**[AWS Solutions Architect Associate (SAA-C03) Cheat Sheet](https://www.stellexgroup.com/blog/aws-solutions-architect-associate-saa-c03-cheat-sheet)**

**AWS Cloud Practitioner: A Prerequisite for AWS SAA?**  
Although the AWS Cloud Practitioner certification is not a prerequisite for the AWS SAA exam, I found it very helpful to study for the AWS Cloud Practitioner, especially to learn the basics of the many AWS services. Check out my [**AWS Cloud Practitioner (ACP) Study Notes**](https://www.stellexgroup.com/blog/aws-acp-exam-cheatsheet).   
  
  
**EC2 INSTANCES**  
  
**300 Instance Types across 5 Instance Families**

* AWS offers [**over 300 EC2 instance types**](https://www.densify.com/resources/ec2-instance-types) across *5 instance families* (general purpose family, memory-optimized, storage-optimized, compute-optimized, and accelerated computing), each with varying resource and performance focuses
* For example, within the compute-optimized family, you have C4 instance types (running on Haswell chips) and more recent C5 instance types (running on Skylake with Nitro system).

**Instance Purchasing Options**

* On-Demand Instances - the default option, for short-term ad-hoc requirements where the job can't be interrupted
* On-Demand Capacity Reservations - the only way to reserve capacity for blocks of time such as 9am-5pm daily
* ​Spot instance - highest discount potential (50-90%) but no commitment from AWS, could be terminated with 2min notice. Could use for grid and high-performance computing.
* Reserved Instances - for long-term workloads, *1 or 3 year commitment* in exchange for 40-60% discount
* Dedicated Instances - run on hardware dedicated to 1 customer (more $$)
* Dedicated Host - fully dedicated and *physically isolated* server. Allows you to use your server-bound software licenses (e.g. IBM, Oracle) and addresses compliance and regulatory requirements and potentially reduce cost (note: billing is per-hour not per-instance)
* Bare metal EC2 instance - for when the workload needs access to the hardware feature set (e.g. Intel hardware)

**Launching Instances**

* **Configurations / Launch Templates**used to create new EC2 instances using stored parameters such as instance family, instance type, AMI, key pair and security groups. Auto-scaling groups can launch instances using config templates. You can't edit a launch config, but you can create a new one and point to it.
* **User data**- pass up to 16KB of user data at launch that the instance can run on startup such as config scripts
* **Instance metadata** (e.g. instance ID, hostname, events, security groups, public keys, network interfaces,) can be accessed via a direct URI or by using the *Instance Metadata Query Tool*
* When you launch an EC2 instance into a default VPC, it has a public and private DNS hostname and IP address. When you launch in a non-default VPC, it may not have a public hostname depending on the DNS and VPC configs.
* Errors when launching  include *InsufficientInstanceCapacity*, *InstanceLimitExceeded*
  + Instances terminate with no error if there are EBS problems (EBS volume limit, EBS snapshot is corrupt), or if the AMI you're launching from is missing a required part
* Each EC2 instance that you launch has an associated **root device volume**, either EBS volume or an Instance Store volume (more these under Storage section below). You can use *block device mapping* to specify additional EBS volumes or instance store volumes to attach to a live instance, attach additional EBS volumes to a running instance, but can't directly add additional Instance Store volumes.
* **Run Command** - run from the AWS Management Console, CLI or SDK, to install software, execute Powershell commands and scripts, configuring Windows settings, on live EC2 instances

**Placement Groups**

* *Cluster placement group* = packs instances close together inside an AZ to achieve low latency, high throughput - use for HPC
* *Partition placement group* = separate instances into logical partitions such that instances in one partition do not share hardware with instances in another partition. Gives you control and visibility into instance placement, but not great for performance. Used by large distributed workloads such as Hadoop.
* *Spread placement group* =  place 1 or few instances each in distinct hardware to reduce correlated failures. Not great for performance

