DOMAIN 1: COLLECTION (Moving data into AWS)

- In real time (for immediate actions): Kinesis, SQS, IoT or Near real time (for reactive actions): Firehose, DMS
- Batch or historical analysis: Snowball, Data Pipeline
- Less than 10Mbps network connection with less than 500GB: unmanaged (CLI or console)
- More than 10Mbps network connection with greater than 500GB: managed (actual AWS server)

Kinesis

- Managed alternative to Kafka for real time data gathering: get streaming data into AWS
  - Data is automatically replicated synchronously to 3 AZs
- Kinesis Streams ingests data from sources, passes to Kinesis Analytics for processing, to Firehose for storage
  - Use CW subscription filter to stream logs to the destinations: Kinesis Streams, Firehose, Lambda
- Producers send data to Kinesis: apps/clients using SDK library, KPL or using Kinesis agent installed on the server
  - SDK allows writing code and using CLI to directly send data into data streams
    - API for PutRecord(s) to use batching: increases throughput by sending many records into 1 HTTP request (less requests in total)
    - Exceed limit -> ProvisionedThroughputExceeded error: sending more data than can be handled (exceeding the number of MB/seconds or the number of records/seconds send to the shard)
      - Ensure no hard shard: if partition key is bad, then too much data is going through this partition
      - Ensure partition key is as distributed as possible
      - Solution: retry with backoff and increase the number of shards through scaling
    - Good for: low throughput use cases, don't mind higher latency, want simple API or working straight from a Lambda function or if want to send data right away
    - Managed AWS sources for data streams use SDK behind the scenes: CW Logs, IoT, Kinesis Analytics
  - KPL (Kinesis Producer Library) is more advanced but has features that allow enhanced throughput to streams
    - It is a Java library to build high performance, long running producers with automatic retry mechanism
    - Configurable retry mechanism: additional processing delays can occur for higher packing efficiencies
    - Compression must be implemented by user and batching helps increase throughput / decrease the cost:
      - Use batch compression before sending data to maximize throughput
      - Collect records and write to multiple shards in the same PutRecords API call
      - Aggregate to increase latency: store multiple records into one to increase payload / improve throughput
    - Write to Kinesis Streams via commands to provide a layer of abstraction for ingesting data
    - Two APIs: synchronous API (SDK) or asynchronous API for better performance
    - KPL records need to be decoded with KCL or a special helper library
    - KPL Batching aggregates many small records into 1 larger record that is less than 1MB:
      - Aggregates many records into a collection of records in the buffer so only need 1 API call
      - Willing to wait a specified time (adds latency) but is is more efficient
      - KPL can occur an additional processing delay of up to RecordMaxBufferTime within the library
        - Larger values result in high packing efficiency and better performance
    - Do not use if application cannot tolerate an additional delay
  - Kinesis Agent is a Linux program that runs on servers and allows log file to get reliable sends into streams:
    - It is a pre built, Java based agent build on top of KPL and installed on Linux based server environments
    - Monitors files to continuously send data to streams: handles file rotation, checkpointing, retry upon failures
    - Write from multiple directories, write to multiple streams: routing feature based on directory and log file
    - Emits metrics to CW for monitoring: almost real time aggregation of a mass of log files
  - 3rd Party Libraries that build off of SDK (Spark, Kafka) to send data to Kinesis Data Streams
- Consumers consume data from Kinesis Streams: apps using KCL or SDK, Lambda, Kinesis Analytics or Firehose
  - SDK/CLI: records pulled from shard when the shard gets a max of 2MB total of aggregate throughput
    - GetRecord(s) returns up to 10MB of data, then throttles for 5 seconds before doing another GetRecord(s)
      - Returns a maximum of 10,000 records
    - Maximum of 5 GetRecords API calls per shard per second to get 200 ms of latency
      - If 5 consumer applications pull from the same shard, then every consumer can poll once a second and receive less than 400 Kb/second (more consumers, less throughput per consumer)
  - KCL (Kinesis Consumer Library) reads records from Kinesis produced by KPL with de-aggregation mechanism
    - Share multiple shards with multiple consumers in one group so there is a shard discovery process
    - Checkpointing feature to resume progress: leverages DynamoDB for coordination (one row per shard but must provision enough WCU/RCU in Dynamo or use on demand to avoid throttling exceptions)
      - ExpiredIteratorException -> increase DynamoDB WCU, table is not fast enough to keep up with writes
  - Kinesis Connector Library: older library that uses KCL to write data to S3, Dynamo, RedShift, ElasticSearch
    - If an application needs to run on EC2 instances to take data from Kinesis and sends to all destinations
  - Lambda reads real time records from stream: it has a small consumer library to de-aggregate record from KPL
    - Mainly used for lightweight ETL: configurable batch size (how much data to read from Kinesis)

Kinesis Data Streams

- Low latency streaming ingestion at scale: collect and process streams of data records in real time
  - Aggregate real time data and load it into some data warehousing solution or compute cluster
  - Durable and elastic so won't lose data records: data retention period is between 1 to 365 days
  - Data scales up or down depending on the amount of records coming into the stream
  - Ability to reprocess or replay data and have multiple consumers
  - Once data is inserted into Kinesis, it cannot be deleted so it is immutable
  - Data can be consumed many times and is deleted after retention period
  - Ordering of records is preserved at shard level (even during replays)
  - Build multiple applications reading from same stream independently (publish/subscribe)
- Write custom code for producer and consumer: Must manage scaling (shard splitting and merging) by ourselves
  - Real time: 200 ms latency for classic, 70 ms latency for enhanced fan out
  - Use with Lambda to insert data real time to ElasticSearch
- A data stream is made up of multiple numbered shards that must be provisioned ahead of time:
  - Shards define stream capacity in terms of infection and consumption: default limit is 500 shards
    - Shards are containers that hold the information shipped off to consumers: consists of data records, where each data record has a partition key, sequence ID and the actual data (up to 1MB) to send
      - Partition key is same for all data records within a shard: defines and helps determine where shard goes
      - Sequence ID is the order in which the shard received the records
      - Data on a shard is stored for 24 hours by default but this can be changed to:
        - 7 days by enabling extended data retention
        - 365 data by enabling long term data retention
    - Scaling up shards if there are more records being input or more user activity
    - Each shard has a sequence of data records that can be ingested at 1000 records per second
      - 2 shards can ingest 2000 records per second
      - 1 shard can process 1MB per second of input data or 2MB per second of output data
      - Consumers consume data faster than we can actually put data into the shard
- Producers send data into Kinesis Data Streams:
  - All producers are apps producing streaming data like log files, user interactions, IoT devices, EC2 instances
  - Producers send data into Kinesis at a rate of 1MB/second or 1000 messages per second per shard
    - 6 shards gives 6MB/second or 6000 messages/second
  - Data sent to data streams by producers remains on the stream until it is read by consumers
- Consumers are any kind of apps that consumes data from within Kinesis Data Streams:
  - Consumers receives a record, partition key, sequence ID -> looks at record location in shard and data block
  - There are different consumption modes:
    - 2MB/second/shard of throughput shared for all consumers
    - 2MB/second/shard/consumer if enabling enhanced fan out
  - There are 10 consumer apps consuming concurrently from 1 shard in classic mode by issuing GetRecords command: the average latency for consuming these records for each app is 200ms (2 seconds)
    - Issue up to 5 GetRecords API calls/second so it takes 2 seconds for each consuming app before next call
  - There are 10 consumer apps consuming concurrently from 1 shard in enhanced fan out mode: the average latency for consuming these records for each app is 70 ms
    - No matter the number of consumers: each consumer receives 2MB/second of throughput
- Standard Consumer: low number of consumer applications to minimize costs
  - Tolerate 200m or more of latency
- Enhanced Fan Out Consumer: for multiple consumer applications for the same stream (Works with KCL, Lambda)
  - Low latency (70ms), higher costs, default limit of 5 consumers per stream
  - Each consumer gets 2MB of provisioned throughput/second/shard: each shard sends consumer 20MB of data
  - 20 consumers get 40MB/second/shard aggregated
  - Benefits: scales more consumer apple, no MB/second limit, reduced latency
- Capacity Mode:
  - Provisioned / Historic Mode: Choose the number of shards provisioned and scale them manually / using API
    - Each shard gets 1MB/second or 1000 records/second for inward throughput
    - Each shard gets 2MB/second for outward throughput (applies to Classic or Fan Out Consumer)
    - Pay per share provisioned per node
  - On Demand Mode: No need to provision or manage capacity so it is adjusted over time on demand
    - Default capacity provisioned is 4MB/second or 4000 records/second
    - Auto scaling is based on observed throughput peak during last 30 days
    - Pay per stream per hour for data in/out per GB

