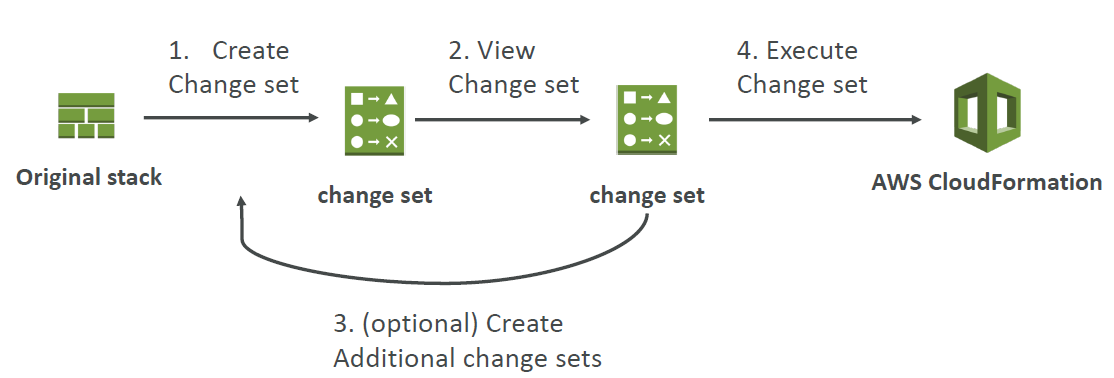
*sudo yum install -y httpd*

*sudo systemctl start httpd.service*

*sudo systemctl enable httpd.service*

*echo "Hello world from user data" > /var/www/html/index.html*

* + The output of the user data goes to ‘*/var/log/cloud-init-output.log’*
  + Alternatively, one can use ‘*cfn-init’* script to make complex EC2 configurations readable.
    - The output of the logs goes to ‘*/var/log/cfn-init.log’*
* One can use the *‘cfn-signal’* script to tell CloudFormation to keep going or fail based on a wait condition (Example: Number of signals sent post successful completion of *‘cfn-init’* script within a specified timeframe)
  + If the wait condition did not receive the required number of signals, then to troubleshoot:
    - Ensure the AMI being used has AWS CloudFormation helper scripts installed.
    - Verify the ‘*cfn-init’* and ‘*cfn-signal’* command was successfully run on the instance.
    - One can do so by viewing their respective logs.
      * NOTE: This requires the rollback on failure configuration to be disabled so that the instance is still up & running.
    - Verify that the instance has a connection to the internet so that it can send the signals to CloudFormation.
* One should isolate repeated patterns/ common components in separate stacks and call them from other stacks as best practice.
  + NOTE: To update nested stacks, always update the root (parent) stack.
* One can make use of ‘Changed Sets’ to determine changes to stack before potentially executing them.



* ‘QuickStart’ is a bunch of CloudFormation templates already pre-built that allows one to create complex environments very quickly.

# Simple Queue Service (SQS)

## Definition

* It is a web service that gives one access to a message queue that can be used to store messages while waiting for a computer to process them.
* It is a distributed system that enables web service applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component.
* A queue is a temporary repository for messages that are awaiting processing.
  + Any component of a distributed application can store messages in a fail-safe queue.
  + Messages can contain up to 256 KB of text in any format.
  + Any component can later retrieve the messages programmatically using the Amazon SQS API.
  + The queue acts as a buffer between the components producing and saving the data, and the component receiving the data for processing.
* It is pull-based, not push-based.

## Use Cases

* Using Amazon SQS, one can decouple the components of an application, so they run independently, easily message management between components.
* This means that the queue resolves issues that arise if the producer is producing work faster than the consumer can process it, or if the producer or consumer are only intermittently connected to the network.
* If the number of messages in the queue exceeds a pre-configured threshold, SQS can also trigger an autoscaling event to handle the queued requests i.e. it scales independently from the applications it is connected to.

## Queue Types

* Standard Queues
  + It is the default queue type.
  + It lets you have a nearly-unlimited number of transactions per second.
  + Standard queues guarantee that a message is delivered at least once.
    - However, occasionally due to the highly-distributed architecture that allows high throughput, more than one copy of the message might be delivered out of order.
    - Standard queues provide best-effort ordering which ensures that messages are generally delivered in the same order as they are sent.
  + Messages can be delivered more than once.
* FIFO Queues
  + Implements first-in-first out delivery with exactly one-time processing.
  + The order in which the messages are sent and received is strictly preserved and a message is delivered once and remains available till the consumer processes and deletes it.
  + Duplicates are not introduced in this queue.
  + Supports ‘Message Groups’ that allow multiple ordered message groups within a single queue.
  + Such queues are limited to 300 transactions per second (TPS) but have all the capabilities of standard queues.
    - Can go up to 3000 TPS if messages are batched.
  + Message-level delay not supported.
  + Ability to do content-based de-duplication at 5-minute intervals.

## Miscellaneous Features

* SQS is AWS’s first service.
* SQS can use long polling to retrieve messages i.e. it doesn’t return a response until a message arrives in the queue or if the long poll times out. (The consumer waits till the long poll times out). In contrast, short polling returns a response immediately to the consumer, even if the queue is empty.
* SQS can point to a message object stored in S3 of up to 2 GB size if message exceeds 256 KB, using the SQS Extended Client.
  + Only the metadata of the large message is stored in queue & polled by consumers, who then proceed to load the large message from S3.
* One can define a threshold for messages that keep getting sent back to SQS by consumers when the visibility timeout occurs. Once this threshold is breached, the messages go to a ‘Dead-Letter Queue’ (which needs to be separately created) where they can be debugged.
* One can define a waiting period that delays a message from being visible to consumers when it first arrives in the queue.
* There are no VPC endpoints for SQS; internet access must be enabled.
* SQS uses HTTPS endpoints for in-transit encryption and SSE-KMS for encryption at rest.
  + The metadata of an SQS message is never encrypted.
* Retention period
  + It is the amount of time the message can be stored in the queue.
* Visibility timeout
  + It is the amount of time that the message is invisible in the SQS queue after a reader picks up that message.
  + If the message is processed before the visibility timeout expires, the message is deleted from the queue by the consumer application.
  + If the message is not processed within that time, the message will become visible again and another reader will process it. This could result in the same message being delivered twice.
* Some key metrics:
  + Visibility timeout
    - Default 30 seconds
    - Minimum 00 seconds
    - Maximum 12 hours
  + Retention period
    - Default 04 days
    - Minimum 01 minute
    - Maximum 14 days
  + DelaySeconds parameter
    - Default 0 seconds
    - Maximum 15 minutes
* Important API calls:
  + ChangeMessageVisibility
    - Changes the visibility of a message while it is being processed.
    - Can be used if application determines it needs a little more time before it completes its processing.
  + DeleteMessage
    - API to tell the SQS the message was successfully processed.
  + WaitTimeSeconds
    - API to enable long polling up to 20 seconds.
  + There are batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility to help reduce costs.