**Scaling Instances**

* In high-availability contexts you use an **Auto-Scaling Group (ASG)**to automatically launch and stop instances, and an**Elastic Load Balancer (ELB)**to distribute traffic among the instances
  + **​**specify which subnets the ASG should launch instances into
  + attach*Target Groups* to the ASG
* ASG *scaling policies*
  + ​*Simple* - maintain the # of instances, manually change the min/desired/max and attach/detach instances
  + *Scheduled* - scale based on a scheduled event or recurring schedule (e.g. if you know that you have traffic spike every morning at 9am)
  + *Dynamic* - scale in response to an event or alarm
  + *Step* - configure multiple changes to scaling based on multiple events
  + *Target Tracking*- uses a custom metric to add/remove instances
  + NOTE: AWS recommends using target tracking over step scaling, and step scaling over simple scaling in most cases
* *Cooldown period*- reducing the cooldown period will more quickly terminate unneeded instances, reducing costs
* *Enhanced networking*provides higher bandwidth, higher packet-per-second (PPS) erformance, and lower inter-instance latency. Consider if PPS is maxed out.

**VPC, SUBNETS, NETWORKING**

* A **VPC** is a virtual network that closely resembles a traditional network that you'd operate in your own data centre.
* After you create a VPC, you can add **subnets**. A subnet is a range of IP addresses in your VPC. After you add subnets, you can deploy AWS **resources** in your VPC.
* Use **route tables** to determine where network traffic from your subnet or gateway is directed.
* A **gateway** connects your VPC to another network. For example, use an *internet gateway* to connect your VPC to the internet.
* Use a **VPC endpoint**to connect to AWS services privately, without the use of an internet gateway or NAT device.
* Use a **VPC peering** connection to route traffic between the resources in two VPCs.
* Use a **Transit Gateway**, which acts as a central hub, to route traffic between your VPCs, VPN connections, and AWS Direct Connect connections.
* Connect your VPCs to your on-premises networks using **AWS Virtual Private Network (AWS VPN)**.

**Subnets**

* A VPC is housed within a Region, and a subnet maps 1-to-1 with an AZ. Therefore, for high availability, you need at least 2 subnets in your VPC so that you can span 2 AZs.
* When you create a new subnet, it is automatically associated to the main route table.
* Example VPC/subnet configurations recommended by AWS
  + VPC with single public subnet: e.g. for single-tier, public-facing web app such as a blog or a simple website
  + VPC with public and private subnets: e.g. for multi-tier web apps where the web servers are in the public subnet and the DBs in the private subnet

**Public Subnet vs. Private Subnet**

* Public Subnets
  + has a route table that routes to an Internet Gateway (note the Internet Gateway is attached to the VPC, not directly to the subnet)
  + When EC2 instances launched in a Public Subnet, they are auto-assigned a public IP address or ENI
  + Security groups and network ACLs on Public Subnet must allow SSH traffic (on port 22) for admin config.
* Private Subnets
  + outbound traffic is routed to a NAT device. The NAT device is installed in the Public Subnet and connected to an Internet Gateway for outbound access to the internet.
    - **NAT Gateway vs. NAT Instance** = NAT Gateway is managed for you by AWS and highly available, whereas NAT Instance is a lot more manual work but can be used as a bastion host / jump box​
  + EC2 instances don't have public IP or ENI
  + You have to use a bastion host ("jump box") to access instances in the Private Subnet over SSH (port 22)

**VPC Endpoints**

* **Interface endpoints***privately* connect your VPC to AWS services, services hosted by other AWS accounts, and supported AWS Marketplace services as if they were in your VPC
  + powered by **AWS Privatelink**
  + applies to many AWS services (API Gateway, CloudFormation, CloudWatch, S3)
  + does not go over the internet
  + no need to use an internet gateway, NAT device, DX connection, or VPN
  + Is an ENI with a private IP address, in the subnet that you specify, directing traffic to the service that you specify. Uses DNS to direct traffic to the service. Protected by a Security Group.
* **Gateway endpoints** direct traffic to S3 or DynamoDB only, using private IP addresses.
  + Does not enable AWS Privatelink
  + You route traffic from your VPC to the gateway endpoint using route tables. Protected by VPC endpoint policies rather than Security Groups.

