Regions and AZ

1. **Considerations when choosing regions:**

* **Compliance with data governance and legal requirements**: data never leaves a region without your explicit permission
* **Proximity to customers:** reduced latency
* **Available services within a Region**
* **Pricing:** pricing varies region to region and is transparent in the service pricing page

1. **Regions and Availability Zones**

* Region is a cluster of data centres
* Each region has many availability zones (usually 3, min is 3, max is 6).
* Each availability zone (AZ) is one or more discrete data centres

Identity and Federation

1. **IAM = Identity and Access Management, Global service**

* **Root account** created by default, shouldn’t be used or shared
* **Users** are people within your organization, and can be grouped, for long term credentials
* **Groups** only contain users, not other groups
  + Users don’t have to belong to a group, and user can belong to multiple groups
  + Users or Groups can be assigned JSON documents called policies which define the permissions of the users
  + Least privilege principle: don’t give more permissions than a user needs
* **Roles:** short-term credentials using STS (Security Token Service)
  + E.g. EC2 instance roles, service roles, cross account roles

1. **IAM Policy Structure**

* Consists of
  + **Version**: policy language version, always include “2012 -10 - 17”
  + Id (optional): an identifier for the policy
  + **Statement**: one or more individual statements (required)
* Statements consists of
  + Sid (optional): an identifier for the statement
  + **Effect**: whether the statement allows or denies access (Allow, Deny)
  + **Principal:** account/user/role to which this policy applied to
  + **Action:** list of actions this policy allows or denies
  + **Resource**: list of resources to which the actions applied to
  + Condition (optional): conditions for when this policy is in effect
* Policy Types: AWS managed policies, customer managed policies, inline policies, resource-based policies

1. **Accessing IAM**

* **AWS Management Console (protected by password + MFA)**
* **AWS Command Line Interface (CLI):** protected by access keys
  + A tool that enables you to interact with AWS services using commands in your command-line shell
  + Direct access to the public APIs of AWS services
  + Open source
* **AWS Software Developer Kit (SDK):** 
  + Language-specific APIs (set of libraries)
  + Embedded within your application
* Access Keys (like passwords) are generated through the AWS Console; Users manage their own access keys

1. **IAM roles for Services**

* Some AWS service will need to perform actions on your behalf
* Assign permissions to AWS services with IAM Roles
* Common roles: EC2 Instance Roles, Lambda Function Roles, CloudFormation

1. **IAM roles vs Resource Based Policy**

* Cross account:
  + attaching a resource-based policy to a resource (example: S3 bucket policy)
  + OR using a role as a proxy
* **When you assume a role (user, application or service), you give up your original permissions and take the permissions assigned to the role**
* When using a resource-based policy, the principal doesn’t have to give up his permissions
* Supported by: Amazon S3 buckets, SNS topics, SQS queues etc

1. **AWS Cognito:**

* A service that provides user authentication, authorization, and user management for web and mobile applications.
* **AWS Cognito vs. IAM:**
  + **Cognito**: Focuses on managing application-level users and their identities, with support for federated logins and temporary credentials for AWS access.
  + **IAM**: Focuses on permissions and identity management for AWS services and resources. Users, roles, and policies in IAM are mainly for internal AWS resource access control.

1. **IAM Security Tools**

* **IAM Credentials Report (account-level):** a report that lists all your account's users and the status of their various credentials
  + **Audit Access Keys**: Identify users with active, unused, or compromised access keys. You can rotate or delete keys that haven’t been used recently.
  + **MFA Enforcement**: Review which users have MFA enabled to ensure you're following best security practices.
  + **Inactive Credentials**: Identify inactive credentials (e.g., passwords or access keys that haven’t been used for a long time) to improve security by disabling or removing them.
  + **Credential Rotation**: Helps ensure access keys and passwords are rotated regularly in line with best security practices.
* **IAM Access Advisor (user-level):** Provides insight into the **last access times** of the AWS services used by an IAM user or role. This helps you determine which permissions are actively being used and which are not, allowing you to refine permissions and follow the least-privilege principle.
  + **Last Access Information**: Displays data on when services were last accessed by a particular IAM user or role.
  + **Permission Cleanup**: Helps identify and remove unused permissions by showing you which services the user or role has not accessed in a long time.
  + **Security**: Helps reduce the attack surface by enabling you to apply the principle of least privilege (i.e., granting only the permissions needed for the user or role).
  + **Use Case**: Use IAM Access Advisor to audit IAM permissions and ensure that users/roles have access only to the services they actively use.
* **IAM Access Analyzer**: Helps you identify resources that have been shared publicly or with external AWS accounts. It continuously monitors your AWS environment to ensure that your resources (like S3 buckets, IAM roles, KMS keys, etc.) are not unintentionally exposed to the public or external entities.
  + **Policy Analysis**: Analyzes resource policies (like S3 bucket policies, IAM role policies, etc.) to identify access that extends outside your AWS account.
  + **Findings**: Provides actionable findings if resources are shared with external accounts, helping you secure your environment.
  + **Security**: Helps with continuous monitoring to ensure resource access is only granted to trusted accounts or entities.
  + **Use Case**: You can use IAM Access Analyzer to continuously ensure that no critical resources are shared publicly unless intended

1. **AWS Organizations (Global service)**

* Allows to manage multiple AWS accounts
  + The main account is the management account
  + Other accounts are member accounts
  + Member accounts can only be part of one organization
* Consolidated Billing across all accounts - single payment method
* Economies of scale: Shared reserved instances and Savings Plans discounts across accounts
* API is available to automate AWS account creation
* Advantages
  + Multi Account vs One Account Multi VPC
  + Use tagging standards for billing purposes
  + Enable CloudTrail on all accounts, send logs to central S3 account
  + Send CloudWatch Logs to central logging account
  + Establish Cross Account Roles for Admin purposes

1. **Service Control Policies (SCP)**

* A feature of AWS Organizations that allow you to manage and control the access to AWS services across accounts in your organization
* Define the maximum permissions that can be granted to AWS resources within the accounts in your organization
* Ensures that the organizational units (OUs) or accounts cannot perform actions that are disallowed by the SCP, regardless of their IAM policies.

Key Concepts of SCPs:

* **Effect:** SCPs can either allow or deny access to AWS services and actions. However, unlike IAM policies, SCPs do not grant permissions by themselves. Instead, they define the boundaries of the permissions that can be granted by IAM policies.
* **Inheritance:** SCPs are inherited down the hierarchy. This means that an SCP attached at the root of an organization applies to all OUs and accounts beneath it, unless overridden by a more specific SCP attached at a lower level.
* **Deny Overrules Allow**: SCPs can explicitly deny actions. If an action is denied by an SCP, it cannot be performed, even if an IAM policy allows it. SCPs operate in a way where a "deny" always overrides any allow from lower-level policies or attached IAM permissions.
* **Attached to Accounts and OUs**: SCPs can be attached to individual accounts or organizational units (OUs). When attached to an OU, the policy applies to all accounts in that OU and any child OUs under it.
* **Preventative Control**: SCPs serve as a preventative control, limiting what can be done, but not granting permissions by themselves. Permissions must still be explicitly granted by IAM policies.
* **FullAWSAccess Policy**: By default, AWS Organizations attaches a policy called **FullAWSAccess** to all accounts and OUs, which allows all actions and services. You must modify or attach more restrictive SCPs if you want to limit access.