# Simple Workflow Service (SWF)

## Definition

* It is a web service that makes it easy to coordinate work across distributed application components.
* It enables applications for a range of use cases, including media processing, web application back-ends, business process workflows, and analytics pipelines, to be designed as a coordination of tasks.
  + Tasks represent invocation of various processing steps in an application which can be performed by executable code, web service calls, human actions, and scripts.

## Actors

* Workflow Starter
  + An application that can initiate (start) a workflow.
* Deciders
  + Controls the flow of activity tasks in a workflow execution.
  + If something has finished (or failed) in a workflow, a ‘Decider’ decides what to do next.
* Activity Workers
  + Carry out the activity tasks.

## SQF vs SQS

* SQS has a retention period of up to 14 days whereas workflow executions can take up to 1 year with SWF.
* SWF presents a task-oriented API, whereas SQS offers a message-oriented API.
* SWF ensures that a task is assigned only once and is never duplicated. With SQS, duplication messages need to be handled separately and also may need to ensure that a message is processed only once.
* SWF keeps track of all tasks and events in an application. With SQS, one needs to implement their own application-level tracking, especially if the application uses multiple queues.

# Elastic Beanstalk

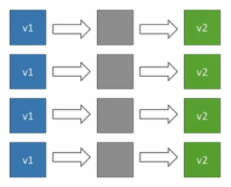
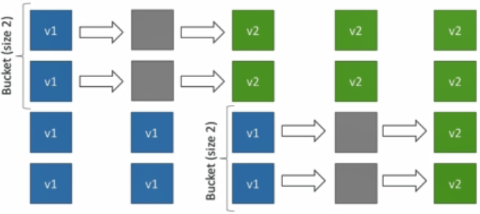
## Definition

* One can quickly deploy and manage applications in the cloud without worrying about the infrastructure that runs those applications.
* Unlike CloudFormation, it is designed as a container for a single application.
* It automatically handles capacity provisioning, load balancing, scaling, and application health monitoring of EC2 instances.

## Key Concepts

* Application
  + A logical collection of Elastic Beanstalk components, including environments, versions and environment configurations.
* Application version
  + Refers to a specific, labeled iteration of deployable code for a web application.
  + Points to an S3 object that contains the deployable code.
* Environment
  + It is a version deployed on AWS resources.
  + An environment runs a single application version at a time, but same or different application versions can be deployed across multiple environments simultaneously.
  + Environment Tier
    - Determines whether Elastic Beanstalk provisions resources for an application that handles HTTP requests (web server environment) or an application that pulls tasks from a queue (worker environment).
  + Environment Configurations
    - Identifies a collection of parameters and settings that define how an environment and its associated resources behave.
    - Changes to environment configurations are automatically applied to existing resources.

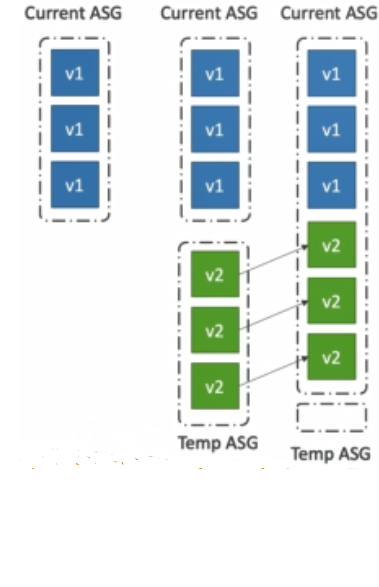
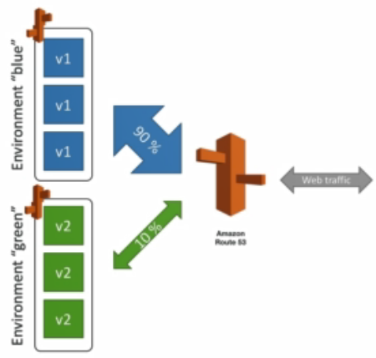
## Deployment Modes

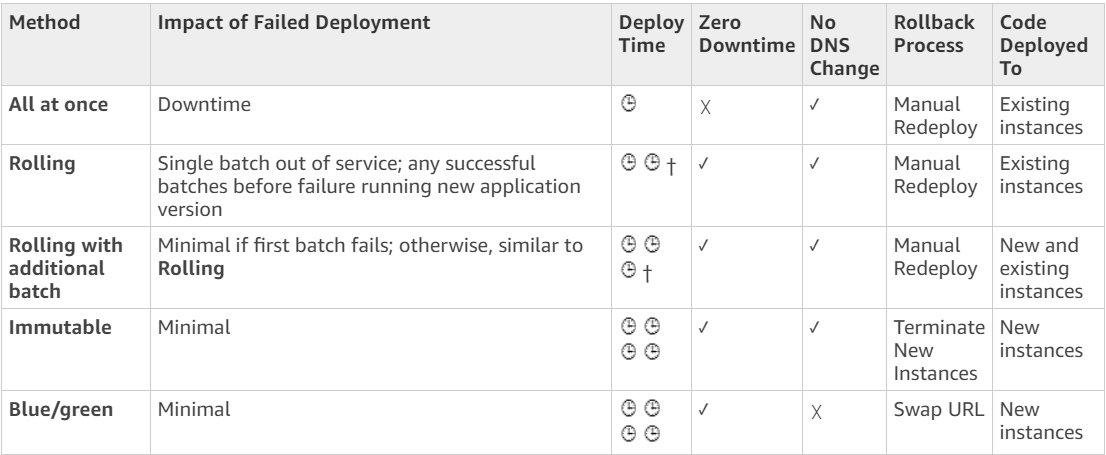
* All at Once
  + Fastest deployment.
  + Application has downtime.
  + Great for quick iterations in development environments.
  + No additional cost.
* Rolling
  + Application running at below capacity.
  + One can set bucket size.
  + Application is running both versions simultaneously.
  + No additional cost.
  + To avoid connection issues when instances are

detached, ELB uses connection draining.

* + Long deployment.
  + If deployment fails mid-way, pending batches

continue to run with the old version.

* Rolling with additional batches
  + Application is running at capacity.
  + One can set bucket size.
  + Application is running both versions simultaneously.
  + Additional batch is removed at the end of deployment.
  + Longer deployment.
  + Good for production.
* Immutable
  + Zero downtime.
  + New code is deployed to instances to new instances in a temporary autoscaling group.
  + High cost, application is at double capacity.
  + Longest deployment.
  + Quick rollback in case of failures (Start with 1 instance in new ASG/Terminate it if it fails)
  + Great for production.
* Blue-Green deployment
  + Not a direct feature of EB.
  + Zero downtime and release facility.
  + One creates a new staging environment and deploys the new version there.
  + The new environment (green) can be validated independently and rolled back if issues crop up.
  + Route 53 can be used to setup using weighted policies to redirect a small portion of traffic to new environment.
  + Using Beanstalk, one can swap URLs when done with the environment testing.
  + Best suited when new version of the application is not compatible with the old version.