- Scaling:
  - Adding shards (shard splitting) to increase stream capacity and divide a hot shard to increase throughput
    - When splitting a shard, the old shard is closed / deleted when data within it is expired
  - Merging shards to decrease stream capacity: group 2 shards with low traffic
    - Old shards are closed and deleted based on data expiration
    - Modularize data stream to increase/decrease throughput over time
  - Autoscaling is not a native feature of Kinesis: API call to change the number of shards is updateShardCount
  - Limitations: resharding cannot be done in parallel (need to plan capacity in advance) and it takes time
    - Perform 1 resharding operation at a time so it takes a few seconds
      - For 1000 shards, it takes 30,000 seconds to double the shards to 200
    - Cannot scale more than 10x for each rolling 24 hour period for each stream
    - Cannot scale up to more than double your current shard count for a stream
    - Cannot scale down below half your current shard count for a stream
    - Cannot scale up to more than 50 shards in a stream
    - Cannot scale up to more than the shard limit for your account
- To increase capacity of Kinesis Data Streams, use the on-demand mode
- Security: Deployed within a region so control access by authorizing who can read from shards using IAM policies
  - Encryption in flight using HTTPs and at rest using KMS
  - VPC endpoints are available for Kinesis to be accessed from within a VPC
    - Supported interface for VPC endpoints / private link to access privately
  - Monitor API calls using CloudTrail
  - SSL endpoints using HTTPs protocol for encryption in flight (KMS provides SSE)
  - For client side encryption (data is encrypted before being sent to Kinesis), use your own encrypted libraries
  - KCL needs read / write access to DynamoDB table

Kinesis Video Streams
- Captures and stores videos, incorporated with image and voice recognition
  - Processes real time streaming video data that can be fed into other AWS services
- Producers use video stream libraries installed onto the devices to connect to video stream applications
- Videos can be streamed in real time and have a buffer to delay the service or be after the media uploads:
  - Software installed on the devices extracts videos or data in the frames to upload them to video stream service
  - After the frames are in the video stream service, use different consumers to get the data (fragments of the frame) and view, process, analyze it in real time (or after it has been stored in S3)

Kinesis Firehose
- Fully managed, near real time service to send streams into S3, Redshift, ElasticSearch, Splunk
  - Serverless data transformations with Lambda, automated scaling, no data storage
- Use cases: stream and store data from devices and to create joins on streaming data
  - Spark/KCL do not read from Firehose, only from Kinesis DataStreams
  - Autoscaling and supports many data formats
  - Data conversions from JSON to Parquet/ORC only or S3 (only gzip is further loaded into Redshift)
- Destinations that it delivers streaming data to: S3, Redshift, ElasticSearch, Splunk
  - S3: Sends directly from Data Streams to S3 without any modifications
  - Redshift: data goes to S3 and then Redshift (trigger COPY command to load data)
  - ElasticSearch: Load data into an ElasticSearch cluster
  - Splunk instances: aggregate log files to have a central repository of log files
- As data comes into Firehose, it is buffered and aggregated together before it is shipped off
  - It buffers incoming streaming data for a certain period of time before delivering to a destination
  - Buffer size ranges from 1MB to 128MB for S3 or 1MB to 100MB for ElasticSearch
  - Data is shipped off whenever the buffer size is hot or the internal buffer is hot
    - If buffer internal is hit, then it ships the data to its final destination (buffer interval ranges from 60-90 secs)
    - Possible time delay due to buffer size and buffer internal before data is delivered may occur
  - To store data into final destination: Firehose reads records 1MB at a time, transform record with Lambda but Firehose will fill batch of data to write it into final destination (60 second latency minimum for non full batches)
  - Any source record that goes through Firehose can be configured to fail in another bucket of choice:
    - In case of transformation failure, archive that failure into an S3 bucket
    - No data lost with Firehose: either it ends up in source/target or failure records to another S3 bucket

- Firehose Buffer Sizing: Firehose receives a bunch of records and accumulates them into a buffer that is not flushed all the time (it is flushed based on time/size)
    - Define a buffer size (32MB): if there is enough data from source that fills buffer size, then it is not auto flushed
    - Set buffer time so after that, the buffer is not full and then it will be flushed (minimum is 1 minute)
    - Firehose can increase buffer size to increase throughput
        - High throughput: a lot of data going through Frirehose, then buffer size limit will be hit
        - Low throughput: buffer size limit is not hit but buffer time limit is hit
- Security: attach IAM roles so it can deliver to S3, Redshift, Splunk, ElasticSearch
    - Encrypt delivery stream with KMS: supported interface for VPC endpoints