1. **SCP Hierarchy in AWS Organizations:**

* The SCP hierarchy follows the structure of AWS Organizations, which consists of:
  + Root
  + Organizational Units (OUs)
  + Accounts
* SCPs are applied at each level and inherited down the hierarchy, meaning that policies applied at the top level (root) apply to all OUs and accounts below, while policies applied at the OU level affect only the specific OU and its child accounts.
* Effective Permissions in the Hierarchy:
  + The effective permissions for an account are the result of the union of SCPs applied at the root, any OUs the account is a part of, and the account itself.
  + Even if an IAM policy grants permission for a certain action, if the action is restricted by an SCP higher in the hierarchy, the action cannot be performed.

1. **IAM Permission Boundaries**

* IAM Permission Boundaries are supported for users and roles (not groups)
* Advanced feature to use a managed policy to set the maximum permissions an IAM entity can get.
* Can be used in combinations of AWS Organizations SCP
* Use cases
  + Delegate responsibilities to non-administrators within their permission boundaries, for example create new IAM users
  + Allow developers to self-assign policies and manage their own permissions, while making sure they can’t “escalate” their privileges (= make themselves admin)
  + Useful to restrict one specific user (instead of a whole account using Organizations & SCP)

1. **IAM Policy Evaluation Logic**

* **Start with Default Deny**: Every request is denied by default.
* **Check for Explicit Deny**: If any policy explicitly denies the action, the request is denied, regardless of any allows.
* **Check for Explicit Allow**: If no explicit deny is found, AWS checks for an explicit allow. If found, the request is allowed (unless overridden by a deny).
* **Evaluate Conditions**: AWS evaluates any conditions attached to the allow policy. If all conditions are met, the request is allowed.

**Other Considerations**

* **Identity-Based Policies**: These are policies attached to IAM users, groups, or roles, specifying what actions they can perform on which resources.
* **Resource-Based Policies**: Policies attached to resources such as S3 buckets, Lambda functions, or SNS topics. These allow cross-account access by specifying who can access the resource and what actions are permitted.
* **Permissions Boundaries**: These are limits on the permissions that an IAM user or role can have. If the requested action exceeds the permissions boundary, it is denied even if there is an allow policy.
* **Service Control Policies (SCPs)**: SCPs limit what IAM principals can do in AWS Organizations. Even if an IAM policy allows an action, if the SCP denies it, the request will be denied.
* **Session Policies**: Policies that are attached to a specific session (e.g., for a role) further restrict the actions that the session can perform.

1. **AWS Identity Centre**

* One login (single sign-on) for all your AWS accounts in AWS Organizations
  + Business Cloud Applications
  + SAML2.0-enabled applications
  + EC2 Windows Instances
* Built-in identity store in IAM Identity Center
* **Multi-Account Permissions**
  + Manage access across AWS accounts in your AWS Organization
  + Permission Sets – a collection of one or more IAM Policies assigned to users and groups to define AWS access
* **Application Assignments**
  + SSO access to many SAML 2.0 business applications (Salesforce, Box, Microsoft 365)
  + Provide required URLs, certificates, and metadata
* **Attribute-Based Access Control (ABAC)**
  + Fine-grained permissions based on users’ attributes stored in IAM Identity Center Identity Store
  + Example: cost center, title, locale
  + Use case: Define permissions once, then modify AWS access by changing the attributes

1. **AWS Directory Service – Equivalent of Microsoft Active Directory**

* AWS Managed Microsoft AD
  + Create your own AD in AWS, manage users locally, supports MFA
  + Establish “trust” connections with your on- premises AD
* AD Connector
  + Directory Gateway (proxy) to redirect to on- premises AD, supports MFA
  + Users are managed on the on-premises AD
* Simple AD
  + AD-compatible managed directory on AWS
  + Cannot be joined with on-premises AD

1. **AWS Control Tower**

* Easy way to set up and govern a secure and compliant multi-account AWS environment based on best practices
* AWS Control Tower uses AWS Organizations to create accounts
* Benefits:
  + Automate the set-up of your environment in a few clicks
  + Automate ongoing policy management using guardrails
  + Detect policy violations and remediate them
  + Monitor compliance through an interactive dashboard

1. **AWS Control Tower Guardrails**

* Provides ongoing governance for your Control Tower environment (AWS Accounts)
* Preventive Guardrail – using SCPs (e.g., Restrict Regions across all your accounts)
* Detective Guardrail – using AWS Config (e.g., identify untagged resources)

EC2 (Compute)

1. **EC2 = Elastic Compute Cloud = Infrastructure as a Service**

It mainly consists in the capability of:

* Renting virtual machines (EC2)
* Storing data on virtual drives (EBS)
* Distributing load across machines (ELB)
* Scaling the services using an auto-scaling group (ASG)

1. **EC2 sizing & configuration options**

* Operating System (OS): Linux, Windows or Mac OS
* How much compute power & cores (CPU)
* How much random-access memory (RAM)
* How much storage space:
  + Network-attached (EBS & EFS)
  + Hardware (EC2 Instance Store)
* Network card: speed of the card, Public IP address
* Firewall rules: security group
* Bootstrap script (configure at first launch): EC2 User Data

1. **EC2 User Data**

* Bootstrapping means launching commands when a machine starts
* That script is only run once at the instance first start
* EC2 user data is used to automate boot tasks such as: installing updates, software, downloading common files from the internet

1. **Introduction to Security Groups**

* Security groups are acting as a “firewall” on EC2 instances
* They regulate:
  + Access to Ports
  + Authorised IP ranges – IPv4 and IPv6
  + Control of inbound network (from other to the instance)
  + Control of outbound network (from the instance to other)
* Can be attached to multiple instances
* Locked down to a region / VPC combination
* Does live “outside” the EC2 – if traffic is blocked the EC2 instance won’t see it
* It’s good to maintain one separate security group for SSH access
* If your application is not accessible (time out), then it’s a security group issue
* If your application gives a “connection refused“ error, then it’s an application error or it’s not launched
* All inbound traffic is blocked by default
* All outbound traffic is authorised by default