## Advanced Concepts

* Under the hood, Elastic Beanstalk uses CloudFormation.
* The application is deployed using a .zip file.
* The *‘.ebextensions/’* directory is location in the root of the source code.
  + Contains all resources as part of the environment.
  + It can be used to add more sources such as RDS, DynamoDB, ElasticCache, etc.
  + The format of the files within the directory must be in YAML/JSON and end with a .config extension.
  + These resources get deleted if the environment goes away.
* One can use an additional CLI called “EB CLI” which has basic commands like ‘eb create’, ‘eb health’, ‘eb deploy’, etc. It is helpful in automating deployment pipelines.
* Dependencies are described when deploying an application such as packages.
  + Each EC2 machine resolves the dependencies, which can be slow.
  + One can package the resolved dependencies with the source code to improve deployment performance speed.
  + One can also use a ‘Golden’ AMI, i.e. an AMI with all the dependencies already configured, to launch new instances.

## Troubleshooting

* If the health of the environment changes to red, one can try the following:
  + Review environment events.
  + Pull logs to view recent log file entries.
  + Roll back to a previous, working version of the application.
* When accessing external resources, one should make sure that the security groups are correctly configured.
* In case of command timeouts, one can increase the deployment timeouts.

# Simple Notification Service (SNS)

## Definition

* It is a web service that makes it easy to set up, operate and send notifications from the cloud.
* It provides developers with a highly scalable, flexible and cost-effective capability to publish messages from an application and immediately deliver them to subscriber or other applications.
* It can also deliver notifications by SMS text messages or email to Amazon SQS queues, or to any HTTP/HTTPS endpoint. It can also deliver messages to AWS Lambda.
* It allows one to group multiple recipients using topics.
  + A topic is an access point for allowing multiple recipients to dynamically subscribe for identical copies of the same notification.
* An ‘event producer’ only sends messages to one SNS topic. One topic can support multiple deliveries to multiple endpoint types – SNS takes care of the formatting.
* The list of endpoints that can receive SNS notifications/eligible for subscription are:
  + Any HTTP/HTTPS endpoint
  + Email (JSON or text)
  + Amazon SQS
  + AWS Lambda
  + SMS
* Some services can send data directly to SNS for notifications such as:
  + CloudWatch Alarms
  + CloudWatch Event Rules
  + Auto Scaling Group Notifications.
  + Amazon S3
  + CloudFormation (upon state changes)

## Benefits

* Instantaneous, push-base delivery with no polling element, unlike SQS which is pull-based.
* Simple APIs and easy integration with applications.
* Flexible message delivery over multiple transport protocols.
* Inexpensive pay-as-you-go model with no up-front costs.
* To prevent messages from being lost, all messages published to SNS are stored redundantly across multiple AZs.

## Miscellaneous Features

* As messages pushed by SNS queues must be processed immediately, one can use SNS + SQS fanout pattern to help decouple the architecture and process images at a desired pace.
  + One producer service ends a single message to an SNS topic. The topic then sends the messages to multiple queues, each queue having a set of consumers attached to it.

# Elastic Transcoder

## Definition

* It is a media transcoder service in the cloud.
* It converts media files from their original source format in to different formats that will play on smartphones, tables, PCs, etc.
* In addition to supporting a wide range of input and output formats, resolutions, bitrates, and frame rates, it also offers features for automatic video bit rate optimization, generation of thumbnails, overlay of visual watermarks, caption support, DRM packaging, progressive downloads, encryption and more.
* Provides transcoding presets for popular output formats, eliminating the need to guess which settings work best on particular devices.
* It is billed on the minutes that one transcodes and resolution at which one transcodes.

# API Gateway

## Definition

* It is a fully managed service that makes it easy for developers to build, maintain, monitor, and secure any APIs at any scale.
* It acts as a ‘front door’ for applications to access data, business logic, or functionality from back-end services, such as applications running on EC2, code running on Lambda, or any web application.

## Use Cases

* Expose HTTPS endpoints to RESTful API (HTTP endpoints not supported).
* Serverless-ly connect to services like Lambda & DynamoDB.
* Send each API endpoint to a different target.
* Run efficiently at low cost & scale effortlessly.
* Track and control usage with API key.
* Throttle requests to prevent attacks.
* Connect to CloudWatch to log all requests for monitoring.
* Maintain multiple versions of one’s API.
* Transform and validate requests and responses/ handle security.

## Deployment Stages

* To make changes in API effective, one must deploy the API first.
* Changes are deployed to ‘stages’.
  + Each stage has its own configuration parameters, such as enabling caching or logs.
  + Stages have environment variables known as ‘stage variables’.
    - They can be used in:
      * Lambda function ARN.
      * HTTP endpoint.
      * Parameter passing templates.
    - They are passed to the ‘context’ object in AWS Lambda.
* It is possible to enable ‘Canary’ deployment for any stage (Similar to Blue/Green deployment, but for AWS Lambda / API Gateway).
  + One can choose the percentage of traffic the canary channel receives.
  + Metrics and logs are separate for better monitoring

## API Gateway Caching

* One can enable API caching to cache an endpoint’s response.
* This can reduce the number of calls made to the endpoint and also improve the latency of the requests to the API.
* When caching is enabled for a stage, the API Gateway caches responses from the endpoint for a specified time-to-live (TTL) period, in seconds. Further requests are served from the cache’s response.
* Default TTL is 5 minutes, which can be extended to 1 hour.
* Caches are defined per stage.
* Cache capacity varies from 0.5 GB to 237 GB.

## Cross-Origin Resource Sharing (CORS)

* Under same-origin policy, a web browser permits scripts contained in first web page to access data in a second web page only if both web pages have the same origin.
  + This is enforced by browsers to prevent Cross-Site Scripting (XSS) attacks.
* This policy is a hindrance for AWS Cloud as different services have different domain names.
* CORS is one way the server at the other end (not the client code in the browser) can relax the same-origin policy.
* It is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was shared.
* If one is using JavaScript/AJAX that uses multiple domains with API Gateway, it must be ensured that CORS has been enabled on API Gateway.
  + Otherwise, an error message “Origin policy cannot be read at the remote resource’ might be displayed.

