Elastic Beanstalk Cheat Sheet

- **Elastic Beanstalk** handles the deployment, from capacity provisioning, load balancing, auto-scaling to application health monitoring
- When you want to run a web-application but you don't want to have think about the underlying infrastructure.
- It costs nothing to use Elastic Beanstalk (only the resources it provisions eg. RDS, ELB, EC2)
- Recommended for test or development apps. Not recommended for production use
- You can choose from the following preconfigured platforms: Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker
- You can run containers on EB either in Single-container or Multi-container, these containers are running on ECS instead of EC2
- You can launch either a **Web Environmen**t or a **Worker Environment**
  - **Web Environments** come in two types **Single-Instance** or **Load Balanced**
    - **Single-Instance Env** launches a single EC2 instance, an EIP is assigned to the EC2
    - **Load Balanced Env** launch EC2s behind an ELB managed by an ASG
  - **Worker Environments** creates an SQS queue, install the SQS daemon on the EC2 instances, and has ASG scaling policy which will add or remove instances based on queue size.
- EB has the following **Deployment Policies:**
  - **All at once** takes all servers out-of-service, applies changes, puts servers back-in-service, Fast, has downtime
  - **Rolling** updates servers in batches, reduced capacity based on batch size
  - **Rolling with additional batch** adds new servers in batches to replace old, never reduces capacity
  - **Immutable** creates the same amount of servers, and switches all at once to new servers, removing old servers

Elastic Beanstalk Cheat Sheet

- Rollback deployment **polices require an ELB** so cannot be used with Single-Instance Web Environments.
- In-Place deployment is when deployment occurs within the environment, all deployment policies are In-Place.
- Blue/Green is when deployment swaps environments (outside an environment), When you have external resources such as RDS which cannot be destroyed its suited for Blue/Green
- **.ebextensions** is a folder which contains all configuration files
- With EB you can provide a **Custom Image** which can improve provisioning times
- If you let Elastic Beanstalk create the RDS instance, that means when you delete your environment it will delete the database. This setup is intended for development and test environments.
- **Dockerrun.aws.json** is similar to a ECS Task Definition files an defines multi container configuration.

ECS and Fargate Cheat Sheet

- **Elastic Container Service (ECS)** is fully-managed container orchestration service. Highly secure, reliable, and scalable way to run containers
- Components of ECS
  - **Cluster** Multiple EC2 instances which will house the docker containers.
  - **Task Definition** A JSON file that defines the configuration of (upto 10) containers you want to run
  - **Task** Launches containers defined in Task Definition. Tasks do not remaining running once workload is complete.
  - **Service** Ensures tasks remaining running eg. web-app.
  - **Container Agent** Binary on each EC2 instance which monitors, starts and stops tasks

Elastic Container Registry (ECR) A fully-managed Docker container registry that makes it easy for developers to **store**, **manage**, and **deploy** Docker container images.

- **Fargate** is **Serverless containers.** Don't worry about servers. Run containers, and pay based on duration and consumption
  - Fargate has **Cold Starts** so if this is an issue for you then use ECS.
  - **Duration** As long as you want
  - **Memory** Up to 30 GB
  - **Pricing** Pay at least 1 min and every additional second

X-Ray Cheat Sheet

- X-Ray **helps developers analyze and debug applications** that utilize **microservice** architecture
- X-Ray is a **Distributed Tracing System** , it is a method used to profile and monitor apps, especially those built using a microservices architecture to. pinpoint where failures occur and what causes poor performance.
- **X-Ray** Daemon is a software application that listens for traffic on UDP port 2000, gathers raw segment data, and relays it to the AWS X-Ray API. Data is generally not sent directly the X-Ray API and passes through the X-Ray Daemon which uploads in bulk
- **Segments** provides the resource's name, details about the request, and details about the work done.
- **Subsegments** provide more granular timing information and details about downstream calls that your app made to fulfill the original request.
- **Service Graph** is a flow chart visualization of average response for micro-services and to visually pinpoint failure
- **Traces** collects all segments generated by a single request so you can track the path of requests through multiple services
- **Sampling** is an algorithm that decides which requests should be traced. By **Default** X-RAY SDK **records the first request each second** and **5% of any additional requests**
- **Tracing Header** is named X-Amzn-Trace-Id and identifies a trace which passed along to downstream services
- **Filter Expression**s allows you to narrow down specific paths or users
- **Groups** allow save FilterExpressions so you can quickly filter traces.

X-Ray Cheat Sheet

- **Annotations and Metadata** allow you to capture additional information in key-value pairs
  - Annotations are **indexed** for use with filter expressions with a limit of 50
  - Metadata are **not indexed.** Use Metadata to record data you want to store in the trace but don't need to use for searching traces
- **Errors** are 400/ 4xx errors
- **Faults** are 400 /5xx errors
- **Throttle** is 429 Too Many Requests
- XRay supports the following **Languages**:
  - Go,NodeJs,Ruby,Java,Python,ASP.NET,Ruby,PHP
- Xray support **AWS Service Integrations** with the following:
- Lambda, API Gateway, App Mesh, CloudTrail, CloudWatch, AWS Config, EB, ELB, SNS, SQS, EC2. ECS, Fargate

Route53 CheatSheet

- Route53 is a DNS provider, register and manage domains, create record sets. Think Godaddy or NameCheap
- Simple Routing - Default routing policy, multiple addresses result in a random endpoint selection
- Weighted Routing - Split up traffic based on different 'weights' assigned (percentages)
- Latency-Based Routing - Directs traffic based on region, for lowest possible latency for users.
- Failover Routing - Primary site in one location, secondary data recovery site in another. (change on health check)
- Geolocation Routing - Route traffic based on the geographic location of a requests origin.
- Geo-proximity Routing - Route traffic based on geographic location using 'Bias' values (needs Route53 Traffic Flow)
- Multi-value Answer Routing - Return multiple values in response to DNS queries. (using health checks)
- Traffic Flow - visual editor, for chaining routing policies, can version policy records for easy rollback
- AWS Alias Record - AWS' smart DNS record, detects changed IPs for AWS resources and adjusts automatically.
- Route53 Resolver - Lets you regionally route DNS queries between your VPCs and your network **Hybrid Environments**
- Health checks can be created to monitor and automatically over endpoints. You can have health checks monitor other health checks