1. **Classic Ports to know**

* 22 = SSH (Secure Shell) - log into a Linux instance
* 21 = FTP (File Transfer Protocol) – upload files into a file share
* 22 = SFTP (Secure File Transfer Protocol) – upload files using SSH
* 80 = HTTP – access unsecured website
* 443 = HTTPS – access secured websites
* 3389 = RDP (Remote Desktop Protocol) – log into a Windows instance

vs RDS Databases ports:

* PostgreSQL: 5432
* MySQL: 3306
* Oracle RDS: 1521
* MSSQL Server: 1433
* MariaDB: 3306 (same as MySQL)
* Aurora: 5432 (if PostgreSQL compatible) or 3306 (if MySQL compatible)

1. **Public vs Private IP**

Public IP:

* Public IP means the machine can be identified on the internet (WWW)
* Must be unique across the whole web (not two machines can have the same public IP).
* Can be geo-located easily

Private IP:

* Private IP means the machine can only be identified on a private network only
* The IP must be unique across the private network
* BUT two different private networks (two companies) can have the same IPs.
* Machines connect to WWW using a NAT + internet gateway (a proxy)
* Only a specified range of IPs can be used as private IP

By default, your EC2 machine comes with:

* A private IP for the internal AWS Network
* A public IP, for the WWW.

When we are doing SSH into our EC2 machines:

* We can’t use a private IP, because we are not in the same network
* We can only use the public IP
* If your machine is stopped and then started, the public IP can change

1. **Elastic IPs**

* When you stop and then start an EC2 instance, it can change its public IP.
* If you need to have a fixed public IP for your instance, you need an Elastic IP
* An Elastic IP is a public IPv4 IP you own as long as you don’t delete it
* You can attach it to one instance at a time
* With an Elastic IP address, you can mask the failure of an instance or software by rapidly remapping the address to another instance in your account.
* You can only have 5 Elastic IP in your account (you can ask AWS to increase that).
* Overall, try to avoid using Elastic IP:
  + They often reflect poor architectural decisions
  + Instead, use a random public IP and register a DNS name to it
  + Or, use a Load Balancer and don’t use a public IP

1. **Placement Groups:** Provide control over the EC2 Instance placement strategy

Placement Group strategies:

* **Cluster**—clusters instances into a low-latency group in a single Availability Zone
  + Pros: Great network (10 Gbps bandwidth between instances with Enhanced Networking enabled - recommended)
  + Cons: If the AZ fails, all instances fails at the same time
  + Use case: Big Data job that needs to complete fast, Application that needs extremely low latency and high network throughput
* **Spread**—spreads instances across underlying hardware (max 7 instances per group per AZ)
  + Pros:
    - Can span across Availability Zones (AZ)
    - Reduced risk is simultaneous failure
    - EC2 Instances are on different physical hardware
  + Cons: Limited to 7 instances per AZ per placement group
  + Use case: Application that needs to maximize high availability, Critical Applications where each instance must be isolated from failure from each other
* **Partition—**spreads instances across many different partitions (which rely on different sets of racks) within an AZ. Scales to 100s of EC2 instances per group (Hadoop, Cassandra, Kafka)
  + Up to 7 partitions per AZ •
  + Can span across multiple AZs in the same region
  + Up to 100s of EC2 instances
  + The instances in a partition do not share racks with the instances in the other partitions
  + A partition failure can affect many EC2 but won’t affect other partitions
  + EC2 instances get access to the partition information as metadata
  + Use cases: HDFS, HBase, Cassandra, Kafka

1. **Elastic Network Interfaces (ENI)**

* Logical component in a VPC that represents a virtual network card
* ENI can have the following attributes:
  + Primary private IPv4, one or more secondary IPv4
  + One Elastic IP (IPv4) per private IPv4
  + One Public IPv4
  + One or more security groups
  + A MAC address
* You can create ENI independently and attach them on the fly (move them) on EC2 instances for failover
* Bound to a specific availability zone (AZ)

1. **EC2 Hibernate**

* The in-memory (RAM) state is preserved
* The instance boot is much faster! (the OS is not stopped / restarted)
* Under the hood: the RAM state is written to a file in the root EBS volume
* The root EBS volume must be encrypted
* Use cases: Long-running processing, Saving the RAM state, Services that take time to initialize
* Supported Instance Families – C3, C4, C5, I3, M3, M4, R3, R4, T2, T3,
* Instance RAM Size – must be less than 150 GB.
* Instance Size – not supported for bare metal instances.
* AMI – Amazon Linux 2, Linux AMI, Ubuntu, RHEL, CentOS & Windows
* Root Volume – must be EBS, encrypted, not instance store, and large
* Available for On-Demand, Reserved and Spot Instances
* An instance can NOT be hibernated more than 60 days

EC2 Instance Types

1. **General Purpose Instances:**

* **Balanced compute, memory, and networking resources**, suitable for a wide variety of applications.
* **Use cases**: Web servers, development environments, small databases.
* **Examples**:
  + **T4g, T3, T3a** (Burstable instances): Ideal for applications with variable CPU usage that benefit from CPU burst capacity.
  + **M7g, M6i, M5, M5a**: For applications requiring a balance of compute, memory, and network resources, like small databases, gaming servers, and caching.

1. **Compute Optimized Instances:**

* **High-performance processors** optimized for compute-intensive tasks.
* **Use cases**: High-performance web servers, batch processing, scientific modeling, high-performance computing (HPC), and machine learning inference.
* **Examples**: **C7g, C6i, C5, C5a**: Suited for compute-bound applications that require high CPU performance, like batch processing, gaming, or distributed analytics.

1. **Memory Optimized Instances:**

* **High memory capacity** for applications that process large datasets in memory.
* **Use cases**: In-memory databases (e.g., Redis, Memcached), big data analytics, real-time processing of large datasets.
* **Examples**:
  + **R7g, R6i, R5, R5a**: Ideal for memory-bound applications such as databases and data analytics.
  + **X2idn, X2iedn**: Offers a larger memory footprint for extremely large in-memory workloads like SAP HANA.
  + **z1d**: Offers both high CPU and memory capacity, suited for high-performance databases and in-memory analytics.

1. **Storage Optimized Instances:**

* **High, fast local storage** optimized for read/write-intensive storage operations.
* **Use cases**: NoSQL databases, data warehousing, Hadoop, distributed file systems.
* **Examples**:
  + **I4i, I3, I3en**: Ideal for low-latency, high-throughput storage applications like databases.
  + **D2, D3, D3en**: Suitable for high-density storage requirements such as log processing and distributed file systems.