## Security

API Gateway handles security in primarily three ways:

* IAM
  + Great for users/roles already within the AWS account.
  + Handles authentication + authorization.
  + Leverages ‘Sig v4’ capability where IAM credentials are in the header.
* Custom Authorizer / Lambda Authorizer
  + Great for OAuth/SAML/third party type of authentication.
  + It uses Lambda to return an IAM policy post validating the 3rd party token.
    - Very flexible in terms of what IAM policy is returned.
  + Handles authentication + authorization.
  + One must pay per Lambda invocation.
    - One can cache the results of the authentication to save on invocations.
* Cognito User Pool
  + One must manage their own user pool (can be backed by Facebook, Google login, etc.)
  + There is no need to write custom code.
  + One must implement authorization in the backend as Cognito only performs authentication.

## Cost Model

* There is no minimum fee or startup cost.
* As one can run the APIs without any servers, one only pays for the API calls received and the amount of data transferred out.

## Miscellaneous Features

* Mapping templates can be used to modify request/response.
  + Renaming parameters.
  + Modifying body content.
  + Mapping JSON to XML for sending to backend or back to client.
  + Filter output results by removing unnecessary data.
  + It uses Velocity Template Language (VTL).
    - One can have FOR loops, IF-ELSE statements, etc.
* Just like CloudFormation, a common way of defining REST APIs (i.e. API definition as code) is by importing existing Swagger / OpenAPI 3.0 spec to API Gateway.
  + Swagger can be written in YAML or JSON.
  + One can generate SDK for one’s applications using Swagger.
* One can generate API keys which can be associated with usage plans and track the usage of said keys.
  + Usage plan includes throttling limits (overall capacity and burst capacity) and quotes (number of requests per day/week/month).

# Kinesis

## Definition

* It is a platform to send streaming data to.
* It makes it easy to load and analyze streaming data, and also providing the ability for one to build one’s own custom applications as per business requirements.
* Streaming data is data generated continuously by thousands of data sources, which typically send the data simultaneously, and in small sizes (order of Kilobytes).
  + Purchase from online stores
  + Stock prices
  + Game data
  + Social network data
  + Geospatial data
  + IOT sensor data

## Kinesis Types

* Kinesis Streams
  + Data produced by producers is persisted temporarily in the form of shards in buffers before being processed by consumers.
    - A shard is a uniquely identified sequence of data records in a stream.
    - 5 transactions per second per shard for reads, up to a maximum total data read rate of 2MB per second.
    - 1000 records per second per shard for writes, up to a maximum total data write of 1 MB per second (including partition keys).
    - Number of shards required can be reduced by:
      * Batch-processing small records (1000 records/second limitation)
      * Compressing the data (1 MB/second bandwidth limitation)
    - A stream is composed of one or more shards, each of which provides a fixed unit of capacity. The total capacity of the stream is the sum of capacities of the shard.
  + The data can be persisted anywhere from 24 hours (default) to up to 7 days.
  + Post analysis, the data can be stored in various storage services like S3, DynamoDB, Redshift, EMR, etc.
  + Does not scale automatically and resources needs to be provisioned.
* Kinesis Firehose
  + Data produced by producers is not persisted temporarily.
  + The data is processed as soon as it streams in/if at all. Post the analysis, it is stored into S3/Redshift/ElasticSearch/Splunk using a managed consumer service.
  + Scales automatically.
* Kinesis Analytics
  + Works in conjunction with other Kinesis types to analyze the data on-the-fly via SQL and then store the data.

## Consumer Models

* Standard Consumer Model
  + Each consumer shares the same streaming pipe and thus it’s limit of 5 transactions per second and 2MB data transfer per second.
  + This model is pull-based, as each consumer can poll the stream every 200 ms at best (One transaction per 200 ms = 5 transactions per second).
  + This model should be used when:
    - The number of consumers is low.
    - The consumers are not latency-sensitive.
    - When costs must be kept low.
* Fan-Out Consumer Model
  + Each consumer has its own pipe and hence does not share service limits with other consumers.
  + This model is push-based, i.e. consumers can subscribe to shards. The messages are pushed to consumers by the service over a long running HTTP2 request made by the consumer of a subscribe to shard API call. This leads to sub 70 ms latency.
  + This model should be used when:
    - The number of consumers is large (Default soft limit is 5).
    - The latency needs to be kept minimum.

## Miscellaneous Features

* Once data is inserted in Kinesis, it can’t be deleted until it expires (data is immutable).
* Billing is based on
  + Number of provisioned shared for Kinesis Streams.
    - Batching stream records or compressing data can be done to reduce shard requirement.
  + Amount of data streamed for Kinesis Firehose.
* To avoid a ‘hot’ shard, i.e. when producers send all the data to the same shard due to the dominance of a single partition key, one should choose a more distributed partition key.
* The exception *‘ProvisionedThroughputExceeded*’ can appear when one sends more data then provisioned i.e. either the TPS or MB/s is being exceeded.
  + One should make sure there is no ‘hot’ shard.
  + Solutions include retries with back off, increasing the number of shards or ensuring the partition key distributes incoming data properly.

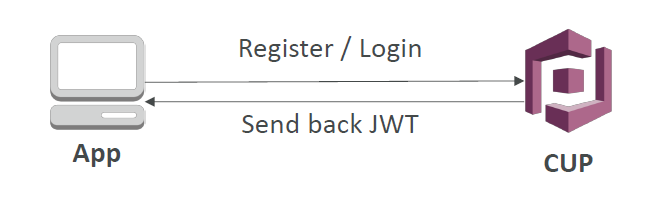
# Cognito

## Definition

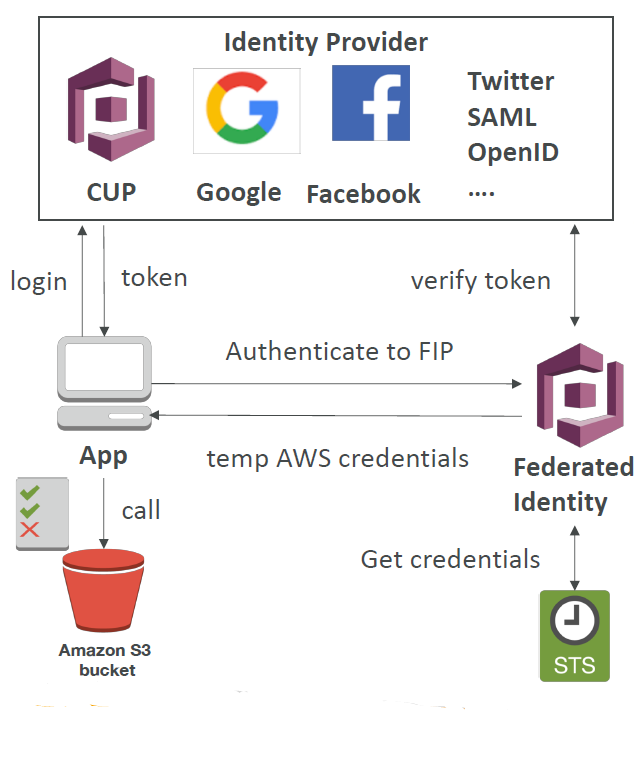
* It is a service that brokers between an application and a web-based identity provider to provide temporary credentials which map to an IAM role allowing access to the required resources.
  + Web Identify Federation lets one give their uses access to AWS resources after they have been successfully authenticated with a web-based identity provided like Google, Facebook, or Amazon.
  + Following successful authentication, the user receives an authentication code from the Web ID provider, which they can trade for temporary AWS security credentials.
* There is no need for the application to embed or store AWS credentials locally on the device it gives users a seamless experience across all mobile devices.
* Cognito tracks the associated between user identity and the various different devices they sign-in from.
  + It uses a push synchronization to push updates and synchronize user data across multiple devices.
  + It uses SNS to send a silent notification to all the associated devices for a given user identity whenever data stored in the cloud changes.