AWS CLI &  SDK *CheatSheet*

- **CLI** stands for Command Line Interface
- **SDK** stands for Software Development Kit
- The **AWS CLI** lets you interact with AWS from anywhere by simply using a command line
- The **AWS SDK** is a set of API libraries that let you integrate AWS services into your applications.
- **Programmatic Access** must be enabled per user via the IAM console to use CLI or SDK
- **aws configure** command used to setup your AWS credentials for the CLI
- The CLI is installed via a Python script
- Credentials get stored in a plain text file (whenever possible use roles instead of AWS credentials)
- The SDK is avaliable for the following programming languages
  - C++
  - Go
  - Java
  - Javascript
  - .NET
  - NodeJs
  - PHP
  - Python
  - Ruby

KMS Cheat Sheet

- **Key Management Service (KMS)** creates and manages encryption keys for a variety of AWS Services or for your apps.
- KMS can be used with CloudTrail to audit keys access history.
- KMS has the ability to automatically rotate out your keys ever year with no need to re-encrypt
- **Customer master keys** (CMKs) are the primary resources in KMS.
- KMS is a multi-tenant HSM, multi-tenant means you are sharing the hardware with multiple customers
- Hardware Security Module (HSM) is a specialized hardware for storing your keys and is tamper proof
- **KMS** is up to **FIPS 140-2 Level 2** compliant
- KMS stores Master Keys (not data keys)
- Master Keys are used to encrypt data keys which is called **Envelope Encryption**
- KMS supports two types of keys **symmetric** and **asymmetric**
  - **Symmetric** is a single key using 256 bit encryption eg. S3 bucket AES-256
  - **Asymmetric** uses two keys eg. key Pair public and private
- Important KMS APIs to remember:
  - **aws kms create-key** – creates a key
  - **aws kms encrypt** – encrypts a key
  - **aws kms decrypt** – decypts a key
  - **aws kms re-encrypt** – re-encrypts a key
  - **aws kms enable-key-rotation** – turn on automatic key rotation (only for **symmetric keys**)

SNS *CheatSheet*

- **Simple Notification Service (SNS)** is a fully managed pub/sub messaging service
- SNS is for **Application Integration**. It allows decoupled services and apps to communicate with each other
- **Topic** a logical access point and communication channel.
- A topic is able to deliver to multiple protocols
- You can encrypt topics via KMS
- **Publishers** use the AWS API via AWS CLI or SDK to push messages to a topic. Many AWS services integrate with SNS and act as publishers.
- **Subscriptions** subscribe to topics. When a topic receives a message it automatically and immediately pushes messages to subscribers.
- All messages published to SNS are stored redundantly across multiple Availability Zones (AZ)
- The following protocols:
    - **HTTPs and HTTPs** create webhooks into your web-application
    - **Email** good for internal email notifications (only supports **plain text**)
    - **Email-JSON** sends you json via email
    - **Amazon SQS** place SNS message into SQS queue
    - **AWS Lambda** triggers a lambda function
    - **SMS** send a text message
    - **Platform application endpoints** Mobile Push eg. Apple, Google, Microsoft Baidu notification systems

Simple Queue Service (SQS) *CheatSheet*

- SQS is a queuing service using messages with a queue. Think Sidekiq or RabbitMQ
- SQS is used for Application Integration, it lets decoupled services and apps to talk to each other
- To read SQS use need to **pull** the queue using the AWS SDK. SQS is **not pushed-based**
- SQS supports both Standard and First-In-First-Out (FIFO) queues
- Standard allows nearly unlimited messages per second, does not guarantee order of delivery, always delivers at least once, you must protect again duplicate messages being processed
- FIFO maintain the order of messages with a 300 limit
- There are two kinds of polling Short (Default) and Long Polling
- Short polling returns messages immediately, even if the message queue being polled is empty.
- Long polling waits until message arrives in the queue, or the long poll timeout expires.
- In majority of cases Long polling is preferred over short polling.
- **Visibility time-out** is the period of time that messages are invisible in the SQS queue
- Messages will be deleted from queue after a job has processed. (before visibility timeout expires)
- If Visibility timeout expires than a job will become visible to the queue
- The default Visibility time-out is 30 seconds. Timeout can be **0 seconds** to a maximum of 12 hours.
- SQS can retain messages from 60 seconds to 14 days and by default is 4 days
- Message size between 1 byte to 256 kb, Extended Client Library for Java can increase to 2GB

Kinesis *CheatSheet*

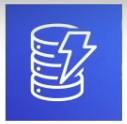
- **Amazon Kinesis** is the AWS solution for **collecting**, **processing**, and **analyzing** **streaming data** in the cloud. When you need **"real-time"** *think* Kinesis.
  Kinesis Data Streams Per per running shard, data can persist within the stream, data is ordered and every consumer keep its own position. Consumers have to be manually added (coded), Data persists for **24 hours (default)** to **168 hours**
- **Kinesis Firehose** - Pay for only the data ingested, data **immediately disappears** once processed. Consumer of choice is from a predefined set of services: S3, Redshift, Elasticsearch or Splunk
- **Kinesis Data Analytics** - allows you to perform **queries in real-time**. Needs a Kinesis Data Streams/Firehose as the input and output.
- **Kinesis Video Analytics** securely ingests and stores video and audio encoded data to consumers such as SageMaker, Rekognition or other services to apply Machine learning and video processing.
- KPL (Kinesis Producer Library) is a Java library to write data to a stream
- You can write data to stream using AWS SDK, but KPL is more efficient