EC2 Instances Purchasing Options

1. **On-Demand Instances** – short workload, predictable pricing, pay by second

* pay for instances and compute capacity that you use by the hour
  + Linux or Windows - billing per second, after the first minute
  + All other operating systems - billing per hour
* no long-term commitments or up-front payments
* Recommended for short-term and un-interrupted workloads, where you can't predict how the application will behave

1. **Reserved Instances** – long workloads

* Up to 72% discount compared to On-demand
* Reserve a specific instance attribute (Instance Type, Region, Tenancy, OS)
* Reservation Period – 1 year (+discount) or 3 years (+++discount)
* Payment Options – No Upfront (+), Partial Upfront (++), All Upfront (+++)
* Consolidate Billing - pay for the entire term regardless of the usage
* can be modified to switch Availability Zones or the instance size within the same instance type, given the instance size footprint (Normalization factor) remains the same
* is not a physical instance that is launched, but rather a billing discount applied to the use of On-Demand Instances
* Recommended for steady-state usage applications (think database)
* Reserved Instance’s Scope – Regional or Zonal (reserve capacity in an AZ)
* Can buy and sell in the Reserved Instance Marketplace
* Convertible Reserved Instance - Can change the EC2 instance type, instance family, OS, scope and tenancy

1. **Scheduled Reserved Instances** – long workloads with flexible instances

* enable capacity reservations purchase that recurs on a daily, weekly, or monthly basis, with a specified start time and duration, for a one-year term.
* Charges are incurred for the time that the instances are scheduled, even if they are not used
* good choice for workloads that do not run continuously, but do run on a regular schedule

1. **EC2 Savings Plans** –commitment to an amount of usage, long workload

* Get a discount based on long-term usage (up to 72% - same as RIs)
* Commit to a certain type of usage ($10/hour for 1 or 3 years)
* Usage beyond EC2 Savings Plans is billed at the On-Demand price
* Locked to a specific instance family & AWS region (e.g., M5 in us-east-1)
* Flexible across:
  + Instance Size (e.g., m5.xlarge, m5.2xlarge)
  + OS (e.g., Linux, Windows)
  + Tenancy (Host, Dedicated, Default)

1. **Spot Instances** – short workloads, cheap, can lose instances (less reliable)

* Can get a discount of up to 90% compared to On-demand
* Instances that you can “lose” at any point of time if your max price is less than the current spot price
* The hourly spot price varies based on offer and capacity
* Provides a two-minute warning if the instance is to be terminated to save any unsaved work
* Useful for workloads that are resilient to failure: Batch jobs, Data analysis, Image processing, Any distributed workloads, Workloads with a flexible start and end time
* Spot blocks can also be launched with a required duration, which are not interrupted due to changes in the Spot price

1. **Spot Fleet**

* Spot Fleets = set of Spot Instances + (optional) On-Demand Instances
* The Spot Fleet will try to meet the target capacity with price constraints
  + Define possible launch pools: instance type (m5.large), OS, Availability Zone
  + Can have multiple launch pools, so that the fleet can choose
  + Spot Fleet stops launching instances when reaching capacity or max cost
* Strategies to allocate Spot Instances:
  + **lowestPrice:** from the pool with the lowest price (cost optimization, short workload)
  + **diversified:** distributed across all pools (great for availability, long workloads)
  + **capacityOptimized:** pool with the optimal capacity for the number of instances
  + **priceCapacityOptimized** (recommended): pools with highest capacity available, then select the pool with the lowest price (best choice for most workloads)
* Spot Fleets allow us to automatically request Spot Instances with the lowest price

1. **Dedicated Instances** – no other customers will share your hardware

* is a tenancy option that enables instances to run in VPC on hardware that’s isolated, dedicated to a single customer

1. **Dedicated Host** – book an entire physical server, control instance placement

* is a physical server with EC2 instance capacity fully dedicated to your use
* Allows you address compliance requirements and use your existing server- bound software licenses (per-socket, per-core, pe—VM software licenses)
* Purchasing Options:
  + On-demand – pay per second for active Dedicated Host
  + Reserved - 1 or 3 years (No Upfront, Partial Upfront, All Upfront)
* The most expensive option
* Useful for software that have complicated licensing model (BYOL – Bring Your Own License), Or for companies that have strong regulatory or compliance needs

EC2 Instance Storage

1. **EBS**

* A network drive you can attach to your instances while they run
* It allows your instances to persist data, even after their termination
* They can only be mounted to one instance at a time (at the CCP level)
* They are bound to a specific availability zone
* Free tier: 30 GB of free EBS storage of type General Purpose (SSD) or Magnetic per month

1. **EBS Volume**

* A network drive (i.e. not a physical drive)
  + Uses the network to communicate the instance, which means there might be a bit of latency
  + It can be detached from an EC2 instance and attached to another one quickly
* Locked to an Availability Zone (AZ)
  + To move a volume across, you first need to snapshot it
* Have a provisioned capacity (size in GBs, and IOPS)
  + You get billed for all the provisioned capacity
  + You can increase the capacity of the drive over time

1. **EBS Delete on Termination Attribute**

* Controls the EBS behaviour when an EC2 instance terminates
* By default, the root EBS volume is deleted (attribute enabled)
* By default, any other attached EBS volume is not deleted (attribute disabled)
* This can be controlled by the AWS console / AWS CLI
* Use case: preserve root volume when instance is terminated

1. **EBS Snapshot**

* Make a backup (snapshot) of your EBS volume at a point in time
* Not necessary to detach volume to do snapshot, but recommended
* Can copy snapshots across AZ or Region
* EBS Snapshot Archive
  + Move a Snapshot to an ”archive tier” that is 75% cheaper
  + Takes within 24 to 72 hours for restoring the archive
* Recycle Bin for EBS Snapshots
  + Setup rules to retain deleted snapshots so you can recover them after an accidental deletion
  + Specify retention (from 1 day to 1 year)
* Fast Snapshot Restore (FSR)
  + Force full initialization of snapshot to have no latency on the first use ($$$)

1. **AMI = Amazon Machine Image**

* AMI are a customization of an EC2 instance
* You add your own software, configuration, operating system, monitoring
* Faster boot / configuration time because all your software is pre-packaged
* AMI are built for a specific region (and can be copied across regions)
* You can launch EC2 instances from:
  + A Public AMI: AWS provided
  + Your own AMI: you make and maintain them yourself
  + An AWS Marketplace AMI: an AMI someone else made (and potentially sells)
* AMI Process (From EC2 instance)
  + Start an EC2 instance and customize it
  + Stop the instance (for data integrity)
  + Build an AMI – this will also create EBS snapshots
  + Launch instances from other AMIs

1. **EC2 Instance Store**

* EBS volumes are network drives with good but “limited” performance
* If you need a high-performance hardware disk, use EC2 Instance Store
* Better I/O performance
* EC2 Instance Store lose their storage if they’re stopped (ephemeral)
* Good for buffer / cache / scratch data / temporary content
* Risk of data loss if hardware fails
* Backups and Replication are your responsibility