## Pool types

* User Pools
  + They are serverless databases/user directories used to manage sign-up and sign-in functionally for mobile and web applications.
    - Simple login: Username (or email) with passwords.
  + Users can sign-in directly to the User Pool, or using Facebook, Google, etc. Successful authentication generates a JSON Web Token (JWTs).
  + They are user-based. Everything from user registration, authentication, and account recovery is handled.
    - MFA can be added to a user pool as a second authentication method.



* Identity Pools
  + They provide temporary AWS credentials to access AWS services like S3 or DynamoDB.
  + They can be integrated with user pools as an Identity Provider.



## Cognito Sync

* Deprecated to AWS AppSync.
* Used to store preferences, configurations, state of the application.
* One can have cross-device synchronization (any platform – IOS, Android, etc.)
* Offline capability is supported (synchronization when back online)
* Requires Federated Identity Pool in Cognito (not User Pool).
* Store up to 20 datasets to synchronize of size up to 1 MB each.

# Simple Token Service (STS)

## Definition

* Enables users limited and temporary access to AWS resources.
* Users can come from three sources
  + Federation (typical Web Directory)
    - Uses Security Assertion Markup Language (SAML).
    - Grants temporary access based off the users Active Directory credentials. One does not need to be a user in IAM.
    - Single-sign-on allows users to log in to AWS console without assigning IAM credentials.
  + Federation with Mobile apps
    - Uses Facebook / Amazon / Google or other OpenID-compatible providers to log in.
  + Cross Account Access
    - Let’s users from one AWS account access resources in another.

## How it works

* Employees enter their username and password to say, a reporting application.
* The application calls an Identity Broker (client-side). The broker captures the username and password.
* The Identity Broker uses the organization’s Lightweight Directory Access Protocol (LDAP) directory to validate the employee’s identity.
* The Identity Broker calls the GetFederationToken function using IAM credentials. The call must include an IAM policy and a duration (1 to 36 hours), along with a policy that specifies the permissions to be granted to the temporary security credentials.
* The STS confirms that the policy of the IAM user making the call to GetFederationToken gives permission to create new tokens and then returns four values to the application: An access key, a secret access key, a token and a duration (the token’s lifetime).
* The Identity Broker returns the temporary security credentials to the reporting application.
* The data storage application uses the temporary credentials (including the token) to make requests to say Amazon S3.
* Amazon S3 uses IAM to verify that the credentials allow the requested operation on the given S3 bucket and key.
* IAM provides S3 with the go-ahead to perform the requested operation.

# Lambda

## Definition

* It is a serverless compute service where one can upload their code and create a Lambda function.
* The Lambda function takes care of provisioning and managing the servers that the code runs in; hence the end users do not need to worry about operating systems, patching, scaling, etc.
* It can be used in the following ways:
  + As an event-driver compute service where Lambda runs the code in response to events, such as change in data in Amazon S3 or DynamoDB.
  + As a compute service to run code in response to HTTP requests using Amazon API Gateway or API calls made using AWS SDK.
* Languages supported include Node.js, Java, Python, C#, Go, Ruby and PowerShell.

## Lambda Configurations

* Lambda functions can be configured to run up to 15 minutes.
  + The default time-out value is 3 seconds.
  + This feature is useful to limit run-times in case of erroneous code.
* Environment variables can be passed to Lambda functions to make the code more dynamic.
  + Lambda encrypts them using the AWS Key Management Service.
  + When the Lambda function is invoked, those values are decrypted and made available to the Lambda code.
* The memory, and by extension CPU, can be configured from 128 MB (base) to up to 3008 MB with 64 MB increments.
* The Lambda function can be deployed within a VPC subnet if chosen to. Appropriate security groups can thus be applied.
* The number of invocations for the same functional is determined by the concurrency parameter.
* Concurrency can be reserved from anywhere from 0 to 1000. (Soft limit).
  + The maximum unreserved concurrency limit is 1000 invocations.
  + If the number of invocations exceeds the concurrency threshold, then:
    - For synchronous invocations, ThrottleError – 429 is thrown.
    - For asynchronous invocations, post up to two retires, all unprocessed events are sent to a configured Dead Letter Queue (DLQ).
      * The DLQ can be a SNS topic or SQS queue.
      * Relevant IAM permissions have to be attached to the Lambda function to use DLQ.
* For enhanced debugging, one can enable active tracing to integrate the Lambda function with AWS X-Ray.
  + This also adds relevant IAM permissions for using X-Ray to the function role.
* Tags can be associated with any Lambda function.
  + Useful for billing.

## Service Limits

* A Lambda function can run for up to 15 minutes.
* Memory allocation is in 64 MB increments, starting from 128 MB to 3008 MB.
* One can have disk capacity in the function container (in /tmp folder) of up to 512 MB.
* The concurrency soft limit for a Lambda function is 1000.
* The deployment size for the Lambda function itself must be:
  + Less than 50 MB compressed in .zip.
  + Less than 250 MB uncompressed (Code + Dependencies).
  + One can load other files at startup in the /tmp directory.
* The size of the environment variables must not exceed 4 KB.

## Cost Model

* Billed by the number of requests (per million requests).
* Duration of requests i.e. from the time the code begins to the time it returns or otherwise terminated, rounded to the nearest 100ms.
* Tags can be associated to Lambda functions to help with billing.