MSK (Managed Streaming for Apache Kafka)
- Kafka is a Kinesis alternative that supports real time applications that transform, deliver, react to streaming data to build pipelines that can reliably get data between multiple systems and applications
    - Distributed streaming platform with pub/sub to stream records, stores streams of records in a durable way, process streams of records as they occur in real time
    - Topics have partitions (can only add partitions, not remove) that are 1MB by default (can configure higher)
- MSK is a fully managed Kafka service on AWS to CRUD Kafka clusters and CRUD/manage worker nodes
    - Auto recovery from common failures: consumer/producer apps can R/W operations with most minimal impact
    - Data is stored in EBS volumes, allows replacement for Kinesis with consumers/producers of data
    - Scaling is not seamless and retention time is 7 days or more (max retention is unlimited)
- Deploy MSK cluster in multi-AZ of VPC: Create custom configurations for your cluster
    - Default message size is 1MB: configure Kafka to be able to send/receive large messages into the cluster
- Data is replicated within brokers: topics are used for replication and consumers pull from Kafka topic
    - Choose number of AZs, the VPC/subnets, broker instance types and the number of brokers per AZ
- Security: encryption includes optional in flight using TLS between brokers and clients
    - At rest encryption for EBS volumes using KMS
    - Network security: attach specific security groups for Kafka clients
    - Authentication / authorization: define who can R/W to which topics (mutual TLS/ACLs, IAM access control)
- MSK Connect: Kafka Connect is a framework to take data from Kafka and put it somewhere else (or vice versa)
    - Kafka connect workers auth auto scaling capabilities: deploy any Kafka connectors to MSK connect as plug in
- MSK Serverless: Kafka on MSK without managing capacity / auto provisions resources, scalescompute/storage
    - Define topics and how many partitions per topic

SQS
- Queue that producers send message to and consumers pull messages from
    - Decouples applications: one application per queue because records are deleted after consumption
    - Messages are processed independently for standard queue: out of order issue (best effort ordering)
        - Messages have body of text and metadata attributes (key/value pairs) with delay delivery
            - Receive message ID and MD5 hash of the body
    - Delay messages with dynamic scaling of load
    - Horizontal scaling (number of consumers): duplicate messages with at least once delivery
    - Use cases: decouple apps, buffer write to DB, handle large loads at messages coming in
    - Pricigin: pay per API request and per network usage
- Fully managed, scales automatically from 1 message/second to 10,000 messages/second
    - Default retention of messages is 4 days, maximum is 14 days
    - No limit to how many messages can be in the queue
    - Very low latency (less than 10ms) on publish/receive messages
    - Limitation of 256 Kb per message sent
- Consumers poll SQS for messages (receive up to 10 at a time) and process message with visibility timeout
    - When done processing the message, they delete it using message ID and receipt handle
    - Consumers pull, process, delete messages from queue (a message cannot be processed by many queues)
- Limitations:
    - Maximum of 120,000 in flight messages being processed by consumers
    - Batch request has a maximum of 10 messages (maximum of 256 Kb size per message)
    - FIFO queue supports up to 3000 messages.second using batching
    - Data retention is from 1 minute to 14 days but once a message is read, then it is deleted from the queue
- FIFO queue: lower throughput (up to 3000 messages/seconds with batching, 300 messages/sec without batching)
    - Messages are processed in order by consumers and are sent exactly once
    - Deduplication interval of 5 minutes using duplication ID
- SQS Extended Client: use to send large messages and put data in S3

    - Send message metadata to SQS so extended client in consumer receives metadata, where the file is in S3 and consumer reads file directly from S3
- Security: VPC endpoint is provided through an interface
    - Encryption in flight using HTTPS endpoint
    - Enable SSE using KMS: SSE encrypts the body, not the metadata
    - Set the SQS queue access policy for finer control over IP over the time the requests come in
    - IAM policy to allow usage of SQS and SQS Queue Access Policy

IoT
- An IoT thing can be any connected device to AWS that are configured to be interacted with
    - The Thing is registered with AWS for managed communication: it communicates using device gateway
- IoT Device Gateway is the entry point for IoT devices connecting to AWS to allow devices to securely communicate with AWS: supports many IoT protocols like MQTT, HTTP (FTP is not supported)
    - Fully managed, scales automatically to support over 1 billion devices
- IoT Message Broker is a pub/sub in IoT Device Gateway to handle which destinations receive which messages:
    - Publish messages to the broker with very low latency: devices use broker to communicate with one another
    - Messages are sent using IoT protocols and are published into topics
    - Broker forwards messages to all clients connected to the topic
- Thing registry is the IAM of IoT, where all connected IoT devices are represented in AWS IoT registry, organizes the resources associated with each device in AWS:
    - Each device gets a unique ID and supports metadata for each device
    - Creates x.509 certificate to help IoT device connect to AWS
    - Use IoT groups to group devices and apply permissions to groups
- Authentication: IoT device policies are attached to x.509 certificate and able to revoke any device at any time
    - IoT policies are JSON documents that can be attached to groups (instead of individual Things)
    - IAM policies are attached to users, groups, roles and are used for controlling IoT AWS APIs
- Device Shadow is a JSON document representing the state of a connect thing: set to different desired states
    - IoT Thing will retrieve state when online and adapt it: use to change the state of the thing in the real world
- Rules Engine: rules are defined on MQTT topics and they say when it is triggered / what action it should do
    - Rules need IAM roles to perform their actions
- IoT Greengrass: brings compute layer to device directly
    - Able to run Lambdas on the devices to reprocess and execute predictions based on ML models or keep the device in sync to communicate between local devices
    - Operate offline to deploy functions from AWS directly to the devices to update devices at anytime
- Security: IoT policies attach certificates/identities into devices, users, groups, roles (revoke any device at any time)
    - Attach IAM role to IoT rules engine to ensure it is capable of performing their actions

DMS
- Quick, secure way to migrate databases to AWS
    - Supports homogeneous and heterogeneous migrations, and continuous data replication using CDC
    - Must create EC2 instance (supports t2, t3, c4, r4, r5) to perform replication tasks
    - Sources: on premise DBs, EC2 instances, Azure, RDS, S3 (remains available during migration)
    - Targets: on premise DBs, EC2 instances, Redshift, S3, DynamoDB, ElasticSearch, Kinesis, DocumentDB
    - Free: All data transferred into DMS and transferred between DMS and RDS/EC2 instances DBs in same AZ
    - Use for cross region replication of DBs running in other regions
- AWS SCT converts your DB scheme from one engine to another: not needed is migrating the same DB engine

Online Data Transfer
- Simple, easy way to transfer data in/out of AWS via online methods: sends via the Internet
- AWS Data Sync: automate moving data between on premise into AWS (S3, EFS, FsX)
    - Transfer data up to 10 times faster than other open source tools
    - Use cases: one time data migration, recurring data processing workflows, automate data replication
    - Good to set up automation for various data locations into AWS
- FTP / SFTP / FTPS: fully managed support for file transfer directly in/out of S3
    - Great if already have authentication set up
- S3 Transfer Acceleration: faster way to use public data to transfer data into AWS
    - Maximize available bandwidth regardless of distance from customers to S3

Direct Connect (DX)