1. **EBS Volume Types**

* EBS Volumes come in 6 types
  + gp2 / gp3 (SSD): General purpose SSD volume that balances price and performance for a wide variety of workloads
  + io1 / io2 Block Express (SSD): Highest-performance SSD volume for mission-critical low-latency or high-throughput workloads
  + st1 (HDD): Low cost HDD volume designed for frequently accessed, throughput- intensive workloads
  + sc1 (HDD): Lowest cost HDD volume designed for less frequently accessed workloads
* EBS Volumes are characterized in Size | Throughput | IOPS (I/O Ops Per Sec)
* Only gp2/gp3 and io1/io2 Block Express can be used as boot volumes

1. **EBS General Purpose SSD**

* Cost effective storage, low-latency
* System boot volumes, Virtual desktops, Development and test environments
  + 1 GiB - 16 TiB
  + gp3:
    - Baseline of 3,000 IOPS and throughput of 125 MiB/s
    - Can increase IOPS up to 16,000 and throughput up to 1000 MiB/s independently
  + gp2:
    - Small gp2 volumes can burst IOPS to 3,000
    - Size of the volume and IOPS are linked, max IOPS is 16,000
    - 3 IOPS per GB, means at 5,334 GB we are at the max IOPS

1. **Provisioned IOPS (PIOPS) SSD**

* Critical business applications with sustained IOPS performance
* Or applications that need more than 16,000 IOPS
* Great for databases workloads (sensitive to storage perf and consistency)
  + io1 (4 GiB - 16 TiB):
    - Max PIOPS: 64,000 for Nitro EC2 instances & 32,000 for other
    - Can increase PIOPS independently from storage size
  + io2 Block Express (4 GiB – 64 TiB):
    - Sub-millisecond latency
    - Max PIOPS: 256,000 with an IOPS:GiB ratio of 1,000:1
* Supports EBS Multi-attach

1. **Hard Disk Drives (HDD)**

* Cannot be a boot volume
* 125 GiB to 16 TiB
* Throughput Optimized HDD (st1)
  + Big Data, Data Warehouses, Log Processing
  + Max throughput 500 MiB/s – max IOPS 500
* Cold HDD (sc1):
  + For data that is infrequently accessed
  + Scenarios where lowest cost is important
  + Max throughput 250 MiB/s – max IOPS 250

1. **EBS Multi-Attach - io1/io2 family**

* Attach the same EBS volume to multiple EC2 instances in the same AZ
* Each instance has full read & write permissions to the high-performance volume
* Use case:
  + Achieve higher application availability in clustered Linux applications (ex: Teradata)
  + Applications must manage concurrent write operations
  + Up to 16 EC2 Instances at a time
  + Must use a file system that’s cluster-aware (not XFS, EXT4, etc…)

1. **EBS Encryption**

* When you create an encrypted EBS volume, you get the following:
  + Data at rest is encrypted inside the volume
  + All the data in flight moving between the instance and the volume is encrypted
  + All snapshots are encrypted
  + All volumes created from the snapshot
* Encryption and decryption are handled transparently
* Encryption has a minimal impact on latency
* EBS Encryption leverages keys from KMS (AES-256)
* Copying an unencrypted snapshot allows encryption
* Snapshots of encrypted volumes are encrypted

1. **Encryption: encrypt an unencrypted EBS volume**

* Create an EBS snapshot of the volume
* Encrypt the EBS snapshot (using copy)
* Create new EBS volume from the snapshot ( the volume will also be encrypted )
* Now you can attach the encrypted volume to the original instance

1. **EFS Elastic File System**

* Managed NFS (network file system) that can be mounted on many EC2
* EFS works with EC2 instances in multi-AZ
* Highly available, scalable, expensive (3x gp2), pay per use
* Use cases: content management, web serving, data sharing, Wordpress
* Uses NFSv4.1 protocol
* Uses security group to control access to EFS
* Compatible with Linux based AMI (not Windows)
* Encryption at rest using KMS
* POSIX file system (~Linux) that has a standard file API
* File system scales automatically, pay-per-use, no capacity planning!

1. **EFS – Performance and Storage Classes**

* EFS Scale
  + 1000s of concurrent NFS clients, 10 GB+ /s throughput
  + Grow to Petabyte-scale network file system, automatically
* Performance Mode (set at EFS creation time)
  + General Purpose (default) – latency-sensitive use cases (web server, CMS, etc)
  + Max I/O – higher latency, throughput, highly parallel (big data, media processing)
* Throughput Mode
  + Bursting – 1 TB = 50MiB/s + burst of up to 100MiB/s
  + Provisioned – set your throughput regardless of storage size, ex: 1 GiB/s for 1 TB storage
  + Elastic – automatically scales throughput up or down based on your workloads
    - Up to 3GiB/s for reads and 1GiB/s for writes
    - Used for unpredictable workloads

1. **EFS – Storage Classes**

* Storage Tiers (lifecycle management feature – move file after N days)
  + Standard: for frequently accessed files
  + Infrequent access (EFS-IA): cost to retrieve files, lower price to store.
  + Archive: rarely accessed data (few times each year), 50% cheaper
  + Implement lifecycle policies to move files between storage tiers
* Availability and durability
  + Standard: Multi-AZ, great for prod
  + One Zone: One AZ, great for dev, backup enabled by default, compatible with IA (EFS One Zone-IA)
* Over 90% in cost savings

Auto Scaling and Load Balancing

1. **Scalability and Availability**

* **Vertical Scaling:** Increasing the size of the instance t2.micro -> t2.large **e.g.** database
* **Horizontal:** Increase # instances; distributed systems. E.g. Web Apps
* **High Availability** = >1 Availability Zones to survive a data center loss
  + Can be passive (for RDS Multi AZ for example) or active (for horizontal scaling)

1. **Availability and Scaling for EC2**

* Vertical Scaling: Increase instance size (= scale up / down)
  + From t2.nano - 0.5G of RAM, 1 vCPU to u-12tb1.metal – 12.3 TB of RAM, 448 vCPUs
* Horizontal Scaling: Increase number of instances (= scale out / in)
  + Auto Scaling Group, Load Balancer
* High Availability: Run instances for the same application across multi AZ
  + Auto Scaling Group multi AZ, Load Balancer multi AZ

1. **Load Balancing**

* Load Balances are servers that forward traffic to multiple servers (e.g., EC2 instances) downstream
* Why use load balancer:
  + Spread load across multiple downstream instances
  + Expose a single point of access (DNS) to your application
  + Seamlessly handle failures of downstream instances
  + Do regular health checks to your instances
  + Provide SSL termination (HTTPS) for your websites
  + Enforce stickiness with cookies
  + High availability across zones
  + Separate public traffic from private traffic

1. **Health Checks**

* Enable the load balancer to know if instances it forwards traffic to are available to reply to requests
* Done on a port and a route (/health is common)
* If the response is not 200 (OK), then the instance is unhealthy