**3 Types of Network Adapters**

1. ENI - basic type
2. ENA - for enhanced networking, high bandwidth and low latency
3. EFA (fabric adapter) - for high performance computing

**Security Groups vs. Network ACL (NACL)**

* Security Group is at the instance level, Network ACL is at the subnet level and applies to all instances within that subnet
* Security Groups don't have deny rules, Network ACL have accept and deny
* Security Groups are stateful, Network ACL stateless
* Security Groups evaluate all rules together, Network ACL processes rules in order
* Neither can block traffic by country
* Security Groups have inbound allow rules allowing traffic from within the group, whereas custom security groups don't allow any inbound traffic by default. All outbound traffic is allowed by default.
* Security Group default state: outbound rule allows all traffic to all IPs, but inbound has no rules and traffic therefore denied by default
* NACLs function at the subnet level with separate allow/deny rules for inbound and allow/deny rules for outbound. They are stateless so it's all about what the rules say each time.  Don't apply -within- the subnet, only in/iout of the subnet.
* Default security groups have inbound allow rules (from within the group). Custom security groups do not allow any inbound traffic. All outbound traffic is allowed.
* VPC automatically comes with a default NACL which allows all inbound/outbound traffic. A custom NACL denies all inbound/outbound traffic by default.

**Amazon Route 53**

* Geolocation routing is by location of the user, geoproximity routing is by proximity of the resources
* weighted routing = split traffic by %
* *Health check* = check the health of your resources and only return healthy resources in response to DNS queries
* apply a *routing policy* such as latency, weighted, failover
* Configurations
  + active/passive: in case of failure, return backup resource. Requires failover policy. Manual intervention can be required to then cause a fail-back to the active site.
  + active/active: return >1 resource. Requires latency policy, weighted policy, or some other policy besides failover. In the case of failover, returns only the healthy resource
  + combination: multiple policies are combined into a tree for more complex DNS failover

**Routing Records**

* Best practice is to use DNS names/URLs whenever possible rather than IP addresses. Some exceptions include pointing ELBs directly to the IP address of a peered VPC, or an on-prem resource linked via DX or VPN connection.
* [**Alias records**](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/resource-record-sets-choosing-alias-non-alias.html) provide a Route 53–specific extension to DNS functionality.
  + They let you route traffic to selected AWS resources: ELBs, APIs, CloudFront distributions, S3 buckets, Elastic Beanstalk, VPC interface endpoints, etc.
  + Unlike a CNAME record, they also let you route traffic from one record in a hosted zone (usually the *zone apex* / naked domain name, such as "example.com") to another record (e.g. "www.example.com")
  + When Route 53 receives a DNS query for an alias record, it responds with 1 or more IP addresses that the record maps to
* **CNAME records** (canonical name records) redirect DNS queries to any DNS record. For example, you can create a CNAME record that redirects queries from acme.example.com to zenith.example.com or acme.example.org.
  + You don't need to use Route 53.
  + Unlike Alias records, they can't be used for resolving apex domain names
* **PTR records** = reverse lookup where you map an IP address to a DNS name

**AWS Services Calling into a VPC**

* To enable AWS serverless services such as Lambda to access resources inside your private VPC, you provide it with VPC-specific info such as your subnet IDs and security group IDs.

**Elastic Load Balancers**

* ELBs send traffic to AWS and on-prem resources. Unlike Route 53, they use resource IP addresses and you don't get to specify policies such as a weighted policy. **VPC flow logs** show traffic going to/from an ELB
* An **Application Load Balancer (ALB)** makes routing decisions at the application layer aka Layer 7 (HTTP/HTTPS)
  + supports *path-based routing and host-based routing*(i.e. based on the content of the request in the host field)
  + can route requests to one or more ports on each ECS container instance in a cluster
  + supports authentication from OIDC compliant IDPs such as Google and Facebook via an integration with Cognito
  + periodically sends messages to its targets to check their status - *health checks*. - and routes only to healthy targets
  + enable *access logs*which can get pushed to S3. They log info on requester, IP, request type, etc.
* A **Network Load Balancer (NLB)** make routing decisions at the transport layer aka Layer 4 (TCP/SSL). They can handle millions of requests per second with extremely low latency. They don't support path-based routing or host-based routing the way ALB does.
* A **Classic Load Balancer (CLB)**operates using TCP, SSL, HTTP and HTTPS. Not as good at high throughput / low latency as NLB. Also unlike NLB, it does not support load balancing to multiple ports on an instance.