- Provides dedicated, private connection from remote or on premise network to VPC:
  - Connection must be setup between DX and AWS DX locations
  - Set up a virtual private gateway on VPC so on the same connection, can access public and private resources
  - Set up new connection from on premise to DX location as a hard line to AWS (Takes >1 month to establish)
  - Once established, connect to public zone (S3, DynamoDB) or VPC services (EC2, RDS, Redshift)
- Use cases: increased bandwidth throughput (good for working with large data sets at a lower cost)
  - More consistent network experience for apps using real time data feeds
  - Supports hybrid environment because have connectivity between on premise and AWS
  - Low latency access for data that needs to be rapidly transformed to AWS
- Cost: data transferred into AWS is free but pay for data transferred out of AWS
  - Pricing is dependent on source region and DX location (from $0.02 to $0.19 per GB per data transfer out)
- Use DX Gateway if you want to set up a DX to one or more VPC in many different regions (in same account)
- Connection types:
  - Dedicated Connections for 1, 10, 1000 Gbps capacity with a physical ethernet port dedicated to customer
    - Request to AWS then it is completed by AWS DX partner
  - Hosted Connections: connection requests are made via AWS DX partner and capacity can be added/removed on demand (use to vary network speeds)
    - Use 50, 100, 300, 400, 500 Mbps and go into internal of 1, 2, 5, 10 Gbps
- No encryption but it is private (data in transit is not encrypted)
  - AWS DX and VPN provides IPSEC encrypted private connection (extra level of security but complex set up)
- Resiliency: high resiliency (one connection at multiple locations) for critical workloads
  - Set up multiple DX with different DX locations for redundancy
  - Max resiliency for critical workloads: multiple DX locations, each with multiple connections for max latency
    - Maximum resilience via separate connections terminating on separate devices in more than one location

Snow Family

- Highly secure, portable device to collect, process data at edge and migrate in/out of AWS
  - Data migration: Snowcone, Snowball Edge, Snowmobile
  - Edge computing: Snowcone, Snowball Edge
- Snowball Edge: a box to move physical data (TBs or PBs) of data in/out of AWS
  - Alternative to moving data over network or paying network fees (Pay per data transfer job)
  - Provide block storage and S3 compatible object storage
  - Snowball Edge Storage Optimized: 80TB of HDD capacity for block volume and S3 object storage
  - Snowball Edge Compute Optimized: 42TB of HDD capacity for block volume and S3 object storage
  - Use case: ;arge data cloud migrations, DX decommission, disaster recovery
- Snowcone: small, portable computing anywhere device that is secure enough to withstand harsh environments
  - 2 CPUs, 4GB memory with word or wireless access (USB-C power using a cord or optimal battery)
  - Right device for edge computing, storage, data transfer
  - Must provide own battery and cables
  - Can be sent back to AWS offline or connect it to internet and use DataSync to send data
- Snowmobile: truck for exabytes of data, where each truck has 100PB of capacity (use multiple in parallel)
  - High security, temperature controlled, GPS, 24/7 surveillance
  - Better than Snowball if you transfer more than 10PB
- Edge Computing: process data as it is being created at an edge location (anywhere without Internet that produces data, thus has access to compute data)
  - 3 locations need Snowball Edge or Snowcone for edge computing
  - Use edge computing to preprocess data, ML at edge or transcoding media streams
- AWS OpsHub is a software on a local machine to manage Snow family devices

DOMAIN 2: STORAGE

S3

- Storage of objects (files) in buckets (directories): buckets are defined at region level (need globally unique names)
  - Objects have a key which represents full object path: object values are content of body (maximum size is 5TB)
  - If uploading more than 5GB, must use multipart upload
  - High durability of objects across multiple AZ
  - Access logging with S3 Access Logging and CloudTrail: alert with CW alarms, audit with access analyzer
- All operations are strongly consistent:
  - After a successful write of a new object (new PUT)
  - After an overwrite or delete of an existing object (overwrite PUT or DELETE)
  - Any subsequent read request immediately receives latest version of the object (read after write consistency)
  - Any subsequent list request immediately reflects changes (LIST consistency)
- Storage classes:
  - Standard General Purpose: used for frequently accessed data for low latency and high throughput
    - Sustain 2 concurrent facility failures
  - Standard Infrequent Access: data that is less frequently accessed but request rapid access when needed (lower cost than standard general purpose)
    - For weekly to monthly access: $0.0125 per GB stored
    - Additional costs:
      - Write charges include $0.01 per 1000 requests
      - Retrieval charges include $0.01 per GB
      - Objects smaller than 128Kb are billed as 128Kb objects (best to aggregate data before)
  - One Zone Infrequent Access: high durability only within a single AZ ($0.01 per GB stored)
  - Glacier: data archives acceptable for retrievals from minutes to hours ($0.004 per GB retrieval)
    - Each capacity unit (compute unit) is used to process archive data back into one of the non archive classes (each capacity unit costs $100 per GB of data processed)
    - Glacier Instant Retrieval: low cost object storage for archiving / backup
      - Millisecond retrieval, great for data accessed once a quarter with minimum storage duration of 90 days
    - Glacier Flexible Retrieval: expedited (1 to 5 mins), standard (3 to 5 hours), bulk (5 to 12 hours) with minimum storage duration of 90 days
    - Glacier Deep Archive: standard (12 hours) and bulk (48 hours) with minimum storage of 180 days
  - Intelligent Tiering: small monthly monitoring and auto tiering fee to move object automatically between access tiers based on usage (no retrieval charges but pay to monitor objects)
    - If S3 Standard object is not accessed for 30 days, then it is moved to the configured IA tier
    - If an object in S3 IA is request ,then it is moved to S3 Standard
    - Monitoring is $0.0025 per 1000 objects and storage is $0.0125 to $0.023 per GB
- API Call: to gain access, must be granted IAM policy
  - Each step has an implicit deny but an explicit allow anywhere will overwrite implicit deny
- Lifecycle Rule: transition objects between storage classes (can be created for a specific prefix)
  - For infrequently accessed objects, move them to Standard IA
  - For archive objects not needed in real time, use Glacier or Deep Archive
  - Moving objects can be automated using a lifecycle configuration
  - Define transition actions for when objects are transitioned to another storage class
    - Move objects to standard IA class 60 days after creation
    - Move to glacier for archiving after 6 months
  - Expiration Actions: configure objects to expire (delete) after some time
    - Access log files can be set to delete after 365 days
    - Use to delete old version of files or incomplete multipart upload
- Versioning: must enable versions on files at bucket level (same key overwrite will increment the versions)
  - Use versioning to store specific numbers of objects in the bucket
  - Best practice is to version your buckets to protect against unintended deletes
  - Any file that is not versioned prior to enabling versioning will have null version
  - Suspending versioning does not delete previous one
  - Use object locking to prevent deletion of objects
  - Use MFA to require a token to delete objects
  - Must enable bucket versioning for cross region replication
- Replication: must enable versioning in source and destination
  - Cross region replication for compliance, lower latency access, replication across accounts
  - Same region replication for log aggregation, live replication between test and production accounts
  - Copying is asynchronous: must give proper IAM permissions to S3 bucket policy