1. **SSL Certificate**

* An SSL Certificate allows traffic between your clients and your load balancer to be encrypted in transit (in-flight encryption)
* SSL refers to Secure Sockets Layer, used to encrypt connections
* TLS refers to Transport Layer Security, which is a newer version
* Public SSL certificates are issued by Certificate Authorities (CA)
* SSL certificates have an expiration date (you set) and must be renewed

1. **SNI Server Name Indication**

* SNI solves the problem of loading multiple SSL certificates onto one web server (to serve multiple websites)
* A “newer” protocol, and requires the client to indicate the hostname of the target server in the initial SSL handshake
* The server will then find the correct certificate, or return the default one

1. **ELB Connection Draining**

* Feature naming
  + Connection Draining – for CLB
  + Deregistration Delay – for ALB & NLB
* Time to complete “in-flight requests” while the instance is de-registering or unhealthy
* Stops sending new requests to the EC2 instance which is de-registering
* Between 1 to 3600 seconds (default: 300 seconds)
* Can be disabled (set value to 0)
* Set to a low value if your requests are short

1. **Auto Scaling Group**

* Goal of an Auto Scaling Group (ASG):
  + Scale out (add EC2 instances) to match an increased load
  + Scale in (remove EC2 instances) to match a decreased load
  + Ensure we have a minimum and a maximum number of EC2 instances running
  + Automatically register new instances to a load balancer
  + Re-create an EC2 instance in case a previous one is terminated (ex: if unhealthy)
* ASG are free (you only pay for the underlying EC2 instances)

1. **ASG Policies**

* Strategies used to automatically adjust the capacity of services (such as EC2 instances) based on demand, ensuring that applications remain responsive while optimizing costs.
* **Simple Scaling**
  + How it works: Triggered by CloudWatch alarms based on a specific metric. When an alarm is triggered (e.g., CPU exceeds 70%), it adds or removes instances.
  + Use case: Works well for straightforward, event-driven scaling actions. However, scaling can be slower as each scaling activity waits for previous actions to complete before re-evaluating.
* **Step Scaling**
  + How it works: Responds to changes in metrics with predefined step actions. You define thresholds and the corresponding scaling actions (e.g., add 2 instances if CPU > 70%).
  + Use case: Provides more granular control, allowing you to scale differently depending on the size of the demand spike.
* **Target Tracking Scaling**
  + How it works: Similar to setting a thermostat, you define a target metric (e.g., average CPU utilization) and the system automatically scales the number of instances to maintain that target.
  + Use case: Simplifies scaling decisions by focusing on a single performance metric, like keeping CPU usage at 60%.
* **Scheduled Scaling**
  + How it works: Scaling actions are scheduled to occur at specific times based on anticipated patterns of demand. You define when to scale in or scale out based on time or predictable patterns.
  + Use case: Ideal for workloads with predictable traffic patterns, such as:
    - An application that sees regular increases in traffic at certain hours of the day (e.g., peak business hours).
    - Batch jobs that run at specific times.
* **Predictive Scaling**
  + **How it works**: Uses machine learning to predict future traffic patterns and automatically adjust capacity in advance. It analyzes historical data and identifies patterns, allowing it to anticipate when traffic will increase or decrease.
  + **Use case**: Ideal for applications where traffic fluctuates regularly but may not always be predictable by simple schedules. Examples include:
    - E-commerce platforms that see spikes during sales or holiday seasons.
    - Applications with irregular but discernible traffic patterns.

Types of load balancer on AWS

1. **Application Load Balancer (ALB)**

* **OSI Layer**: Operates at Layer 7 (Application layer).
* **Traffic Type**: Deals with HTTP/HTTPS (Web traffic).
* **Routing**: ALB routes traffic based on content of the request, like URL path, headers, and HTTP methods (i.e., path-based or host-based routing). It can also terminate SSL connections.
* **Use Case**: Ideal for distributing traffic across web applications, API gateways, or microservices architectures. It can handle advanced routing and content-based rules.

1. **Network Load Balancer (NLB)**

* **OSI Layer**: Operates at Layer 4 (Transport layer).
* **Traffic Type**: Handles TCP, UDP, and TLS traffic (lower-level network protocols).
* **Routing**: NLB forwards traffic to targets based on IP address and port, without inspecting the content of the packets. It supports extreme performance, handling millions of requests per second with low latency.
* **Use Case**: Suitable for scenarios requiring very high-performance, low-latency load balancing for TCP/UDP-based traffic (like database traffic, non-HTTP services, etc.).

1. **Gateway Load Balancer (GWLB)**

* **OSI Layer**: Operates at Layer 3 (Network layer).
* **Traffic Type**: Handles IP traffic.
* **Routing**: Primarily used to distribute traffic across multiple network appliances, such as firewalls, intrusion detection systems, or other security devices.
* **Use Case**: Ideal for integrating third-party virtual appliances (e.g., firewall, security inspection devices) into your network at scale, allowing traffic to pass through these appliances before reaching its destination.

1. **Elastic Load Balancer** – AWS managed load balancer that includes ALP, NLB and GWLB

* AWS guarantees that it will be working
  + AWS takes care of upgrades, maintenance, high availability
  + AWS provides only a few configuration knobs
* Costs less to setup your own load balancer but it will be a lot more effort
* It is integrated with many AWS offerings / services
  + EC2, EC2 Auto Scaling Groups, Amazon ECS
  + AWS Certificate Manager (ACM), CloudWatch
  + Route 53, AWS WAF, AWS Global Accelerator
* Features:
  + **Elasticity**: Automatically adjusts capacity to handle the incoming traffic, scaling up or down depending on demand.
  + **Health Checks**: Monitors the health of registered targets and routes traffic only to healthy instances.
  + **Security**: Works with AWS security groups and integrates with AWS Identity and Access Management (IAM) and AWS Web Application Firewall (WAF) to enhance security.
  + **Cross-Zone Load Balancing**: Distributes traffic evenly across registered instances in different Availability Zones.
  + **Automatic Scaling**: As traffic increases or decreases, ELB scales accordingly to maintain consistent application performance.
  + **Session Affinity (Stickiness):** a feature in AWS ELB that allows you to bind a user's session to a specific target (such as an EC2 instance) within the load balancer's target group.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Application Load Balancer (ALB)** | **Network Load Balancer (NLB)** | **Gateway Load Balancer (GWLB)** |
| **OSI Layer** | Layer 7 (Application) | Layer 4 (Transport) | Layer 3 (Network) |
| **Traffic** | HTTP/HTTPS | TCP/UDP | IP traffic |
| **Routing** | Content-based routing | IP and port-based routing | Route to network appliances |
| **Primary Use Case** | Web applications, APIs | High-performance networking | Network security appliances |
| **SSL Termination** | Yes | Yes (TLS offloading) | No |