**AWS Direct Connect (DX) Gateway**

* You can use Direct Connect (DX) to connect an on-prem data centre to one or multiple VPCs
* DX can take > 1 month to setup
* For resilience, add a 2nd DX connection. As this can take time to setup and is costly, in the short term consider also adding an IPSec VPN connection (with the same BGP prefix) for resiliency.
* You must create one of the following **virtual interfaces** to begin using DX:
  + **Private virtual interface (private VIF)**: access a VPC using private IP addresses
  + **Public virtual interface (public VIF)**: access all AWS public services using public IP addresses
  + **Transit virtual interface (transit VIF)**: access one or more VPC Transit Gateways associated with DX gateways, within a Region.
* A **hosted virtual interface (hosted VIF)** allows another AWS account to access your DX
* Use **AWS DataSync**to copy large amount of data from on-prem to S3, EFS, FSx, NFS shares, SMB shares, AWS Snowcone (via Direct Connect).  For copying data, use DMS to copy databases.

**AWS Transit Gateway**

* Central Hub connecting on-prem networks and VPCs.
  + Reduces operational complexity as you can easily add more VPCs, VPN capacity, Direct Connect gateways, without complex routing tables.
  + Provides additional features over-and-above VPC peering
* A**transit virtual interface** is used to access VPC Transit Gateways
* Pattern for connecting 1 DX to multiple VPCs in the same Region is to associate the DX with a transit gateway
  + on-prem -> DX -> DX location -> transit virtual interface -> transit gateway association -> Transit Gateway -> multiple VPCs

**VPN Connection**

* VPN connections go over the internet
* **AWS Managed site-to-site VPN Connection**is connected between a **Customer Gateway**on the customer side and **Virtual Private Gateway (VPG, or** **VPN gateway)**that you create at the edge of your VPC.​

**AWS CloudFormation**

* provision infrastructure using a text-based template that describes exactly what resources are provisioned and their settings. Can use scripts to automate the creation of member accounts and VPCs.
* manages the template history similar to how code is managed in source control
* 2 methods of updating a stack
  1. *​direct update* - CloudFormation immediately deploys your changes
  2. *change sets* - preview your changes first, then decide if you want to deploy
* **​AWS SAM (Serverless Application Model)** is an extension of CloudFormation for packaging, testing and deploying serverless applications

**Disaster Recovery (DR)**

* DR approaches
  + Backup and restore = lowest cost, just create backups
  + Pilot Light = small part of core services that is running and syncing data or documents
  + Warm Standby = scaled down version of a fully functional environment that is actively running
  + Multi-site = on-prem and in AWS in an active-active configuration
* For disaster recovery in a different region, create a AMI from your EC2 instance and copy it into a 2nd region.

**STORAGE**  
  
**Instance Store**

* Block-level storage (with EBS disk that is physically attached to the host computer)
* Temporary/ephemeral, ideal for
  + temp info that changes frequently such as caches, buffers, scratch data,
  + data that is replicated across a fleet of instances where you can afford to lose a copy once in a while and the data is still replicated across other instances
* Very high performance and low latency
* Can be cost effective since the cost is included in the instance cost
* You can hibernate the instance to keep what's in memory and in the EBS, but if you stop or terminate the instance then you lose everything in memory and in the EBS storage.