    - After activating, only new objects are replicated but can replicate existing objects using S3 Batch Replication
    - For delete operations: replicate delete markers from source to target and deletions with a version ID are not replicated to avoid malicious deletes
    - No chaining of replication: if bucket 1 has replication into bucket 2 (which has replication in bucket 3), then the objects in bucket 2 are not replicated to bucket 3
- Performance: S3 automatically scales to high request rates with latency of 100 - 200ms
    - Achieve at least 3500 PUT/COPY/POST/DELETE and 550 GET/HEAD requests/second/prefix in a bucket
    - No limits to the number of prefixes in a bucket
    - Spread reads across all 4 prefixes evenly, can achieve 22,000 requests per second for GET/HEAD
    - Transfer acceleration for upload/download: increase transfer speed by transferring file into an AWS edge location which will forward data to S3 bucket in target region (compatible with multi part upload)
    - KMS Limitations: If you have SSE-KMS encryption on objects, then may be impacted on KMS limits
        - Upload calls the GenerateDataKey KMS API
        - Download called the Decrypt KMS API
        - Count towards the KMS quote per second but can request a quota increase
    - Optimize performance with multipart upload: recommended for files greater than 100MB and must be used for files greater than 5GB (parallelize uploads to speed up transfers)
    - S3 Byte Range Fetches: parallelize GETs by requesting specific byte range
        - Better resilience in case of failures: used to speed up downloads or retrieve only partial data
- Multipart Upload: utilize for files larger than 100MB
    - A single S3 PUT can be up to 5GB of data but S3 objects can be up to 5TB
    - Multipart objects acts as a meta objects that stores all the info about the upload while it is happening
    - Can only have 10,000 parts in a multi part upload
    - All parts (except th final) must be at least 5MB (all objects must be between 5 to 100MB)
    - Specifying the same part number as a previously uploaded part can be utilized to overwrite that part
    - Create lifecycle policy for bucket t automatically abort multipart uploads after a specified time period
- Transfer Acceleration: works well if within one region (becomes content ingestion and distribution network)
    - Users in far regions are going to see latency, which impacts application performance
    - Implement Cloudfront to get data to users quickly, with a network of caching nodes that each have an optimized network path between them to increase application performance
    - To upload data, transfer acceleration must be enabled per bucket and must use special endpoints:
        - bucketname.s3-accelerate.amazonaws.com
        - bucketname.s3-accelerate.dualstack.amazonaws.com
    - Additional $0.04 per GB for data transferred from US, Europe, Japan and $0.08 for all other locations
- Encryption: There are 4 methods of encrypting objects in S3:
    - SSE-S3: encrypt objects using keys handled and managed by AWS
        - Objects are encrypted server side with AES256 encryption types
        - Must set header to: x-amz-server-side-encryption = AES-256

    - SSE-KMS: leverages KMS to manage encryption keys
        - Objects encrypted server side: KMS gives user control and audit trail
        - Must set header to: x-amz-server-side-encryption = aws:kms
    - SSE-C: manage encryption keys outside of AWS
        - S3 does not store the encryption key you provide, must use HTTPS to transfer data in S3 so encryption key must be provided in HTTP headers for every HTTP request made
    - Client Side Encryption: client encrypts object before uploading to S3 and decrypt data when retrieving from S3
        - Use S3 encryption client library: customer fully managed keys and encryption cycle
        - Customer fully manages the keys and encryption cycle
    - Encryption In Transit (SSL/TLS): S3 exposes an HTTP endpoint that is not encrypted and an HTTPS endpoint for encryption in flight (free to use endpoint but HTTPS is recommended)
        - Most clients use HTTPS endpoints by default and it is mandatory for SSE-C
- Security:
    - User Based (IAM Policies): which API calls are allowed for a specific user from IAM console
    - Resource Based:
        - Bucket Policies are bucket wide rules from S3 which allows cross account
            - JSON based, apply buckets and objects by allowing a set of API actions to allow or deny
            - Use to: grant access to bucket, force objects to be encrypted at upload, grant access to other account
        - Object ACL for finer grain access, Bucket ACL is less common
    - IAM principle can access an S3 object: user IAM permissions allow it OR the resource policy allows it and need to make sure there is no explicit deny
    - Logging and audit: S3 access logs can be stored in other S3 buckets (API calls logged in Cloudtrail)

    - Bucket settings for blocking public access to block objects from being public if the account has some restrictions (new ACLs, any ACLs or new public bucket or access point policies)
        - Block public / cross account access to buckets/objects
        - If you know your bucket should never be public (to prevent data leaks), then leave these on
    - User Security: MFA delete can be required in versioned buckets to delete objects
        - Pre Signed URLs are valid only for a limited time
- S3 and Glacier Select: retrieve less data using SQL by performing server side filtering
    - Filter by rows and columns using SQL statements: less network transfer, less CPU cost client side
    - Glacier Select can only do uncompressed CSV files: SQL statements on data in Glacier without having to restore the data within minutes
    - S3 Select is an API call that allows making SELECT statements against data in S3 bucket
        - Available in AWS CLI/SDK: read data from CSV, JSON, Parquet
        - Supports columnar compression via GZIP
        - Data must be encoded in UTF8 format and encryption is supported on server side
- Event Notifications: events are anything that happens to objects (delivered within seconds to destinations)
    - Create as many S3 events as desired, where object name filtering is possible for only certain filters
    - Destinations: Lambda, SQS, SNS, EventBridge
    - Eventbridge for advanced filtering objects to multiple destinations with archive, replay events, reliable delivery
- CLI commands:
    - aws s3 cp cli.txt s3://bucketname -> copy local file to S3
    - aws s3 mv cli.txt s3://bucketname -> move file to S3
    - aws s3 ls bucketname -> list contents of a bucket
    - aws s3 sync s3://bucketname -> only move files that are not already in the bucket

Database Engine Types
- Relational: data normalization, uses IDs to hold relations between tables
    - Row based: Aurora, MySQL, Oracle, MariaDB, SQL server
        - Good for OLTP (online transaction processing): rapid transactions, translation roll back, low latency apps
    - Columnar based: Oracle, Redshift
        - Good for OLAP (online analytical processing): analytic workload, managing large amounts of data, handling complex and long running query operations
    - Key Value: ElasticSearch (Redis/Memcached)
        - Very fact because all data i stored in memory bit only used as a caching point
        - If elasticsearch instance goes down or it restarted, then could lose all data stored there
        - Redis supports disk persistence so can use it as a primary data store
        - DynamoDB can be used in key value or document style but table structure can’t be changed after creation
        - Documents: structured or semi structured documents, where the metadata is stored in the records
            - Can be susceptible to critical scale failure
        - Graph: Neptune is between relation and hyperrelational (could reveal otherwise hidden patterns in data)
            - Data is structured as nodes connected with edges, where edges are stored attributes between nodes

DynamoDB
- Traditional apps leverage RDBMS databases: places strong requirements on how data should be modeled
    - Vertical scaling allows more powerful CPU, RAM, IO (SQL databases)
    - Horizontal scaling allows increasing reading capability by adding EC2/read replicas (noSQL DBs(
- noSQL DBs are non relational and distributed DBs like mongoDB, documentDB
    - Does not support query joins so all data needed for query needs to be present in one row
    - Does not perform aggregation like sum or average
- DynamoDB is fully managed, highly available with replication across multiple AZs (integrates with IAM)
    - It is a key value and document store with standard and infrequent access classes
    - Scales to massive workflows and distributed databases
    - Handles millions of requests per second, trillions of rows, 100s of TBs of storage
    - Enables event driven programming with dynamoDB streams
    - If storing documents and want to access the values within the documents, store it as a flat JSON
- Made up of tables, where each table has a primary key (must be decided at creation time):
    - Each table can have an infinite number of items (rows) and each item has attributes (can be added over time or can be null) but max item/row size is 400Kb
    - Supported data types are scalar, document, set types
    - Ideal for hot data that needs to be ingested at scale in a DB
    - Not good for prewritten apps tied to a traditional RDB, for joins or complex transactions, for binary large object data, for large data with low I/O rate