Route 53

1. **DNS (Domain Name System)**

* translates the human friendly hostnames into the machine IP addresses
* www.google.com => 172.217.18.36
* DNS is the backbone of the Internet • DNS uses hierarchical naming structure

A screen shot of a computer

Description automatically generated

1. **Route 53**

* A highly available, scalable, fully managed and Authoritative DNS
* Authoritative = the customer (you) can update the DNS records
* Route 53 is also a Domain Registrar
* Ability to check the health of your resources
* The only AWS service which provides 100% availability SLA
* Why Route 53? 53 is a reference to the traditional DNS port

1. **Route 53 – Records**

* Each record contains:
  + Domain/subdomain Name – e.g., example.com
  + Record Type – e.g., A or AAAA
  + Value – e.g., 12.34.56.78
  + Routing Policy – how Route 53 responds to queries
  + TTL time to live – amount of time the record cached at DNS Resolvers
* Route 53 supports the following DNS record types:
  + (must know) A / AAAA / CNAME / NS
  + (advanced) CAA / DS / MX / NAPTR / PTR / SOA / TXT / SPF / SRV

1. **Route 53 – Record Types**

* A – maps a hostname to IPv4
* AAAA – maps a hostname to IPv6
* CNAME – maps a hostname to another hostname
  + The target is a domain name which must have an A or AAAA record
  + Can’t create a CNAME record for the top node of a DNS namespace (Zone Apex)
  + Example: you can’t create for example.com, but you can create for www.example.com
* NS – Name Servers for the Hosted Zone
  + Control how traffic is routed for a domain

1. **Route 53 – Hosted Zones**

* A container for records that define how to route traffic to a domain and its subdomains
* Public Hosted Zones – contains records that specify how to route traffic on the Internet (public domain names) application1.mypublicdomain.com
* Private Hosted Zones – contain records that specify how you route traffic within one or more VPCs (private domain names) application1.company.internal
* You pay $0.50 per month per hosted zone

Route 53 Routing Policy

**Routing Policy**

* Define how Route 53 responds to DNS queries
* DNS does not route any traffic, it only responds to the DNS queries

1. **Simple**

* Typically, route traffic to a single resource
* Can specify multiple values in the same record
* If multiple values are returned, a random one is chosen by the client
* When Alias enabled, specify only one AWS resource
* Can’t be associated with Health Checks

1. **Weighted**

* Control the % of the requests that go to each specific resource
* Assign each record a relative weight
* Weights don’t need to sum up to 100
* DNS records must have the same name and type
* Can be associated with Health Checks
* Use cases: load balancing between regions, testing new application versions
* Assign a weight of 0 to a record to stop sending traffic to a resource
* If all records have weight of 0, then all records will be returned equally

1. **Latency**

* Redirect to the resource that has the least latency close to us
* Latency is based on traffic between users and AWS Regions e.g. Germany users may be directed to the US (if that’s the lowest latency)
* Can be associated with Health Checks (has a failover capability)

1. **Health checks**

* HTTP Health Checks are only for public resources
* Health Check -> Automated DNS Failover
  + Health checks that monitor an endpoint (application, server, other AWS resources)
  + Health checks that monitor other health checks (Calculated Health Checks)
  + Health checks that monitor CloudWatch Alarms (full control) – e.g., throttles of DynamoDB, alarms on RDS, custom metrics (helpful for private resources)
* Health Checks are integrated with cloudwatch metrics

1. **Failover**

* Used to route traffic to a primary resource (usually a website or application) and a secondary (backup) resource in case the primary fails.
* Useful for achieving **high availability and disaster recovery** for applications.
* Primary Resource: The resource that normally receives traffic (e.g., a primary server, primary instance, or primary data center).
* Secondary Resource (Failover): The backup resource that takes over in case the primary resource becomes unavailable.
* Health Checks: Route 53 continuously monitors the health of the primary resource using health checks. If the primary resource becomes unhealthy (fails a health check), Route 53 redirects traffic to the secondary resource.

1. **Geolocation**

* Different from latency-based; based on user location
* Specify location by continent, country or by US State (if there’s overlapping, most precise location selected)
* Should create a “Default” record (in case there’s no match on location)
* Use cases: website localization, restrict content distribution, load balancing
* Can be associated with Health Checks

1. **Geoproximity**

* Route traffic to your resources based on the geographic location of users and resources
* Ability to shift more traffic to resources based on the defined bias
* To change the size of the geographic region, specify bias values:
  + To expand (1 to 99) – more traffic to the resource
  + To shrink (-1 to -99) – less traffic to the resource
* Resources can be:
  + AWS resources (specify AWS region)
  + Non-AWS resources (specify Latitude and Longitude)
* Must use Route 53 Traffic Flow to use this feature

1. **IP-based Routing**

* Routing is based on clients’ IP addresses
* Provide a list of CIDRs (Classless Inter-Domain Routing) for your clients and the corresponding endpoints/locations (user-IP-to-endpoint mappings)
* Use cases: Optimize performance, reduce network costs
* Example: route end users from a particular ISP to a specific endpoint

1. **Multi-Value**

* Use when routing traffic to multiple resources
* Route 53 return multiple values/resources
* Can be associated with Health Checks (return only values for healthy resources)
* Up to 8 healthy records are returned for each Multi-Value query
* Multi-Value is not a substitute for having an ELB

S3

S3 Storage Classes

S3 Security

CloudFront & Global Accelerator

Decoupling Applications

Containers

Serverless Services in AWS

**Serverless services:** a cloud computing model where the cloud provider manages the server infrastructure, allowing developers to abstract away the details of server management. **Serverless** means:

* **No Server Management**: You don’t have to provision, scale, or maintain servers. The cloud provider handles all infrastructure concerns.
* **Automatic Scaling**: The system scales automatically to accommodate incoming traffic or load without manual intervention.
* **Pay-as-You-Go Pricing**: You only pay for the compute resources you use, not for idle server time.

**Key AWS Serverless Services:**

1. **AWS Lambda**

* **Purpose**: Run code in response to events (such as changes in data or system state) without provisioning or managing servers.
* **Use Cases**: Processing files uploaded to S3, responding to HTTP requests via API Gateway, performing ETL tasks, etc.
* **Integration**
* **Limitations (Execution):**
* **Limitations (Deployment):**
* **SnapStart:**

1. **Lambda@Edge**
2. **CloudFront Functions**
3. **Amazon API Gateway**

* **Purpose**: Create, publish, maintain, monitor, and secure APIs at any scale. It can integrate with AWS Lambda to run code in response to API requests.
* **Use Cases**: Building RESTful APIs, WebSocket APIs, and HTTP APIs for your serverless applications.