**EBS**

* General Purpose SSD (gp2, gp3) - for low-latency interactive apps, dev&test environments.
  + Can have bursts of CPU performance but not sustained.
* Provisioned IOPS SSD (io1, io2) - for sub-millisecond latency, sustained IOPS performance.
  + Be sure to distinguish: IOPS solves I/O aka disk wait time, not CPU performance
  + IOPS is related to volume size, specifically per GB.
  + These are more $
* In contrast to SSD volumes, EBS also offers HDD volumes:
  + EBS Cold HDD (sc1) lowest cost option for infrequently accessed data and use cases like sequential data access
  + EBS Throughput Optimized HDD (st1) which is for frequent access and throughput intensive workloads such as MapReduce, Kafka, log processing, data warehouse and ETL workloads. Higher $ than sc1.
  + however note that the HDD volumes have no IOPS SLA.
* EBS can't attach to multiple AZs (there is a new EBS multi-attach feature but it's only single AZ, and only certain SSD volumes such as iop1, iop2). EBS is considered a "single point of failure".
* To implement a *shared storage layer*of files, you could replace multiple EBS with a single EFS
* Not fully managed, doesn't auto-scale (as opposed to EFS)
* Use **EBS Data Lifecycle Manager (DLM)** to manage backup snapshots. Backup snapshots are incremental, but the deletion process is design so that you only need to retain the most recent snapshot.
* iSCSI is block protocol, whereas NFS is a file protocol
* EBS supports encryption of data at rest and encryption of data in transit between the instance and the EBS volume.

**EFS**

* can attach to many instances across multiple AZ, whereas EBS cannot (there is a new EBS multi-attach feature but it's only single AZ, and only certain SSD volumes such as iop1, iop2)
* fully managed, auto-scales (whereas EBS is not)
* Linux only, not Windows!
* Since it is Linux, use POSIX permissions to restrict access to files
* After a period up to 90 days, you can transition unused data to EFS IA
* Protected by*EFS Security Groups* to control network traffic and act as firewall

**S3**

* durable (99.999999999%)
* a best practice is to enable versioning and MFA Delete on S3 buckets
* S3 lifecycle 2 types of actions:
  1. ​​transition actions (define when to transition to another storage class)
  2. expiration actions (objects expire, then S3 deletes them on your behalf)
* objects have to be in S3 for > 30 days before lifecycle policy can take effect and move to a different storage class.
* Intelligent Tiering automatically moves data to the most cost-effective storage
* Standard-IA is multi-AZ whereas One Zone-IA is not
* A *pre-signed URL* gives you access to the object identified in the URL (URL is made up of bucket name, object key, HTTP method, expiration timestamp). If you want to provide an outside partner with an object in S3, providing a pre-signed URL is a more secure (and easier) option than creating an AWS account for them and providing the login, which is more work to then manage and error-prone if you didn't lock down the account properly.
* You can't send long-term storage data directly to Glacier, it has to pass through an S3 first
* Accessed via API, if you want to access S3 directly it can require modifying the app to use the API which is extra effort
* Can host a static website but not over HTTPS. For HTTPS use CloudFront+S3 instead.
* Best practice: use IAM policies to grant users fine-grained control to your S3 buckets rather than using bucket ACLs
* Can use *multi-part upload* to speed up uploads of large files to S3

**Glacier**

* slow to retrieve, but you can use *Expedited Retrieval* to bring it down to just 1-5min.

**Amazon FSx**

* to replace Microsoft Windows file server
* can be multi-AZ
* supports DFS (distributed file system) protocol
* integrates with AD
* FSx for Lustre is for high-performance computing (HPC) - does not support Windows

**Amazon Aurora Global Database**

* for globally distributed applications. 1 DB can span multiple regions
* If too much read traffic is clogging up write requests, create an Aurora replica and direct read traffic to the replica. The replica serves as both standby instance and target for read traffic.
* "Amazon Aurora Serverless" is different from "Amazon Aurora" - it automatically scales capacity and is ideal for infrequently used applications.

**RDS**

* Transactional DB (OLTP)
* If too much read traffic is clogging up write requests, create an RDS read replica and direct read traffic to the replica. The read replica is updated *asynchronously.* Multi-AZ creates a read replica in another AZ and *synchronously* replicates to it
* RDS is a managed database, not a data store. Careful in some questions if they ask about migrating a data store to AWS, RDS would not be suitable.
* To encrypt an existing RDS database, take a snapshot, encrypt a copy of the snapshot, then restore the snapshot to the RDS instance. Since there may have been data changed during the snapshot/encrypt/load operation, use the AWS DMS (Database Migration Service) to sync the data.
* RDS can be restored to a backup taken as recent as 5min ago using point-in-time restore (PITR). When you restore, a new instance is created from the DB snapshot and you need to point to the new instance.