## Miscellaneous Features

* Lambda scales out (not up) automatically i.e. multiple invocations trigger multiple Lambda functions.
* Lambda functions are independent, one event equals one function, unless the original function triggers other Lambda functions.
* As architectures can get extremely complicated with Lambda, AWS X-ray can use used to debug the process.
* For version control:
  + When a Lambda function is published, an immutable version is created with its own ARN. No changes can be made to them.
  + The version where one can modify the function is known as $LATEST version.
  + One can also create aliases which can point to an already-created version of a Lambda code.
    - Alias are mutable i.e. the version they are pointing to can be modified.
    - Each alias has its own ARN.
    - One can have an alias point to two different versions. Each version gets a configurable percentage of traffic.
      * This is useful for Blue/Green deployment of new updates.
* Some of AWS Lambda best practices include:
  + Perform heavy-duty work outside the function handler.
    - This includes connecting to a database, initializing AWS SDK, pulling dependencies, etc.
  + Use environment variables for:
    - Database connection strings, S3 buckets, etc.
    - Passwords and other sensitive values should always be used as environment variables encrypted with KMS.
  + Minimize deployment package to bare minimum.
  + Break down the function if need be.
  + One should not put the Lambda function in a VPC unless required.
    - An ENI needs to be attached to the Lambda function and an IP from the subnet CIDR block needs to be assigned to it, increasing the startup delay.
    - Each invocation needs a separate ENI. One might run out of IP addresses in the subnet.

# AWS X-Ray

## Definition

* It is a monitoring service that can provide one a graphical end-to-end view of how requests flow through an application pipeline.
* It leverages tracing i.e. it follows a request end to end, with each component dealing with the request leaving behind a trace.
* Tracing is made of segments and sub-segments.
* It is compatible with a large number of services such as Lambda, Elastic Beanstalk, ECS, ELB, API Gateway, EC2 instances or any application server (even on-premise), etc.

## Use Cases

* To troubleshoot performance & check for throttling.
* To understand dependencies in a microservice architecture.
* To pinpoint service issues & determine if SLAs have been met.
* To find errors and exceptions
* To review request behavior.
* To identify the users that are impacted.

## How it works

* The code must import the AWS X-Ray SDK.
* The X-Ray daemon or enable X-Ray integration (It is already running in compatible services).
* Each application must have IAM rights to write data to X-Ray.
* X-Ray service collects data from all the different services.
* The service map is computed from all the segments and traces.

## Security

* It uses IAM for authorization.
* It uses KMS for encryption at rest.

# Elastic Container Service (ECS)

## Definition

* It is a highly scalable, high performance container management service that supports Docker containers and allows one to easily run applications on a managed cluster of Amazon EC2 instance, eliminating the need to install, operate, and scale one’s own cluster management infrastructure.
* ECS clusters are a logical grouping of EC2 instances that run the ECS agent (Docker container) that registers the instances to the ECS cluster.
  + The EC2 instances run a special AMI, made specifically for ECS.

## Task Definition

* Task definitions specify the container information for one’s application, such as
  + how many containers are part of the task
  + what resources they will use
  + how they are linked together, and
  + which host ports they will use.
* Each task definition is either ECS-Classic compatible or AWS Fargate compatible.
* One can define IAM roles at the task level to execution actions against other AWS services.
* Security groups cannot be attached to tasks. They are defined at instance level.

## ECS Service

* Services help define how many tasks should run and how they should be run.
* They ensure that the number of tasks desired is running across our fleet of EC2 instances.
* They are defined at the ECS-Cluster level.
* They can be linked to an ELB (Elastic Load Balancer) if needed.
* Services can be of two types:
  + Replica
    - Places and maintains a desired number of tasks across the cluster.
  + Daemon
    - Places one copy of the task on each container service.
    - Useful if it’s just a monitoring/metric gathering task.

## Elastic Container Registry (ECR)

* It is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
* It is tightly integrated with IAM.
* The following are the series of steps to push a custom image onto ECR for :
  + For AWS Account: 115312177287; Region: us-east-1; Repository name: demo; Tag: latest …
  + Authenticate docket client to ECR
    - $(aws ecr get-login --no-include-email --region us-east-1)
  + Build Docker image
    - docker build -t demo .
  + Tag the image
    - docker tag demo:latest 115312177287.dkr.ecr.us-east-1.amazonaws.com/demo:latest
  + Push the image
    - docker push 115312177287.dkr.ecr.us-east-1.amazonaws.com/demo:latest

## Miscellaneous Features

* AWS Fargate is the serverless variant of Classic ECS wherein one need not manage EC2 instances. The infrastructure is handled by AWS.
* The ECS cluster name(s) must be configured in the ‘*/etc/ecs/ecs.config*’ file for each ECS instance running the ECS agent.
* EC2 instances can run multiple containers of the same type.
  + One must not specify a host port (only container port).
  + One should use Application Load Balancer with dynamic port mapping.
    - The EC2 instance security group must allow traffic from the ALB on all ports.

# CodeCommit

## Definition

* Managed code repository hosted on AWS Cloud.
* Interactions are done via standard Git.
* Integrated with build services like CodeBuild, Jenkins, etc.
* Benefits include:
  + Version control, enabling one to understand changes to one’s codebase over time.
  + Possibly rolling-back to an earlier version.
  + Collaborating with other developers.
  + Backing up the code.
  + Making sure the code is fully viewable and auditable.

## Security

* Authentication
  + SSH Keys: AWS users can configure SSH keys in their IAM Console.
  + HTTPS: Done through AWS CLI configuration helper or generating HTTPS credentials.
  + MFA can be enabled for extra safety.
* Authorization
  + IAM policies manage user/roles rights to repositories.
* Encryption
  + Repositories are automatically encrypted at rest using KMS.
  + For transit, only HTTPS or SSH can be used – both secure.
* Cross Account Access
  + One can use IAM Role & STS.

## Notifications

* One can trigger notifications in using AWS SNS, Lambda or CloudWatch Event Rules.
* Use cases for SNS/Lambda notifications.
  + Deletion of branches.
  + Trigger for pushes that happens in a branch.
  + Notify external build system.
  + Trigger Lambda to perform codebase analysis.
    - Among others, one possible reason to do this could be to check if the code contains any credentials.
  + Use cases for CloudWatch Event Rules (which goes into a SNS Topic)
    - Trigger for pull request updates (created/updated/deleted/commented)
    - Commit comment events (When comments are made on a commit)

# CodePipeline

## Definition

* It is a continuous delivery service one can use to model, visualize, and automate the steps required to release one’s software.
* It compromises of stages that can have sequential or parallel actions, such as Build/Test/Deploy/etc.
* Manual approval can be defined at any stage.
* Each pipeline stage creates ‘artifacts’ which are stored in S3 & passed to the next stages.
* State changes can appear in CloudWatch Events, which can in turn be used to create SNS Notifications.
  + One can create SNS notifications for failed pipelines/cancelled stages.

# CodeBuild

## Definition

* It is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy.
* It has continuous scaling. There are no servers to manage or provision/no build queue.
* It leverages Docker under the hood for reproducible builds.
* Build instructions is defined in the root directory of the code in the *buildpsec.yml* file.
* CodeBuild supports the following environments: Java, Ruby, Python, Go, Node.js, Android, .NET Core, PHP. Other environments can be extended using Docker.