Kinesis *CheatSheet*

- **Amazon Kinesis** is the AWS solution for **collecting**, **processing**, and **analyzing** **streaming data** in the cloud. When you need **"real-time"** *think* Kinesis.
  Kinesis Data Streams Per per running shard, data can persist within the stream, data is ordered and every consumer keep its own position. Consumers have to be manually added (coded), Data persists for **24 hours (default)** to **168 hours**
- **Kinesis Firehose** - Pay for only the data ingested, data **immediately disappears** once processed. Consumer of choice is from a predefined set of services: S3, Redshift, Elasticsearch or Splunk
- **Kinesis Data Analytics** - allows you to perform **queries in real-time**. Needs a Kinesis Data Streams/Firehose as the input and output.
- **Kinesis Video Analytics** securely ingests and stores video and audio encoded data to consumers such as SageMaker, Rekognition or other services to apply Machine learning and video processing.
- KPL (Kinesis Producer Library) is a Java library to write data to a stream
- You can write data to stream using AWS SDK, but KPL is more efficient

Ultimate DynamoDB Cheat Sheet

- **DynamoDB** is a fully managed **NoSQL** key/value and document database.
- DynamoDB is suited for workloads with any amounts of data that **require predictable read and write performance** and automatic scaling from large to small and everywhere in between.
- DynamoDB scales up and down to support whatever **read and write capacity you specify** per second in provisioned capacity mode or you can set it to On-Demand mode and there is little to no capacity planning.
- DynamoDB can be set to support **Eventually Consistent Reads (default)** and **Strongly Consistent Reads** on a per-call basis.
- **Eventually consistent reads** data is returned immediately but data can be inconsistent. Copies of data will be generally consistent in 1 second.
- **Strongly Consistent Reads** will always read from the leader partition since it always has an up-to-date copy. Data will never be inconsistent but latency may be higher. Copies of data will be consistent with a guarantee of 1 second.
- DynamoDB stores 3 copies of data on SSD drives across 3 AZs in a region.
- DynamoDB most common datatypes are **B** (Binary), **N** (Number), **S** (String)
- Tables consist of **Items** (rows) and items consist of **Attributes** (columns)
- A **Partition** is when DynamoDB slices your table up into smaller chunks of data, this speeds up reads for very large tables
- DynamoDB automatically creates Partitions for:
  - Every 10 GB of Data or
  - When you exceed RCUs (3000) or WCUs (1000) limits for a single partition
  - When DynamoDB sees a pattern of a hot partition, it will split that partition in an attempt to fix the issue

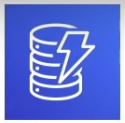


Ultimate DynamoDB Cheat Sheet

- DynamoDB will try to **evenly split** the RCUs and WCUs across Partitions
- Primary keys define **where and how** your data will be stored in partitions
- Primary key comes in two types:
  - **Simple** Primary Key (Using only a Partition Key)
  - **Composite** Primary Key (Using both a Partition and Sort Key)
- Partition Key is also known as **HASH**
- Sort Key is also known as **RANGE**
- When creating a Simple Primary Key the Partition Key value must be unique
- When creating a Composite Primary Key the combined Partition and Sort Key must be unique
- When using Sort key records on the partition are logically group together in Ascending order
- **DynamoDB Global tables** provide a fully managed solution for deploying multi-region, multi-master databases.
- DynamoDB supports transactions via the **TransactWriteItems** and **TransactGetItems** API calls
- **Transactions** let you query multiple tables at once and is an all-or-nothing approach (all API calls must succeed)
- **DynamoDB Streams** allows to setup a Lambda function triggered every time data is modified in a table to react to changes
  - Streams do not consume RCUs

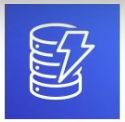
Ultimate DynamoDB Cheat Sheet

- DynamoDB has two types of Indexes:
  - **LSI** - Local Secondary index
    - Supports **strongly** or eventual consistency reads
    - can only be created with initial table (cannot be modified or and cannot deleted unless also deleting the table)
    - only Composite
    - 10GB or less per partition
    - Share capacity units with base table
    - Must share Partition Key (PK) with base table.
  - **GSI** - Global Secondary Index (cannot provide strong consistency)
    - Only eventual consistency reads
    - Can create, modify or delete at anytime
    - Simple and Composite
    - Can have whatever attributes as Partition Key (PK) or Sort Key (SK)
    - No size restriction per partition
    - Has its own capacity settings



Ultimate DynamoDB Cheat Sheet

- **Scan**
  - Your table(s) should be designed in such a way that your workload primary access patterns **do not use Scans**. Overall, scans should be needed sparingly. e.g. infrequent report.
  - Scans through all items in a table and then returns one more items through filters
  - By default returns all attributes for every item (use **ProjectExpression** to limit)
  - Scans are sequential, you can speed up a scan through parallel scans using **Segments** and **Total Segments**
  - Scans can be slow, especially with very large tables and can easily consume your provisioned throughput.
  - Scans are one of the most expensive ways to access data in DynamoDB.
- **Query**
  - Find items based on primary key values
  - Table must have a composite key in order to be able to query
  - By default queries are Eventually Consistent (use **ConsistentRead True** to change Strongly Consistent)
  - By default returns all attributes for each item found by a query (use **ProjectExpression** to limit)
  - By default is sorted ascending (use ScanIndexForward to False to reverse order to descending)



Ultimate DynamoDB Cheat Sheet

- DynamoDB has two capacity modes **Provisioned** and **On-Demand**.
- You can switch between these modes once every 24 hours.
- **Provisioned Throughput Capacity** is the maximum amount of capacity your application is allowed **to read or write per second** from a table or index
  - **Provisioned** is suited for predictable or steady state workloads
  - **RCUs** is Read Capacity Unit
  - **WCUs** is Write Capacity Unit
  - You should enable Auto Scaling with Provisioned capacity mode. In this mode, you set a floor and ceiling for the capacity you wish the table to support. DynamoDB will automatically add and remove capacity to between these values on your behalf and throttle calls that go above the ceiling for too long.
  - If you go beyond your provisioned capacity, you'll get an Exception: **ProvisionedThroughputExceededException** (throttling)
  - **Throttling** is when requests are blocked due to read or write frequency higher than set thresholds. E.g. exceeding set provisioned capacity, partitions splitting, table/index capacity mismatch.