**ElastiCache**

* Database cache. Put in front of DBs such as RDS or Redshift, or in front of certain types of DB data in S3, to improve performance
* As a cache, it is an in-memory key/value store database (more OLAP than OLTP)
* *Redis* vs. *Memcached*
  + Redis has replication and high availability, whereas Memcached does not. Memcached allows multi-core multi-thread however.
  + Redis can be token-protected (i.e. require a password). Use the *AUTH command* when you create the Redis instance, and in all subsequent commands.
  + For Redis, *ElastiCache in-transit encryption* is an optional feature to increase security of data in transit as it is being replicated (with performance trade-off)
* Use case: accelerate autocomplete in a web page form

**DynamoDB**

* Use when the question talks about key/value storage, near-real time performance, millisecond responsiveness, and very high requests per second
* Not compatible with relational data such as what would be stored in a MySQL or RDS DB
* No concept of read replica like in RDS and Aurora. For read-heavy or bursty workloads, use DAX, an in-memory cache, to accelerate performance.
* DynamoDB measures RCUs (read capacity units, basically reads per second)  and WCUs (write capacity units)
* *DynamoDB auto scaling* uses the **AWS Application Auto Scaling** service to dynamically adjust throughput capacity based on traffic.
* Best practices:
  + keep item sizes small (<400kb) otherwise store in S3 and use pointers from DynamoDB
  + store more frequently and less frequently accessed data in different tables
  + if storing data that will be accessed by timestamp, use separate tables for days, weeks, months

**AWS Storage Gateway**

* Replace on-prem without changing workflow
* Types: File Gateway (for NFS and SMB), Volume Gateway, Tape Gateway.
* Stores data in S3 (e.g. for file gateway type, it stores files as objects in S3)
* Provides a cache that can be accessed at low latency, whereas EFS and EBS do not have a cache

**Copying and Converting**

* Use **AWS Schema Conversion Tool (SCT)**to convert a DB schema from one type of DB to another, e.g. from Oracle to Redshift
* Use **Database Migration Service (DMS)**to copy database. Sometimes you do SCT convert, then DMS copy.
* Use **AWS DataSync**to copy large amount of data from on-prem to S3, EFS, FSx, NFS shares, SMB shares, AWS Snowcone (via Direct Connect).  For copying data, not databases.

**Analytics** (OLAP)

* **Redshift** is a columnar data warehouse that you can use for complex querying across petabytes of structured data. It's not serverless, it uses EC2 instances that must be running. Use**Amazon RedShift Spectrum** to query data from S3 using a RedShift cluster for massive parallelism
* **Athena** is a serverless (aka inexpensive) solution to do SQL queries on S3 data and write results back. Works natively with client-side and server-side encryption. Not the same as QuickSight which is just a BI dashboard.
* **​Amazon S3 Select** - analyze and process large amounts of data faster with SQL, without moving it to a data warehouse

**SERVICES FOR ARCHITECTURE**  
  
**Amazon SQS**

* ideal for solutions that must be durable and loosely coupled
* pull-based (use SNS for pushing messages, especially broadcasting to multiple services)
* Standard vs. FIFO: FIFO is very rigorous whereas Standard is best-effort. The trade-off is that Standard has unlimited throughput of transactions per sec.
* Short polling vs. Long polling = time to wait before polling again
  + ​Short polling is the default. When you poll the SQS, it doesn't wait for messages to be available in the queue to respond. It checks a subset of servers for messages and may respond that nothing is available yet.
  + Long polling waits for a message to be in the queue before responding, so it uses fewer total requests and reduces cost.
* *batching* adds efficiency
* SQS doesn't prioritize items in the queue. If you need to prioritize use multiple queues, one for each priority type
* Max message size is 256kb (otherwise use S3 to log events), and max retention time of 14 days
* When a reader picks a message from the queue, the message stays in the queue but is invisible until the job is processed. If the *visibility timeout* occurs (job is not processed in time), then the message reappears in the queue for another reader to take.
* To use industry standards with Apache ActiveMQ, use an **Amazon MQ** instead of SQS (this is similar to using EKS instead of ECS, the industry-standard version of containers rather than the Amazon proprietary version)