1. **AWS Fargate**

* **Purpose**: Run containers without managing the underlying server infrastructure. It’s used with Amazon ECS (Elastic Container Service) and Amazon EKS (Elastic Kubernetes Service).
* **Use Cases**: Containerized applications that need to run without managing the server or cluster infrastructure.

1. **Amazon DynamoDB**

* **Purpose**: A fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.
* **Use Cases**: Building serverless applications that require high-speed, low-latency data access.

1. **Amazon S3 (Simple Storage Service)**

* **Purpose**: Object storage service that provides scalable, durable, and secure storage for data.
* **Use Cases**: Storing static assets like images, videos, and backups; triggering Lambda functions upon data uploads.

1. **AWS Step Functions**

* **Purpose**: Coordinate the components of distributed applications and microservices using visual workflows.
* **Use Cases**: Orchestrating serverless workflows and state machines, managing complex workflows with retries and error handling.

1. **Amazon EventBridge**

* **Purpose**: A serverless event bus that makes it easy to connect applications with data from various sources, including AWS services, integrated SaaS applications, and custom applications.
* **Use Cases**: Building event-driven architectures, integrating microservices, and reacting to application events.

1. **AWS AppSync**

* **Purpose**: Create and manage GraphQL APIs that connect to data sources like DynamoDB, Lambda, or Elasticsearch.
* **Use Cases**: Building real-time applications, managing complex data interactions with GraphQL.

1. **AWS Aurora Serverless**

* **Purpose**: An on-demand, auto-scaling configuration for Amazon Aurora (a relational database service) that automatically adjusts capacity based on application needs.
* **Use Cases**: Applications with variable database workloads where you want to avoid over-provisioning or under-provisioning.

1. **Amazon Cognito**
2. SNS & SQS
3. Kinesis Data Firehose

Database

1. RDS
2. Aurora
3. Difference between RDS and Aurora
4. DynamoDB
5. DocumentDB
6. Neptune
7. **ElasticCache**

* ElastiCache is to get managed Redis or Memcached
* Caches are in-memory databases with really high performance, low latency
* Helps reduce load off of databases for read intensive workloads • Helps make your application stateless
* AWS takes care of OS maintenance / patching, optimizations, setup, configuration, monitoring, failure recovery and backups
* Using ElastiCache involves heavy application code changes

1. Keyspace
2. QLDB
3. Timestream

Data and Analytics

* [Redshift](https://jayendrapatil.com/aws-redshift/) as a business intelligence tool
* [Kinesis](https://jayendrapatil.com/aws-kinesis/)
  + for real-time data capture and analytics.
  + Integrates with Lambda functions to perform transformations
* [AWS Glue](https://jayendrapatil.com/aws-glue/)  
  + fully-managed, ETL service that automates the time-consuming steps of data preparation for analytics

Machine learning

**1. Amazon Kendra**

* **Purpose**: Enterprise search service that uses machine learning to provide more relevant search results
* **Use Case**: Building search capabilities for internal documents and knowledge bases.
* **Features**: Natural language queries, document indexing, integrated with various data sources, incremental learning

**2. Amazon Transcribe**

* **Purpose**: Automatic speech recognition (ASR) service that converts speech to text.
* **Use Case**: Transcribing audio recordings, voice commands, and meeting notes.
* **Features**: Custom vocab, speaker identification, real-time transcription, remove PIIs

**3. Amazon Polly**

* **Purpose**: Text-to-speech (TTS) service that converts text into lifelike speech.
* **Use Case**: Creating audio versions of written content for applications and accessibility.
* **Features**: Multiple languages and voices, and support for SSML (Speech Synthesis Markup Language).

**4. Amazon Comprehend**

* **Purpose**: Natural language processing (NLP) service that extracts insights from text.
* **Use Case**: Sentiment analysis, entity recognition, language detection, and topic modeling.
* **Features**: Custom entity recognition and topic modeling, multi-language support.

**5. Amazon Lex**

* **Purpose**: Service for building conversational interfaces (chatbots) using voice and text.
* **Use Case**: Creating chatbots and voice assistants for customer service and interactive applications.
* **Features**: Natural language understanding, speech recognition, and integration with AWS Lambda.

**6. Amazon Rekognition**

* **Purpose**: Image and video analysis service that provides object, scene, and activity detection.
* **Use Case**: Facial recognition, object detection, and moderation of content in images and videos.
* **Features**: Face comparison, facial analysis, and integration with other AWS services.

**7. Amazon Translate**

* **Purpose**: Neural machine translation service for translating text between languages.
* **Use Case**: Localizing content for global audiences and translating documents or user-generated content.
* **Features**: Real-time translation, support for multiple languages.

**8. Amazon Personalize**

* **Purpose**: Real-time personalization service that delivers customized recommendations.
* **Use Case**: Product recommendations, content suggestions, and user experience personalization.
* **Features**: Personalized ranking, user behavior tracking, and integration with other AWS services.

**9. Amazon Forecast**

* **Purpose**: Time series forecasting service
* **Use Case**: Demand forecasting, financial forecasting, and resource planning.
* **Features**: Automated model selection, custom forecasting models.

**10. Amazon Textract**

* **Purpose**: Extracts text, forms, and tables from scanned documents.
* **Use Case**: Automating data extraction from invoices, forms, and contracts.
* **Features**: Text extraction, form recognition, and table analysis.

**11. Amazon SageMaker**

* **Purpose**: Building, training, and deploying machine learning models.
* **Use Case**: End-to-end ML development, from data preparation to model deployment.
* **Features**: Built-in algorithms, Jupyter notebooks, SageMaker Studio, hyperparameter tuning, and model monitoring.

**11a. Amazon Augmented AI (A2I)**

* **Purpose**: Human-in-the-loop service to review and improve ML performance
* **Use Case**: Enhancing model accuracy with human review
* **Features**: Custom workflows, integration with SageMaker.

**11b. Amazon SageMaker Ground Truth**

* **Purpose**: Data labeling service to create high-quality training datasets.
* **Use Case**: Annotating images, text, or other data types for machine learning models.
* **Features**: Human labeling with built-in quality control.

**11c. Amazon SageMaker Model Monitor**

* **Purpose**: Monitors machine learning models in production to ensure performance and accuracy.
* **Use Case**: Detecting and addressing data drift and model performance issues.
* **Features**: Automated monitoring, drift detection, and alerting.

**11d. Amazon SageMaker Neo**

* **Purpose**: Optimizes machine learning models for deployment on edge devices.
* **Use Case**: Improving model performance and reducing latency on edge devices.
* **Features**: Model optimization for various hardware platforms.

AWS Monitoring and Audit

A screenshot of a computer

Description automatically generated

https://d1.awsstatic.com/training-and-certification/docs-sa-assoc/AWS-Certified-Solutions-Architect-Associate\_Exam-Guide.pdf