Ultimate DynamoDB Cheat Sheet

- **On-Demand Capacity** is pay per request. So you pay only for what you use.
  - On-Demand is suited for **new** or **unpredictable** workloads
  - The throughput is only limited by the default upper limits for a table (40K RCUs and 40K WCUs)
  - Throttling can occur if you exceed double your previous peak capacity (high water mark) within 30 minutes. E.g. If you previously peaked to a maximum of 30,000 ops/sec, you could not peak immediately to 90,000 ops/sec, but you could to 60,000 ops/sec
  - Since there is no hard limit On-Demand could become very expensive based on emerging scenarios
- **Calculating Reads (RCU)**
  - A read capacity unit represents: one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size.
  - How to calculate RCUs for **strong**
    - Round data up to nearest 4.
    - Divide data by 4
    - Times by number of reads

50 reads at 40KB per item. (40/4) x 50 = 500 RCUs
10 reads at 6KB per item. (8/4) x 10 = 20 RCUs
33 reads at 17KB per item. (20/4) x 33 = 132 RCUs

  - How to calculate RCUs for **eventual**
    - Round data up to nearest 4.
    - Divide data by 4
    - Times by number of reads
    - Divide final number by 2
    - Round up to the nearest whole number

50 reads at 40KB per item. (40/4) x 50 / 2 = 250 RCUs
11 reads at 9KB per item. (12/4) x 11 / 2 = 17 RCUs
14 reads at 24KB per item. (24/4) x 14 / 2 = 35 RCUs

Ultimate DynamoDB Cheat Sheet

- **Calculating Writes (Writes)**
  - A write capacity unit represents: one write per second, for an item up to 1 KB
  - How to calculate Writes
    - Round data up to nearest 1.
    - Times by number of writes

50 writes at 40KB per item. 40 x 50 = 2000 WCUs
11 writes at 1KB per item. 1 x 11 = 11 WCUs
18 writes at 500 BYTES per item. 1 x 18 = 18 WCUs

- **DynamoDB Accelerator (DAX)** is a fully managed in-memory write through cache for DynamoDB that runs in a cluster
  - Reads are eventually consistent
  - Incoming requests are evenly distributed across all of the nodes in the cluster.
  - DAX is can reduce read response times to **microseconds**
  - **DAX is ideal for:**
    - fastest response times possible
    - apps that read a small number of items more frequently
    - apps that are **read intensive**
  - **DAX is not ideal for:**
    - Apps that require strongly consistent reads
    - Apps that do not require microsecond read response times
    - Apps that are **write intensive**, or that do not perform much read activity
    - If you don't need DAX consider using **ElastiCache**

DynamoDB Cheat Sheet

- DynamoDB notable commands API commands using the CLI eg. **aws dynamodb <command>**
  - **get-item** returns a set of attributes for the item with the given primary key. If no matching item, then it does not return any data and there will be no Item element in the response.
  - **put-item** Creates a new item, or replaces an old item with a new item. If an item that has the same primary key as the new item already exists in the specified table, the new item completely replaces the existing item.
  - **update-item** Edits an existing item's attributes, or adds a new item to the table if it does not already exist.
  - **batch-get-item** returns the attributes of one or more items from one or more tables. You identify requested items by primary key. A single operation can retrieve up to **16 MB of data**, which can contain as many as **100 items**.
  - **batch-write-item** puts or deletes multiple items in one or more tables. can write up to **16 MB of data**, which can comprise as many as **25 put or delete requests**. Individual items to be written can be **as large as 400 KB**.
  - **create-table** adds a new table to your account, table names must be unique within each Region
  - **update-table** Modifies the provisioned throughput settings, global secondary indexes, or DynamoDB Streams settings for a given table.
  - **delete-table** operation deletes a table and all of its items
  - **transact-get-items** is a synchronous operation that atomically retrieves multiple items from one or more tables (but not from indexes) in a single account and Region. Call can contain up to **25 objects**. The aggregate size of the items in the transaction **cannot exceed 4 MB**.
  - **transact-write-items** a synchronous write operation that groups up to **25 action requests**. These actions can target items in different tables, but not in different AWS accounts or Regions, and no two actions can target the same item.
  - **query** finds items based on primary key values. You can query table or secondary index that has a composite primary key
  - **Scan** returns one or more items and item attributes by accessing every item in a table or a secondary index.

EC2 CheatSheet

- **Elastic Compute Cloud (EC2)** is a Cloud Computing Service
- Configure your EC2 by choosing your **OS, Storage, Memory, Network Throughput.**
- Launch and SSH into your server **within minutes**.
- EC2 comes in variety Instance Types specialized for different roles:
    - **General Purpose** balance of compute, memory and networking resources
    - **Compute Optimized** Ideal for compute bound applications that benefit from high performance processor
    - **Memory Optimized** fast performance for workloads that process large data sets in memory.
    - **Accelerated Optimized** hardware accelerators, or co-processors
    - **Storage Optimized** high, sequential read and write access to very large data sets on local storage
- Instance Sizes **generally double** in price and key attributes
- **Placement Groups** let you to choose the logical placement of your instances to optimize for communication, performance or durability. Placement groups are free.
- **UserData** a script that will be automatically run when launching an EC2 instance.
- **MetaData** meta data about the current instance. You access this meta data via a local endpoint when SSH'd into the EC2 instance. eg. curl http://169.254.169.254/latest/meta-data meta data could be the instance type, current ip address etc...
- **Instance Profiles** a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts.