**Amazon SNS**

* fully managed messaging service for pushing async notifications, especially used for broadcasting to multiple services

**Amazon Kinesis**

* for use cases that require ingestion of real-time data (e.g. IoT senor data)
* Kinesis data stream is made up of *shards*, which are made up of *data records*, which each have a *sequence #*. Then you map devices to partition keys which group data by shard.

**Amazon API Gateway**

* *Throttling limits*: you can configure a server-side throttling limit, a per-method throttling limit, a per-client throttling limit, and an account-level throttling limit.
* *API Caching for a STAGE* by specifying a TTL = time-to-live (by default 300 seconds).

**Amazon CloudFront**

* CloudFront distributes files from an origin. The origin can be an S3 bucket, EC2 instance, ELB, Route 53, or external.
* CloudFront+S3
  + S3 can host a static website but not over HTTPS. For HTTPS use CloudFront+S3 instead.
  + To prevent users accessing S3 content directly, create an *origin access identity (OAI)*which is a special CloudFront user and change S3 bucket permissions so that only the OAI can access. This is specific to CloudFront+S3.
* **Lambda@Edge** is a feature of CloudFront that lets you run code closer to users of your application, which improves performance and reduces latency
* *Field-level encryption*is a feature that applies extra encryption at edge locations to ensure sensitive data provided by the user (e.g. PII) is secured end-to-end
* Can be configured to load an error page ("content not found") for operationally simple error handling
* Not just for static content, CloudFront is used for streaming content too
* *Geo restriction* (whitelist/blacklist access to content by country, e.g. due to copyright restrictions)
* Set the price class to US, Canada, Europe, etc. to determine where the content will be cached
* To only allow specific IP addresses to access content, CloudFront can use signed URLs or signed cookies which include an expiration timestamp, and the range of IP addresses of users who can access the content.

**AWS Global Accelerator**

* increases availability and performance
* can be expensive
* runs over AWS global network
* directs traffic to optimal endpoints across multiple regions
* By default, provides you with 2 static IP addresses that are anycast from the AWS edge network. You can migrate existing IPv4 (/24) IPs rather than creating new.

**AWS STS (Security Token Service)**

* request temporary limited-privilege credentials for IAM users, or for users that you authenticate such as federated users from an on-prem directory
* *Federation*: STS can be used Federation (typically with Azure AD). It uses SAML 2.0 for authentication to grant temporary access based on the AD creds
* *Single Sign-On*: STS can be used to develop a custom identity broker for SSO to a service such as the AWS management console:
  1. Verify that the user is authenticated on the local IDP (AD)
  2. Call STS AssumeRole or GetFederationToken API to get temp credentials
  3. Pass the temp creds to AWS federation endpoint to request a sign-in token
  4. Construct a URL to the service that includes the token which can be provided to the user

**Amazon ECS**

* Container management service for Docker containers
* Highly scalable / high performance, lets you run applications on an EC2 cluster
* ECS Launch Types
  1. **Fargate Launch Type** is serverless, managed by AWS
  2. **EC2 Launch Type**gives you direct access to the instances, but you have to manage them
* ECS uses the *ECS Service Auto Scaling* (aka Application Auto Scaling) service to scale tasks using a scaling policy that you configure.
* ECS is about tasks. You pay for the running time of tasks. For example, you can't add container instances to an IAM group, you associate tasks with IAM roles/groups.