- Primary keys:
  - Partition Key (hash strategy): must be unique for each item and diverse enough so data is distributed
  - Partition Key + Sort Key (hash + range): combination must be unique for each item and data is grouped by partition key (max number of fields that can make a primary key is 2)
- Read/Write Capacity Modes control table capacity management (switch modes every 2-4 hours)
  - Provisioned Mode (default): specify the number of R/Ws per second
    - Plan capacity beforehand: pay for provisioned R/W capacity units
  - On Demand Mode: R/Ws automatically scale up or down with your workloads
    - No capacity planning needed, pay for what you use (more expensive)
- Provisioned R/W Capacity Mode: table must have provisioned R/W capacity units
  - RCU (Read Capacity Unit) is throughput for reads, there are two kinds of reads:
    - Eventually Consistent Read: read right after a write, can get some stale data because of replication
    - Strongly Consistent Read: a read right after a write will immediately get correct data
      - Set ConsistentRead parameter to True in API calls
      - Consumes twice the RCU (higher latency, more expensive query)
    - 1 RCU represents 1 strongly consistent read per second or 2 eventually consistent read per second for an item up to 4Kb in size (if items are larger than 4Kb, then more RCUs are consumed)
      - 10 SCRs per second with item size 4Kb = 10 x (4Kb/4Kb) = 10 RCUs
      - 10 SCRs per second with item size 6Kb = 10 x (8Kb/4Kb) = 20 RCUs (round 6 to 8)
      - 16 ECRs per second with item size 12Kb = (16/2) x (12Kb/4Kb) = 24 RCUs
      - SCRs of 10Kb at rate of 10/second = 10 x (12Kb/4Kb) = 30 RCUs
      - 12 ECRs per second with item size 16KB = 12 x (16Kb/4Kb) / 2 because can only do 2 eventually consistent reads per second for an item size of 4Kb with 1 RCU = 24 RCUs
  - WCU (Write Capacity Unit) is throughput for writes:
    - 1 WCU represents 1 write per second for an item up to 1Kb (if item > 1Kb, then more WCUs consumed)
      - Write 10 items/sec with item size 2Kb = 10 x (2Kb/1Kb) = 20 WCUs
      - Write 6 items/sec with item size 4.5Kb = 6 * (5Kb/1Kb) = 30 WCUs (always rounded to upper Kb)
      - Write 120 items/minute with item size 2Kb = (120/60) x (2Kb/1Kb) = 4 WCUs
      - Write an item of 8Kb in size at rate of 12/second = 12 x (8Kb/2Kb) = 96 WCUs
    - Option to setup auto scaling of throughput to meet demand
    - Throughout can be temporarily exceeded: if burst capacity consumed, then get a ProvisionedThroughputExceeded exception (need to implement an exponential backup retry mechanism)
      - Throttling: if exceed RCUs or WCUs, then get a ProvisionedThroughputExceeded
      - Reasons: hot keys or partitions (one partition key is being read too often / popular item) or large items
      - Solutions: exponential backoff when exception is encountered (already in SDK), distributed partition keys as much as possible or if RCU is an issue, then can use dynamoDB accelerator (DAX)
- On Demand R/W Capacity Mode: auto scale up/down with workloads (no capacity planning needed)
  - Unlimited WCU/RCU, no throttle, more expensive
  - Charged for R/Ws that you use in terms of RRU (throughput for reads)/ WRU (throughput for write)
- Partitions Internal: each table has partitions (copy of data that lives on specific servers)
  - Data is stored in partitions: partition keys go through a hashing algorithm to know which partitions to map to
  - WCUs/RCUs are spread evenly across partitions
- Indexes:
  - Local Secondary Index: Alternative sort key for your table (same partition key as that of base table)
    - It is a way of adding a secondary key to the table: the sort key is consistent with one scalar attribute
    - Up to 5 local secondary indexes per table (Must be defined at table creation time)
    - Attribute projections: contain some or all the attributes of the base table (keys only, include all)
    - Uses the WCUs/RCUs of the main table so no special throttling consideration
  - Global Secondary Index: alternative primary key (hash or hash + range) from the base table
    - A complete second key structure to query against in the same way as it would the primary index
    - Speeds up queries on non key attributes: Index key consists of scalar attributes
    - Attribute projections: some or all of the attributes of the base table (keys only, include, all)
    - Must provision RCUs and WCUs for the index: can be added or modified after table creation
    - If the writes are throttled on GSI, then main table with throttle
    - Even if WCU on main tables are fine, if throttling on GSI then main table will throttle
- DAX: fully managed, highly available, seamless in memory cache for individual objects, queries, scans
  - Microsecond latency for cached reads and queries (doesn't request application logic modification because it is compatible with existing DynamoDBs APIs)
  - Solves hotkey problem (too many reads)
  - Secure: encryption at rest with KMS, VPC, IAM, CloudTrail
  - DAX Cluster is made up of cache nodes that need to be provisioned in advance (up to 10 nodes in cluster)