EC2 Auto Scaling Groups *CheatSheet*

- An ASG is a collection of EC2 instances grouped for scaling and management
- Scaling Out is when add servers
- Scaling In is when you remove servers
- Scaling Up is when you increase the size of an instance (eg. updating Launch Configuration with larger size)
- Size of an ASG is based on a **Min, Max** and **Desired Capacity**
- **Target Scaling policy** scales based on when a target value for a metric is breached eg. Average CPU Utilization exceed 75%
- **Simple Scaling** policy triggers a scaling when an alarm is breached
- **Scaling Policy with Steps** is the new version of Simple Scaling policy and allows you to create steps based on eculation alarm values.
- Desired Capacity is how many EC2 instances you want to ideally run
- An ASG will always launch instances to meet minimum capacity
- Health checks determine the current state of an instance in the ASG
- Health checks can be run against either an ELB or the EC2 instances
- When an Autoscaling launches a new instance it uses a Launch Configuration which holds the configuration values for that new instance eg. AMI, InstanceType, Role
- Launch Configurations cannot be edited and must be cloned or a new one created
- Launch Configurations must be manually updated in by editing the Auto Scaling settings.

VPC Endpoint *CheatSheet*

- VPC Endpoints help keep traffic between AWS services **within the AWS Network**
- There are two kinds of VPC Endpoints. Interface Endpoints and Gateway Endpoints
- Interface Endpoints **cost money**, Gateway Endpoints **are free**
- Interface Endpoints uses an Elastic Network Interface (ENI) with Private IP (powered by AWS PrivateLink)
- Gateway Endpoints is a target for a specific route in your route table
- Interface Endpoints support many AWS services
- Gateway Endpoint only support DynamoDB and S3

ELB *CheatSheet*

- There are three Elastic Load Balancers: **Network**, **Application** and **Classic** Load Balancer
- A Elastic Load Balancer must have **at least two** Availability Zones.
- Elastic Load Balancers **cannot go cross-region**. You must create one per region.
- ALB has **Listeners**, **Rules** and **Target Groups** to route traffic
- NLB use **Listeners** and **Target Groups** to route traffic
- CLB use **Listeners** and EC2 instances are **directly registered** as targets to CLB
- Application Load Balancer is for HTTP(S) traffic and the name implies it good for Web Applications
- Network Load Balancer is for TCP/UDP is good for high network throughput eg. Video Games
- Classic Load Balancer is legacy and its recommended to use ALB or NLB
- Use X-Forwarded-For (XFF) to get original IP of incoming traffic passing through ELB
- You can attach Web Application Firewall (WAF) to ALB but not to NLB or CLB
- You can attach Amazon Certification Manager SSL to any of the Elastic Load Balancers for SSL
- ALB has advanced Request Routing rules where you can route based on subdomain header, path and other HTTP(S) information
- Sticky Sessions can be enable for CLB or ALB and sessions are remembered via Cookie

Security Groups *CheatSheet*

- Security Groups acts as a firewall at the instance level
- Unless allowed specifically, all **inbound traffic** is **blocked by default**.
- All **Outbound traffic** from the instance is **allowed by default.**
- You can specific for the source to be either an IP range, single Ip Address or another security group
- Security Groups are **STATEFUL** (if traffic is allowed inbound it is also allowed outbound)
- Any changes to a Security Group take effect immediately.
- EC2 Instances can belong to multiple security groups
- Security groups can contain multiple EC2 Instances.
- You **cannot block specific IP addresses** with Security Groups, for this you would need a Network Access Control List (NACL)
- You can have upto 10,000 Security Groups per Region (default 2,5000)
- You can have 60 inbound and 60 outbound rules pre Security Group
- You can have 16 Security Groups associated to an ENI (default is 5)

NACLs *CheatSheet*

- Network Access Control List is commonly known as NACL
- VPCs are automatically given a default NACL which allows **all** outbound and inbound traffic.
- Each subnet within a VPC must be associated with a NACL
- Subnets can only be associated with 1 NACL at a time. Associating a subnet with a new NACL will remove the previous association.
- If a NACL is not explicitly associated with a subnet, the subnet will automatically be associated with the default NACL.
- NACL has inbound and outbound rules (just like Security Groups).
- Rule can either **allow** or **deny** traffic. (unlike Security Groups which can only allow)
- NACLs are STATELESS (incoming rule will not be applied to the outgoing)
- When you create a NACLs it will deny all traffic by default
- NACLs contain a numbered list of rules that gets evaluated in order from lowest to highest.
- If you needed to block a single IP address you could via NACLs (Security Groups cannot deny)

CloudFront *CheatSheet*

- CloudFront is a CDN (Content Distribution Network). It makes website load fast by serving cached content that is nearby
- CloudFront distributes cached copy at **Edge Locations**
- Edge Locations aren't just not read-only, you can write to them eg. PUT objects
- **TTL** (Time to live) defines how how long until the cache expires (refreshes cache)
- When you invalidate your cache, you are forcing it to immediately expire (refreshes cached data)
- Refreshing the cache **costs money because of transfer costs** to update Edge Locations
- **Origin** is the address of where the original copies of your files reside eg. S3, EC2, ELB, Route53
- **Distribution** defines a collection of Edge Locations and behaviour on how it should handle your cached content
- **Distributions** has 2 Types: **Web Distribution** (static website content) **RTMP** (streaming media)
- **Origin Identity Access (OAI)** is used access private S3 buckets
- Access to cached content can be protected via **Signed Urls** or **Signed Cookies**
- **Lambda@Edge** allows you to pass each request through a Lambda to change the behaviour of the response.