## Security

* KMS is used to encrypt build artifacts.
* IAM is used for build permissions.
* VPC for network security.
* CloudTrail for API calls logging.

## Troubleshooting

* One can output logs to S3 & AWS CloudWatch Logs.
* CloudWatch metrics can be used to monitor CodeBuild statistics and alarms can be set to detect failed builds and send notifications.
* One can re-produce CodeBuild locally to troubleshoot in case of errors.

# CodeDeploy

## Definition

* It is a managed deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.
* It can be integrated with CodePipeline and use artifacts from there.
* It does not provision resources.

## Primary Components

* Application: Unique name
* Compute platform: EC2/On-Premise or Lambda
* Deployment configuration: Deployment rules for success / failures
  + EC2/On-Premise: E.g. Specify the minimum number of healthy instances for the deployment.
  + AWS Lambda: E.g. Specify how traffic is routed to updated Lambda function versions.
* Deployment group: Group of tagged instances (allows to deploy gradually)
* Deployment type: In-place deployment or Blue/green deployment:
* IAM instance profile: Provide EC2 the permissions to pull from S3 / GitHub
* Application Revision: Application code + *appspec.yml* file
* Service role: Role for CodeDeploy to perform what it needs
* Target revision: Target deployment application version

## How it works

* EC2 instances are grouped by ‘deployment’ groups with auto-scaling integration.
* Each EC2 (or on-premise) machine runs a CodeDeploy agent which continuously polls CodeDeploy for work.
* CodeDeploy will send the *appspec.yml* file. (This file must be located at the root of the source code).
* Application is pulled from GitHub/S3.
* EC2 will run the deployment instructions.
* CodeDeploy agent will report the success/failure status of the deployment.

## AppSpec File

* File section
  + How to source and copy from S3/GitHub to filesystem.
* Hooks
  + Set of instructions to carry out the deployment to the new version.
  + Hooks can have timeouts.
  + The order of hooks is:
    - ApplicationStop
    - DownloadBundle
    - BeforeInstall
    - AfterInstall
    - ApplicationStart
    - ValidateService

## Deployment Modes

* All-at-once
  + The application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated.
    - One-at-a-time
    - Half-at-a-time
    - All-at-once
* Blue-Green deployment
  + Instances are provisioned in a replacement environment with new version of the application.
  + Blue/Green deployment only works with EC2 instances (not with on-premise instances).

# AWS Simple Systems Manager (SSM)

## Definition

* It is a service that enables one to manage EC2 or on-premise servers at scale by providing operation insights about the state of one’s infrastructure, detect problems and patch automations.
* One registers their servers with the service through an SSM agent. (SSM agent comes pre-installed with AWS Linux/Ubuntu AMIs)

## Run Command

* Allows one to execute a document/ run a script across one or more instances/ resource groups.
* One does not require SSH access.
* Integrated with IAM & CloudTrail.

## Inventory

* Lists out all the software installed on an instance.
* Inventory uses *‘State Manager’* under the hood.

## Patch Manager

* Provides compliance by patching installed software(s) on selective instances/tags/resource groups.
* One can set fixed maintenance window under which patching operations can be run.
* Patch manager uses *‘Run command’* under the hood.

## State Manager

* Ensures that all instances are in consistent state for compliance.
* One can schedule the state manager to provide updates to Inventory at fixed intervals.

## Session Manager

* Allows one to start a secure shell on one’s VM without SSH access/ bastion hosts. Instead, IAM is used.
* Log actions done through CloudWatch Logs or S3.
* One can keep track of ‘*StartSession*’ events via CloudTrail.

## Parameter Store

* It is used for secrets and configuration management.
* Allows one to securely store configurations (encrypted using KMS, or otherwise).
* It is serverless, scalable, durable and easy to use.
* It has version-control built in.
* Parameters are stored in the form of a tree hierarchy and can be accessed via the ‘*GetParameters*’ or ‘*GetParametersByPath*’ API, using appropriate IAM permissions.

# Miscellaneous Services

## AWS Key Management Service (KMS)

* It is a managed service that by provides a highly available key storage, management (rotation), and auditing solution for one to encrypt data within their own applications and control the encryption of stored data across AWS services.
* Encryption is done via a Customer Master Key (CMK) that can never be retrieved by the user.
  + AWS Managed Service Default CMK
  + User Keys created by KMS
  + User Keys imported (must be 256-bit symmetric keys)
* KMS can only help in encrypting up to 4 KB per call.
  + If data is > 4 KB, one can use envelope encryption/ GenerateDataKey API.
* One can audit key usage via CloudTrail.
* For some services, using KMS requires migration (snapshot/backup), such as EBS volumes, RDS databases, ElastiCache, EFS, etc.
* KMS supports only symmetric keys.

## AWS Config

* It is a monitoring service that enables auditing and compliance of one’s AWS resources.
* It can record configurations of specified resources & it’s changes over a period of time.
  + These configurations can be aggregated over regions and accounts.
  + Any changes to configurations can be written to S3 or streamed to SNS topics.
* It can record compliance over time with the help of AWS managed or custom configuration rules.
  + Any configuration that fails the rule is specified as non-compliant.
  + Examples:
    - Check if any security group allows unrestricted SSH access.
    - Check if any ELB is missing a SSL certificate.

## AWS Organizations

* It is an account management service that enables one to consolidate multiple AWS accounts into a single organization account that is centrally managed.
  + There is a single root account with one or more ‘Organizational Units’ (OU) below.
  + Each unit can have further units i.e. there is a hierarchy.
  + There can be one or many AWS accounts associated with each OU.
  + Service Control Policies (SCPs) can be applied by the root account at the OU/account level as required.
    - These policies can help restrict accounts in the OU to access specific AWS services or specific APIs within these services.
    - Nested OUs inherit SCPs from higher tier OUs.
* It enables consolidated billing i.e. the resource costs associated with individual accounts are billed in a single account i.e. a paying account.
  + The individual accounts are known as linked accounts. There is a soft limit of 20 for one paying account.
  + The paying account cannot access resources of the other accounts.
  + This mode of billing reduces costs by utilizing cross-account volume discounts.
    - For example, discount from an unused reserved instance in one account can be applied to another linked account.
  + Billing alerts can be set for the final consolidated bill, or at the linked account level.
* Since CloudTrail service operates at per account per region level, one can aggregate the logs by turning on CloudTrail in the paying account and creating a S3 bucket with a bucket policy that allows cross account access, with the linked account’s CloudTrail pointing to it.