        - Multi AZ setup so at least 3 nodes are recommended for production
    - 5 minute TTL for cache by default so every cached data will live for 5 minutes in your DX cluster
    - Elasticache stores aggregation results in a cache to avoid performing the same queries
- Streams: Ordered stream of item level modifications (CRUD) in a table
    - Stream of data comes into the table and logs all the changes: use this to trigger Lambda to perform secondary functions based off of changes in the table
    - Stream records can be: sent to kinesis data streams, read by lambda or KCL applications
    - Data retention for up to 24 hours
    - Use cases: react to change in real time analytics, insert to tables or ElasticCache, cross region replication
    - Ability to choose information written to the stream:
        - Keys only: only key attributes of the modified item
        - New image: the entire item, as it appeared after it was modified
        - Old image: the ensure image, as it appeared before it was modified
        - New and old images: both new and old images of the item
    - Streams are made up of shards but no need to provision shards (automated by AWS)
    - Records are not retroactively populated in nstreams, it must be enable to receive updates on changes
    - Integration with Lambda: define event source to read from stream and lambda is involved synchronously
    - To react in real time to users activating accounts and send an email, integrate lambda with dynamodb streams
- TTL: automatically delete items after an expired timestamp
    - Doesn't consume any WCUs (no extra cost)
    - TTL attribute must be a numerical data type with epoch timestamp value
    - Expired items that haven't been deleted will still appear in reads, queries, scans so need to filter them out
    - Expired items are deleted from both LSIs and GSIs
    - A delete operation for each expired item enters the dynamoDB streams (can help recover expired items)
    - Use cases: reduce stored data by keeping only current items, adhere to regulatory obligations
- Security: access fully controlled by IAM
    - Use VPC endpoints to access dynamo without internet
    - Encryption at rest using KMS and encryption in transit with SSL/TLS
    - Backup and restore feature available: point in time restoration like RDS, no performance impact
    - Global tables: multi region, full replication, high performance
    - Tables encrypted at rest: KMS encryption for base tables and secondary indices
        - AWS owned key, managed key, customer managed key
        - Streams are encrypted
- PartiQL: use SQL like syntax to manipulate DynamoDB table
    - Supports insert, update, delete, select, batch operations
- Basic APIs:
    - PutItem: creates or fully replaces an old item with same partition key, consues WCUs
    - UpdateItem: edits an existing item attributes or adds a new item if it doesn't exist
        - Used to implement atomic counters: numeric attribute that is unconditionally incremented
    - ConditionalWrites: accepts a write/update/delete only if conditions are met, otherwise an error
        - Helps with concurrent access to items and no performance impact
    - GetItem: reads based on primary key (can be hash or hash + range)
        - Eventually consistent read by default or strongly consistent reads (more WRU and latency)
        - Specify ProjectionExpression to retrieve only certain attributes
    - Query: returns items based on KeyConditionExpression (partition key value required) and FilterExpression to allow additional filtering for non key attributes before the data is returned to you
        - Returns the number of items specified in limit or up to 1MB of data, but can paginate results
        - Query tables, local secondary index or global secondary index
    - Scans: scans the entire table and filters out data, returns up to 1MB of data (inefficient: consumes alot of RCU)
        - Faster performance, use parallel scan (multiple workers scan multiple data segments at the same time, increases throughput and RCU consumed but limit impact of parallel scans using limit conditions)
        - Use ProjectionExpression and FilterExpression with no changes to RCU
    - DeleteItem: deletes individual items (can perform conditional delete)
    - DeleteTable: deletes a whole table and all its items (quicker deletion than calling DeleteItems on all items)
    - BatchOperations: save inward latency by reducing the number of API calls (operations in parallel for efficiency)
        - Part if the batch can fail so need to try again for failed items
    - BatchGetItem: returns items from one or more tables (up t0 100 items, up to 10MB data)
        - Items retrieved in parallel to minimize latency
    - BatchWriteItem: up to 25 PutItem and/or DeleteItem in one call
        - Up to 16 Mb of data written or 400 Kb of data per item (can't update items so use UpdateItems for that)

RDS

- Hosted relational DB (Aurora, MySQL, PSQL, MariaDB, Oracle, SQL Server) for small data
  - Managed service that doesn't give access to the OS of the instance it is run on
  - Provides management API to manage DB instance
  - Connects to EC2 instances using Route53 to create/manage DNS endpoints
  - RDS offers full ACID compliance
- Built from other AWS services: manages DB engines on EC2 but it is more expensive
  - EC2 instances connects with EBS and DB engine
  - Manage OS instance with parameter group controlled through RDS API and are used to manage plugins
- Aurora is the only engine that provides a cluster endpoint so it connects to nodes
- Disaster recovery through multi AZ deployments: Entire process takes 60 minutes
  - Primary and secondary instance in different regions that replicate at the block level (DB engine not involved)
  - DB engine does not run at all on secondary instance
  - RDS monitors the engine, compute instances, EBS volumes, block level replication
  - If engine or EC2 or EBS fails in primary instance, then that instance is terminated and the secondary is upgraded to be the primary instance
  - A new secondary in region of failed primary is created with engine not running (set up block level replication)
  - Possible to run RDS in a single AZ but not recommended for production environments
- Security: VPC provides network isolation and security groups control network access to DB instance
  - KMS provides encryption at rest, SSE provides encryption in flight
  - IAM policies provide protection for RDS API and IAM authentication is supported by MySQL/PSQL
  - Must manage user permissions with the DB itself

Aurora

- MySQL and PSQL compatible relational DBs with 1/10th of the cost of commercial databases
- Run MySQL/PSQL compatible versions with a proxy and a storage cluster:
  - As the storage is used, it will add 10GB chunks
  - No hard IOPS limit because data is stored on chunks, which is individual nodes within the storage cluster
  - Storage engine provides unlimited pool of IOPS that you pay for
  - Compute nodes can go into a zero allocation state so it should only be used for infrequently accessed data
  - Aurora capacity unit allows the instance to sale up or go down to meet DB demand:
    - These scale events can fill if there are long running transactions happening
  - Use case: analytic systems (need to access data through SQL that is already in format)
- Up to 64TB per database instance and up to 15 read replicas
- Supports continuous backup to S3, replication across AZs and automatic scaling with Aurora serverless
- Offers VPC network isolation, at rest with KMS and in transit with SSL
- Security: VPC offers network isolation and security groups control network access to DB instance
  - KMS for encryption at rest, SSL for encryption in flight
  - IAM authentication is supported by PSQL and MySQL
  - Manage user permissions with database itself

Amazon OpenSearch (ElasticSearch)

- A search domain that runs more of the ELK stack: Elasticsearch, LogStash, Kibana
  - LogStash ingots, processes, stores log data
  - Kibana acts as a utility interface GUI for Elasticsearch
  - Any service that can interact with APIs can send data into Elasticsearch
  - Integrates with Kinesis Firehose as the target, with CW logs and IoT
- Fork between Elasticsearch and Kibana: it is a search engine, analysis tools, visualization tool for data stores
  - Primarily used for search and analytics: import log data using Kinesis and use dashboard for data stored
  - Used for: full text search, log analytics, app monitoring, security analytics, clickstream analytics
  - Enable snapshots of the domain so don't lose cluster data
  - Zone awareness for higher availability
  - Not used for: OLTP, ad hoc data querying
  - Kibana does not allow email/password combinations as a supported form of login
- Organizes data as indices with the purpose to enable word searching (search engine utility) using reverse index:
  - Creates an index that defines characters and puts all words from document into that inde (part of metadata is which document the word appears in and exactly where they are within that document)
  - A mapping of all the words in that document and returns document quickly to return words or phases
- There are 3 entities:
  - A document is what you are searching for: could be more than text (any structured JSON data works)
    - Every document has a unique ID and type: split into shards (every shard has its own self contained index)