CloudTrail *CheatSheet*

- CloudTrail logs calls between AWS services
- **governance**, **compliance**, **operational auditing**, and **risk auditing** are keywords relating to CloudTrail
- When you need to know **who to blame** *think* CloudTrail
- CloudTrail by default logs event data for the past 90s days via **Event History**
- To track beyond 90 days you need to create **Trail**
- To ensure logs have not been tampered with you need to turn on **Log File Validation** option
- CloudTrail logs can be encrypted using **KMS (Key Management Service)**
- CloudTrail can be set to log across all AWS accounts in an Organization and all regions in an account.
- CloudTrail logs can be streamed to CloudWatch logs
- Trails are outputted to an S3 bucket that you specify
- CloudTrail logs two kinds of events: **Management Events** and **Data Events**
- **Management events** log management operations eg. AttachRolePolicy
- **Data Events** log data operations for resources (S3, Lambda) eg. GetObject, DeleteObject, and PutObject
- Data Events are **disabled** by default when creating a Trail.
- Trail logs in S3 can be analyzed using Athena

IAM *CheatSheet*

- **Identity Access Management** is used to manage **access** to users and resources
- IAM is a universal system. (applied to all regions at the same time). IAM is a free service
- A root account is the account initially created when AWS is set up (full administrator)
- New IAM accounts have no permissions by default until granted
- New users get assigned an Access Key Id and Secret when first created when you give them programmatic access
- Access Keys are only used for CLI and SDK (cannot access console)
- Access keys are only shown once when created. If lost they must be deleted/recreated again.
- Always setup MFA for Root Accounts
- Users must enable MFA on their own, Administrator cannot turn it on for each user
- IAM allows your set password policies to set minimum password requirements or rotate passwords
- **IAM Identities** as Users, Groups, and Roles
- **IAM Users** End users who log into the console or interact with AWS resources programmatically
- **IAM Groups** Group up your Users so they all share permission levels of the group
- eg. Administrators, Developers, Auditors
- **IAM Roles** Associate permissions to a Role and then assign this to an Users or Groups
- **IAM Policies** JSON documents which grant permissions for a specific user, group, or role to access services. Policies are attached to to IAM Identities
- **Managed Policies** are policies provided by AWS and cannot be edited
- **Customer Managed Policies** are policies created by use the customer, which you can edit
- **Inline Policies** are policies which are directly attached to a user

CI/CD Cheat Sheet

- CI/CD is automated methodologies **that prepare, test, deliver or deploy code** onto a servers
- **Production** (prod) an environment which is intended to be used by paying users
- **Staging** an environment which is intended to simulate a production environment for last stage debugging
- Continuous Integration (CI)
  - Automating the review of developer's code.
  - eg. Run test suites with a build server to eg. CodeBuild
- Continuous Delivery (CD)
  - Automating the preparation of the developer code for release
  - Eg. Run test suites with a build server to eg. CodeBuild and if tests pass create pull request or merge branch into staging
- Continuous Deployment (CD)
  - Automatically deploying developers code which ready for release
  - Eg. Automatically merge pull requests will all tests passing and deploying into production.
  - Continuous Deployment can refer to the entire pipeline eg. CodePipeline using CodeCommit + CodeBuild + CodeDeploy

CodeBuild Cheat Sheet

- **CodeBuild** is a fully-managed **build pipeline** to create temporary servers to build and test code
- Compile source code, runs unit tests, and produces artifacts that are ready to deploy.
- Provides prepackaged build environments or you can build your own environments as a Docker container
- Uses a **Buildspec.yml** to provide build instructions. This file is stored in the root of your project.
  - Version 0.1 – runs each build command in a separate instance
  - Version 0.2 – runs all build commands in the same instance
  - Commands run through different **phases:**
    - **Install** only for installing packages in the build env
    - **pre_build** commands that run before building
    - **build** commands that you run during the build
    - **post_build** commands run after the build

CloudFormation Cheat Sheet

- When being asked to **automate** the provisioning of resources *think* CloudFormation
- When Infrastructure as Code (IaC) is mentioned *think* CloudFormation
- CloudFormation can be written in either JSON or YAML
- When CloudFormation encounters an error it will rollback with **ROLLBACK_IN_PROGRESS**
- CloudFormation templates larger than 51,200 bytes (0.05 MB) are too large to upload directly, and must be imported into CloudFormation via an S3 bucket.
- **NestedStacks** helps you break up your CloudFormation template into smaller reusable templates that can be composed into larger templates
- **At least one resource** under resources: must be defined for a CloudFormation template **to be valid**
- CloudFormation **Template Sections**
  - **MetaData** extra information about your template
  - **Description** a description of what the template is suppose to do
  - **Parameters** is how you get user inputs into templates
  - **Transforms** Applies marcos (like applying a mod which change the anatomy to be custom)
  - **Outputs** are values you can use to import into other stacks
  - **Mappings** maps keys to values, just like a lookup table
  - **Resources** defines the resources you want to provision, **at least one resource is required**
  - **Conditions** are whether resources are created or properties are assigned

CloudFormation Cheat Sheet

- Stack Updates can be preformed two different ways:
  - Direct updates
    - You directly update the stack
    - You submit changes and AWS CloudFormation immediately deploys them
  - Executing Change Sets
    - You can preview the changes to CloudFormation will make to your stack (Change Set)
    - Then decide whether you want to apply those changes
  - Stack Updates will change state of your resources based on circumstances:
    - **Update with No Interruption** Updates the resource **without disrupting** operation and **without changing** the resource's physical ID
    - **Updates with Some Interruptions** Updates the resource **with some interruption** and **retains** the physical ID.
    - **Replacement** **recreates** the resource during an update, also **generates new** physical ID.
    - You can use a **StackPolicy** to prevent stack updates on resources to prevent data loss or interruption to services
  - **Drift Detection** feature lets CloudFormation tell you when your expected configuration has changed to due manual overrides. eg. A CFN creates an SG but a Developer deletes it.