**SECURITY**  
  
**Encryption**

* Best practice is to lock away or delete the root user access keys. Never store in an S3 bucket, even if encrypted.
* Data at rest
  + Client-side encryption can be done by 1) using a customer master key (CMK) stored in KMS, or 2) using a master key that you store in your application. You can't use S3 managed keys client-side
  + Server-side encryption can be done in several ways
    - SSE-C: use customer-provided keys and manage them yourself (*on-prem*)
    - SSE-S3: Amazon manages the keys
    - SSE-KMS: keys are managed in Amazon Key Management Service
    - CloudHSM: generate and use your own encryption keys, held in the cloud in Amazon's HSM
* Data in motion
  + SSL/TLS is for encrypting data in transit, not data at rest.
  + SSL/TLS is synonymous with HTTPS traffic. It goes over port 443.

**Amazon GuardDuty**

* use with CloudWatch+SNS to trigger notifications to services

**IAM**

* The *permissions boundary* for an IAM entity (user or role) sets the max permissions that the entity can have

**AWS Accounts**

* To apply security restrictions across multiple AWS accounts, use Service Control Policy (SCP). For just a single account, use IAM policies.
* You can migrate an account to another AWS organization, e.g.  if you divest a business unit

**AWS CloudTrail**

* Audit trail of API calls
* Logs *Data Events*(resource operations) aka Data Plane Operations
* Logs*Management Events* (management operations on resources) aka Control Plane Operations
* Use other tools such as VPC Flow Logs to capture network packets

**MANY MORE AWS SERVICES**  
  
Although the AWS Cloud Practitioner certification is not a prerequisite for the AWS SAA exam, I found it very helpful to study for the AWS Cloud Practitioner, especially to learn the basics of the many AWS services. Check out my [**AWS Cloud Practitioner (ACP) Study Notes**](https://www.stellexgroup.com/blog/aws-acp-exam-cheatsheet) especially to review a fuller list of AWS services such as:

* **AWS App Mesh**= for application networking for microservices applications
* **AWS Resource Access Manager** = share a Transit Gateway connection (only?) with other AWS accounts
* **AWS Server Migration Service (SMS)** is for migrating virtual machines
* **AWS Step Functions** coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. Includes long-running executions not supported within Lambda execution limits.
* **AWS Elastic Beanstalk** is a PaaS service for describing and provisioning resources. Can be used to quickly deploy and manage applications in AWS. Developers upload applications and Beanstalk handles the deployment details. Note that it's not serverless, it relies on EC2 instances.
* **AWS Simple Workflow Service (SWF)** is for executing tasks. Helps developers build, run, and scale background jobs
* **AWS CodeStar** quickly develop, build and deploy applications on AWS
* **AWS Config** manage the config of AWS resources
* **AWS Batch** batch processing of computing jobs
* **Amazon Lex** builds conversational interfaces into an application using voice and text
* **AWS X-Ray** analyze, debug and improve performance of serverless applications
* **Amazon EMR** allows researchers, scientists, businesses, etc. to process vast amounts of data using hosted Hadoop running on EC2 and S3
* **AWS Import/Export** send HDDs with data to AWS and they import the data into S3
* **Amazon Connect**: call centre
* **Amazon SES**: Simple E-mail Service for sending marketing e-mails (like a Marketo or ConstantContact)
* **Amazon QuickSight**: BI
* **Amazon Elasticsearch Service**: "operational analytics" that you visualize
* **Amazon Neptune**: interactive graphs of DBs
* **AWS Config**: tracks resource inventory, config history and config change notifications for the purpose of security and compliance. Assess, audit and evaluate the configurations of AWS resources.
* **Amazon AppStream**: streaming service
* **Amazon Kinesis**: collect and process streaming data
* **Amazon Elastic Transcoder:** convert video and audio files into versions that play on phones, tablets and PCs
* **CloudSearch**: search engine for your site
* **CLI:**command-line interface. To use, it requires an IAM access key ID & secret access key
* **AWS LightSail**: easy alternative to setting up a VPC. Product set includes virtual servers (instances), MySQL DBs, HA storage and load balance
* **Amazon MSK**
* **AWS IoT Core**: connected devices interact securely with cloud applications
* **Amazon Cognito** = authentication for mobile devices. Use *identity pools* to provide temp AWS credentials to guest users. *User pools*are user directories. Compatible with SAML identity providers.