## AWS Cost Explorer

* It is a graphical tool that allows one to view and analyze costs and usage.
* One can forecast expected costs for next 3 months.
* One can receive recommendations on which reserved instances to purchase and potential savings.
* One has access to default reports and to build custom cost management applications.
* One can use ‘Cost Allocation Tags’ to active certain tags which are attached to AWS resources. This enables one to slice cost explorer reports by the tag value.
  + Example: Adding tag ‘ Environment’ can allow one to explore tags at environment level.

## AWS Budgets

* It is a cost management service wherein one can create budgets and send alarms when the cost exceeds the budget or when the cost is forecasted to exceed the budget.
  + The ‘budget’ can be defined at a very granular level, such as for 10$ monthly budget *‘t2.micro’* EC2 instance in *‘us-east-1’* region.
  + There are three types of budgets on can set:
  + Cost Budget
    - Track budget in terms of user-defined cost limits.
  + Usage Budget
    - Track budget in terms of user-defined resource limits.
    - Example: Monthly EC2 Data Transfer < 1 TB.
  + Reservation Budget
    - Track RI Utilization.

## AWS Certificate Manager

* Security service that lets one easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and one’s internal connected resources.
* One can quickly request a certificate, deploy it on ACM-integrated AWS resources, such as Elastic Load Balancers, Amazon CloudFront distributions, and APIs on API Gateway, and let AWS Certificate Manager handle certificate renewals.
* It also enables one to create private certificates for your internal resources and manage the certificate lifecycle centrally.

## AWS Shield

* It is a security-based service that protects from Distributed Denial-of-Service (DDOS) attacks.
* There are two variants to AWS Shield:
  + Shield Standard
    - It is free and is activated for every user.
    - Protects against attacks such as SYN/UPD floods, reflection attacks and other layer 3 / 4 attacks.
  + Shield Advanced
    - Optional premium service (Can cost $3000 per month for an organization).
    - Provides protection against more sophisticated attacks on CloudFront, Route 53, Elastic IP, Load balancers (for select regions), etc.
    - Provides 24/7 access to DDOS response team (DRP).
    - Hike in costs due to DDOS attacks are waived off.

## AWS Web Application Firewall (WAF)

* It is a security-based service that protects web applications from common exploits such as SQL-injections and
* One can define customizable web security rules such as:
  + Control which traffic to allow or block for one’s web application. Rules can be based on:
    - IP addresses
    - HTTP headers/body.
  + Impose constraints on the size of requests/ the geographical location from which the requests originate.
* One can deploy WAF on CloudFront, Application Load Balancer or API Gateway.
* One can leverage existing marketplace of web security rules

## AWS Inspector

* It is a security-based service, specifically for EC2 instances.
* Upon installation of Inspector agent on OS in EC2 instances, one can:
  + Analyze against known vulnerabilities.
  + Analyze against unintended network accessibility.
* One would need to define a template, consisting of rule packages, duration, attributes, etc.
  + Custom rules cannot be defined for Inspector and hence one must use AWS-managed rules.
* One can schedule to run the inspection at fixed windows, post which a complete report on the security assessment is obtained.

## AWS Trusted Advisor

* It is a tool analyzes one’s AWS environment and provides best practice recommendations in these five categories: Cost Optimization, Performance, Fault Tolerance, Security, and Service Limits.
* Full set of Trusted Advisor checks are available only to Business and Enterprise plans.

## AWS GuardDuty

* It is a security-based service that provides intelligent threat discovery to protect one’s AWS account.
* It utilizes machine learning algorithms for anomaly detection, etc.
* It uses log data, including VPC Flow logs, CloudTrail logs, as well as DNS logs
* It automatically notifies you in case of any findings.

## AWS CloudHSM

* It is a cloud-based security service that enables one to meet corporate, contractual, and regulatory compliance requirements for data security by using dedicated Hardware Security Module (HSM) instances within the AWS cloud.
  + HSMs provides secure key storage and cryptographic operations within a tamper-resistant hardware device.
  + HSMs are designed to securely store cryptographic key material and use the key material without exposing it outside the cryptographic boundary of the hardware.
* One should consider using AWS CloudHSM over AWS KMS if one requires:
  + Keys stored in dedicated, third-party validated hardware security modules under one’s exclusive control.
  + FIPS 140-2 compliance.
  + Integration with applications using PKCS#11, Java JCE, or Microsoft CNG interfaces.
  + High-performance in-VPC cryptographic acceleration (bulk crypto).
* CloudHSM clusters are spread across multiple AZs.
* CloudHSM supports both symmetric as well as asymmetric keys (SSL/TLS keys).

## AWS Directory Service

* It is a service that enables one to integrate their on-premise Active Directory with AWS.

## AWS Step Functions

* It is an orchestration service that enables one coordinate multiple AWS services into serverless workflows so that one can build and update applications quickly.
* Workflows are made up of a series of steps, with the output of one step acting as input into the next.
* While both Step Functions and SWF support task execution via Lambda, coordination between stages i.e. the orchestration logic is handled in SWF via one or more ‘Deciders’, which can’t execute via Lambda. In Step Functions, the orchestration logic is governed by ‘State Definition’ i.e. the coordination logic is executed by the service itself, making the process completely serverless.
* One can monitor each step as it executes via DAG visualization.
* One can catch exceptions within each state and failover to known states.

## AWS OpsWorks

* It is a configuration management service that provides managed instances of Chef and Puppet.
  + Chef and Puppet are automation platforms that allows one to use code to automate the configurations of one’s servers.
* It lets one use Chef and Puppet to automate how servers are configured, deployed, and managed across your Amazon EC2 instances or on-premises compute environments.
* Chef/Puppet are similar to SSM/ Beanstalk/ CloudFormation, but are open-source tools that work cross-cloud.

## AWS IOT Core

* It is a managed cloud service that lets connected devices easily and securely interact with cloud applications and other devices.
* Supports HTTP, WebSockets, and MQTT, a lightweight communication protocol specifically designed to tolerate intermittent connections, minimize the code footprint on devices, and reduce network bandwidth requirements.

## AWS Data Migration Service

* It is a service that enables one to migrate databases to AWS cloud quickly and securely.
* The source database remains fully operational during the migration, minimizing downtime to applications that rely on the database.
* It supports both homogenous as well as heterogeneous migrations
  + For heterogenous migrations that require schema and code transformation, AWS Schema Conversion Tool is used first, followed by the AWS DMS service to move the data, making the migration a two-step process.

## AWS Serverless Application Model

* It is an open source framework for building serverless applications.
* It provides shorthand syntax to express functions, APIs, databases, and event source mappings modeled in the form of a YAML template.
* It uses the ‘*sam package*’ and ‘*sam deploy*’ commands in its CLI to bundle the application code and dependencies into a ‘deployment package’ and then deploy the serverless application on the AWS Cloud.