CloudFormation Cheat Sheet

- **Rollbacks** occur when a CloudFormation encounters an error when you create, update or destroy a stack
  - When a rollback is in progress you'll see **ROLLBACK_IN_PROGRESS**
  - When a rollback succeeds you'll see **UPDATE_ROLLBACK_COMPLETE**
  - When a rollback fails you'll see **UPDATE_ROLLBACK_FAILED**
- **Pseudo Parameters** are predefined parameters eg. !Ref AWS:Region return us-east-1
- Resource Attributes
  - **CreationPolicy** – prevent its status from reaching create complete until AWS CloudFormation receives a specified number of success signals or the timeout period is exceeded.
  - **DeletionPolicy** – reserve or (in some cases) backup a resource when its stack is deleted **Delete**, **Retain** or **Snapshot**
  - **UpdatePolicy** – How to handle an update for ASG, ElastiCache, Domain or Lambda Alias
  - **UpdateReplacePolicy** – To retain or (in some cases) backup the existing physical instance of a resource when it is replaced during a stack update operation. **Delete**, **Retain** or **Snapshot**
  - **DependsOn** That resource is created only after the creation of the resource specified in the DependsOn attribute
- **Intrinsic Functions** allow you to assign properties that are not available during runtime most important two to know:
  - **Ref** returns the value of the specified parameter or resource
  - **Fn:GetAttr** returns the value of an attribute from a resource in the template
- aws cloudformation **create-stack –** CLI command to create a stack
- aws cloudformation **update-stack –** CLI command to update a stack
- **Serverless Application Model (SAM)** is an **extension** of CloudFormation that lets you define serverless applications

RDS *CheatSheet*

- Relational Database Service (RDS) is the AWS Solution for relational databases.
- RDS instances are managed by AWS, You cannot SSH into the VM running the database.
- There are 6 relational database options currently available on AWS, Aurora, MySQL, MariaDB, Postgres, Oracle, Microsoft SQL Server
- Multi-AZ is an option you can turn on which makes an exact copy of your database in another AZ that is only standby
- For Multi-AZ AWS automatically synchronizes changes in the database over to the standby copy
- Multi-AZ has Automatic Failover protection if one AZ goes down failover will occur and the standby slave will be promoted to master
- Read-Replicas allow you to run **multiples copies** of your database, these copies only allows **reads** (no writes) and is intended to alleviate the workload of your primary database to improve performance
- Read-Replicas use Asynchronous replication
- You must have automatic backups enabled to use Read Replicas

RDS CheatSheet

- You can have upto 5 read replicas
- You can combine Read Replicas with Multi-AZ
- You can have Read Replicas in another Region (Cross-Region Read Replicas)
- Replicas can be promoted to their own database, but this breaks replication
- You can have Replicas of Read Replicas
- RDS has 2 backup solutions: Automated Backups and Database Snapshots
- Automated Backups, you choose a retention period between 1 and 35 days, There is no additional cost for backup storage, you define your backup window
- Manual Snapshots, you manually create backups, if you delete your primary the manual snapshots will still exist and can be restored
- When you restore an instance it will create a new database. You just need to delete your old database and point traffic to new restored database
- You can turn on encryption at-rest for RDS via KMS

S3 CheatSheet

- **Simple Storage Service** (S3) Object-based storage. Store **unlimited** amount of data without worry of underlying storage infrastructure
- S3 replicates data across at least 3 AZs to ensure 99.99% Availability and 11' 9s of durability
- Objects contain your data (they're like files)
- Objects can be size anywhere from **0 Bytes** up to 5 Terabytes
- Buckets contain objects. Buckets can also contain folders which can in turn can contain objects.
- Bucket names are unique across all AWS accounts. Like a domain name.
- When you upload a file to S3 successfully you'll receive a HTTP 200 code
  Lifecycle Management Objects can be moved between storage classes or objects can be deleted automatically based on a schedule
- **Versioning** Objects are giving a Version ID. When new objects are uploaded the old objects are kept. You can access any object version. When you delete an object the previous object is restored. Once Versioning is turned o it cannot be turn off, only suspended.
- **MFA Delete** enforce DELETE operations to require MFA token in order to delete an object. Must have versioning turned on to use. Can only turn on MFA Delete from the AWS CLI. Root Account is only allowed to delete objects
- All new buckets are **private by default**
- Logging can be turned to on a bucket to log to track operations performed on objects
- **Access control** is configured using **Bucket Policies** and **Access Control Lists (ACL)**
- **Bucket Policies** are JSON documents which let you write complex control access
- **ACLs** are the legacy method (not deprecated) where you grant access to objects and buckets with simple actions

S3 CheatSheet

- **Security in Transit** Uploading files is done over SSL
- **SSE** stands for Server Side Encryption. S3 has **3 options** for SSE.
- **SSE-AES** S3 handles the key, uses AES-256 algorithm
- **SSE-KMS** Envelope encryption via AWS KMS and you manage the keys
- **SSE-C** Customer provided key (you manage the keys)
- **Client-Side Encryption** You must encrypt your own files before uploading them to S3
- **Cross Region Replication (CRR)** allows you to replicate files across regions for greater durability. You must have versioning turned on in the source and destination bucket. You can have CRR replicate to bucket in another AWS Account
- **Transfer Acceleration** provide faster and secure uploads from anywhere in the world. data is uploaded via distinct url to an Edge Location. Data is then transported to your S3 bucket via AWS backbone network.
- **Presigned Urls** is a url generated via the AWS CLI and SDK. It provides temporary access to write or download object data. Presigned Urls are commonly used to access private objects.

S3 CheatSheet

- S3 has **6 different** Storage Classes:
  - **Standard** Fast! 99.99% Availability, 11 9's Durability. Replicated across at least three AZs
  - **Intelligent Tiering** Uses ML to analyze your object usage and determine the appropriate storage class. Data is moved to the most cost-effective access tier, without any performance impact or added overhead.
  - **Standard Infrequently Accessed (IA)** Still Fast! Cheaper if you access files less than once a month. Additional retrieval fee is applied. 50% less than Standard (reduced availability)
  - **One Zone IA** Still Fast! Objects only exist in one AZ. Availability (is 99.5%). but cheaper than Standard IA by 20% less (Reduce durability) Data could get destroyed. A retrieval fee is applied.
  - **Glacier** For long-term cold storage. Retrieval of data can take minutes to hours but the off is very cheap storage
  - **Glacier Deep Archive** The lowest cost storage class. Data retrieval time is 12 hours.

API Gateway *CheatSheet*

- API Gateway is a solution for creating secure APIs in your cloud environment at any scale.
- Create APIs that act as a front door for applications to access data, business logic, or functionality from back-end services.
- API Gateway throttles api endpoints at **10,000** requests per second (can be increase via service request through AWS support)
- **Stages** allow you to have multiple published versions of your API eg. prod, staging, QA
- Each Stage has an **Invoke URL** which is the endpoint you use to interact with your API
- You can use a custom domain for your Invoke URL eg. api.exampro.co
- You need to publish your API via Deploy API. You choose which Stage you want to publish your API
- Resources are your URLs eg. /projects
- Resources can have child resources eg. /projects**/-id-/edit**
- You defined multiple Methods on your Resources eg GET, POST, DELETE
- CORS issues are common with API Gateway, CORS can be enabled on all or individual endpoints
- Caching improves latency and reduces the amount of calls made to your endpoint
- Same Origin Policies help to prevent XSS attacks
- Same Origin Policies ignore tools like postman or curl
- CORS is always enforced by the client.
- You can require Authorization to your API via AWS Cognito or a custom Lambda.

ElastiCache *CheatSheet*

- **ElastiCache** is a managed **in-memory** caching service
- ElastiCache can launch either **Memcached** or **Redis**
- **Memcached** is a simple key / value store preferred for caching HTML fragments and is arguably faster than Redis
- **Redis** has richer data types and operations. Great for leaderboard, geospatial data or keeping track of unread notifications.
- A cache is a **temporary storage** area.
- Most frequently identical queries are stored in the cache
- Resources only **within the same VPC** may connect to ElastiCache to ensure low latencies.

Lambda CheatSheet

- **Lambda's** are serverless **functions.** You upload your code and it runs without you managing or provisioning any servers.
- Lambda is **serverless**. You don't need to worry about underlying architecture
- Lambda is a good fit for short running tasks where you don't need to customize the os environment. If you need long running tasks (> 15mins) and a custom OS environment than consider using **Fargate**
- There are **7 runtime language environments** officially supported by Lambda: **Ruby, Python, Java, NodeJs, C#, Powershell** and **Go**
- You pay per invocation (The **duration** and the amount of **memory** used) rounded up to the nearest 100 milliseconds and you based on amount of requests. First 1M requests per month are free
- You can adjust the duration timeout for up to **15 mins** and memory up to **3008 MB**
- You can trigger Lambdas from the SDK or multiple AWS services eg. S3, API Gateway, DynamoDB
- Lambdas by default run in No VPC. To interact with some services you need to have your Lambda in the same VPC eg. RDS
- Lambda can scale to **1000 of concurrent functions** in seconds. (1000 is the default, you can increase with AWS Service Limit Increase)
- Lambdas have **Cold Starts.** If a function has not been recently been execute there will be a delay

Step Functions Cheat Sheet

- **Step Functions** coordinate multiple AWS services into serverless workflows using **State Machines**.
- A state machine is an abstract model which decides how one state moves to another based on a series of conditions. **Think of a state machine like a flow chart.**
- Step Functions automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected, every time
- **Amazon States Language** is how you define all your states and is written in **JSON**.
- There are two types of State Machines available:
  - **Standard** – general purpose, long workloads
  - **Express** – streaming data, short workloads
- Use Cases for Step Functions:
  - Manage a batch job, Fargate or ECS container
  - Sequential batch processing
  - Creating a pipeline to transcode media files
  - Transferring data records, by performing operations on each record queued up via SQS
  - Sequence steps of ML workflows
  - Coordinate Extract, Transform and Load (ETL) Jobs

Step Functions Cheat Sheet

- **The type of Step Functions States**
  - **Pass State** Passes its input to its output, without performing work (dummy/mock). Pass states are useful when constructing and debugging state machines.
  - **Task State** represents a single unit of work performed by a state machine.
    - **Lambda** – triggers a lambda function
    - **Activity** - the work is performed by a worker that can be hosted on anywhere eg. EC2, ECS, mobile phones
      Supported AWS Services pass parameters to the API of an AWS Service
  - **Choices State** adds branching logic to a state machine.
  - **Wait State** delays the state machine from continuing for a specified time.
  - **Succeed State** stops an execution successfully
  - **Fail State** stops the execution of the state machine and marks it as a failure
  - **Parallel State** can be used to create parallel branches of execution in your state machine The state machine does not move forward until both states complete
  - **Map State** can be used to run a set of steps for each element of an input array.