        - Every document is hashed of a particular shard and every shard may be on a different node in a cluster
    - A type defines the schema and mapping shared by documents that represent the same sort of things
    - An index powersearch into all documents within a collection of types:
        - Contain inverted indices that let you search across everything
- Redundancy: an index has two primary shards and two replicas
    - App goes round robin requests amongst nodes so write request is routed to primary shard and then replicated to however many replicas specified
    - Read requests are routed to the primary or any replica shard
- OpenSearch is fully managed (not serverless) so scale up or down without downtime (not automatic)
    - Pay for what you use (instance hours, storage, data transfer, network isolation)
    - AWS integration with S3, Kinesis Data Streams, DynamoDB Streams, CloudWatch, CloudTrail
    - Kinesis, DynamoDB, LogStash/beats, ElasticSearch native APIs offer ways to import data to OpenSearch
- Dedicated master nodes: need to choose instance types and counts
    - Only used for management of OpenSearch domain, doesn't hold/ process any data so don't need a lot of them
    - A domain is a collection of all the resources needed to run an OpenSearch custer
- Security: resource based, identify based, IP based policies, request signing, put cluster in VPC, Cognito
    - Configured in VPC: want customers outside VPC to visualize logs reaching ElasticSearch using Kibana
        - Achieve by using a reverse proxy or VPN or VPC direct connect
    - VPC provides network isolation: encryption data at rest using KMS or in transit using SSL
    - IAM or Cognito (allows end users to log into Kibana through enterprise identity provided based authentication)
- Storage Types:
    - Standard Data Nodes: use hot storage, instance store or EBS volumes for faster performance
    - Ultra Warm Storage: uses S3 and caching, best for indices with few writes (like log or immutable data) but has slower performance (at a much lower cost) but must have a dedicated master node
    - Cold Storage: uses S3 (een cheaper) for periodic research on older data
        - Must have dedicated master and UltraWarm enabled
        - Not compatible with t3 or t3 instance types on data nodes
        - Using fine grained access control: must map users to cold_managers role in OpenSearch dashboards
    - Data muay be migrated between storage types
- Index State Management: automates index management policies
    - Examples: delete old indices, move indices to read only state, move indices from hot to warm to cold
    - ISM policies are run every 30-40 minutes and notifications are sent when completed
    - Index roll ups: periodically roll up data into summarized indices to save on storage costs
    - Index transforms: create a different view to analyze data differently for grouping and aggregations
- Cross Cluster Replication: replicate indices, mappings, metadata across domains
    - Ensures high availability in an outage: replicate data geographically for better latency
    - Follower index pulls data from leader index: requires fine grained access control and node to node encryption
    - Remote index allows copying indices from one cluster to another on demand
- Stability: best practice is 3 dedicated master nodes to avoid 'split brain'
    - Don't run out of disk space: ruin storage requirement is = source data x (1 + number of replicates) x 1.45
    - Choosing number of shards is important
    - Choosing instance types: should be at least 3 nodes (important for storage requirements)
- Performance:
    - Memory pressure can be from unbalanced shard allocations across nodes or too many shards per cluster
    - Fewer shards can have better performance if have memory pressure errors (delete old or unused indices)

ElasticCache

- Caches are in memory databases with very high performance and low latency
    - Reduces load off of databases for read intensive workloads, make apps stateless, write scaling using sharding, read scaling using read replicas, multi AZ with failover capability
- AWS handles OS maintenance, patching, optimizations,setup, configuration, monitoring, failure recovery, backups
- Redis: in memory key value store with super low latency, support for read replicas
    - Cache services reboot by default (persistent), great to host user sessions / leaderboards / etc
    - Multi AZ with automatic failover for disaster recovery if you don't want to lose cache data
- MemCached: in memory object store but cache does not survive reboot
    - Use cases: quick retrieval of objects from memory, cache often accessed objects

Neptune

- GraphDB services: graphs have nodes that contains data and are connected via edges to traverse
    - Supports multi AZ deployment, instance monitoring, snapshots / backups
    - Use cases: security (pattern recognition / fraud detection), social media (identity connections), analytic science

DocumentDB

- Document store engine: formats data as JSON
- Use cases: social media profiles, object catalogs,content management systems, semi status documents
- MongoDB provides a compatible interface, fully managed service with storage and auto scaling feature
    - Automatically scales upward and uses index JSON data structures

DOMAIN 3: PROCESSING

Lambda

- Serverless data processing to run code in AWS as it is moved around between services
    - AWS handles server management (scaling, failures): don't pay for unused processing time (generous free tier)
    - Easy to split up development, code is automatically cached
    - Uses: real time file/stream processing, ETL, cron replacement, process events
    - Supported languages: node.js, python, java, C#, go, powershell, ruby
    - High availability: no scheduled downtime, retries failed code 3 times
    - Unlimited scalability with safety throttle of 1000 concurrent executions per region
    - High performances: new function are callable in seconds, events are processed in ms
    - Limits: timeout is 15 mins, invocation payload (request/response) is 6MB synchronous or 256Kb asynchronous
        - Invocation frequency per region (amount of request per second) is 10 concurrent executions
- Anti Patterns: long running apps, dynamic websites, stateful apps (use dynamoDB or S3 to keep track of state)
- Integration with Redshift: best practice for loading data into Redshift is with the COPY command
    - Use lambda to respond to new data that shows up at any time (batch new data and load them with COPY)
    - Use dynamoDB to track what's been loaded
- Integration with Kinesis:
    - Lambda receives event with batch of stream records: specify batch size when setting up trigger (up to 10,000 records) but too large of a batch size can cause timeouts (split batches beyond lambda 6MB payload limit)
    - Lambda retries the batch until it succeeds or the data expires (use more shards to ensure processing isn't held up by errors) and processed data synchronously

Glue

- System for building table definitions and defining ETL:
    - Serverless ETL discovery and definition of tables and schemas: S3, RDS. Redshift, most SQL databases
    - Custom ETL, fully managed jobs: trigger driven, on a schedule or on demand
    - Use to query data in S3, join data for a data warehouse, create a centralized data catalog
    - Requires a data source where data is loaded from(S3, DynamoDB, RDS, Redshift, DB in EC2, external DBs)
    - Serverless streaming ELT by consuming Kinesis or Kafka (clean/transform data in flight, store results in S3)
- Anti Patterns: multiple ELT engines
- Cost: billed by the second for crawler and ETL jobs
    - First million objects stored and accessed for free for Glue Data Catalog
    - Development endpoint for developing ETL code charged by the minutes
- Glue Crawler: scans data in S3, create schema and populates the Glue Data Catalog
    - Crawls DBs to find out information about data or metadata (column names, types, etc)
    - Stores only table definition, original data stays in S3 (Extracts partition based on how S3 is organized)
    - Once cataloged, treat unstructured data like it's structured
- Glue Data Catalog can serve as a Hive metastore or can import Hive metastore into Glue
    - It is a persistent metadata store that stores, annotates, shares metadata between AWS services
    - A centralized repository where this is only 1 data catalog per AWS region, providing a uniform repository so different systems can store and find metadata to query / transform that data
    - Provides comprehensive audit to track schema changes and data access control to ensure data is safe
    - Hive is a service to run on MapReduce to run SQL like statements on EMR
    - Data catalog is made up of tables that you query to run ETL joins on
- Glue ETL provides automated code generation in Python or Scala with encryption (serverless or SSL) but can provide your own PySpark or Spark scripts
    - Fully managed, cost effective, pay only for resources consumed: jobs run on serverless Spark platform
    - Could be event driven and provision additional DPUs to increase performance of underlying Spark jobs
    - Enabling job metrics can help you understand the maximum capacity in DPUs needed
    - Errors reported to CW and can tie into SNS for notification
    - Used for transform, clean, enrich data before doing analysis
    - ETL joins can run: made of Scala or Python code (pre generated or manually written)
    - Target: S3, JDBC (RDS or Redshift) or in Glue Data Catalog
    - Jobs are run on serverless Spark platform

    - Main data structure is a DynamicFrame, which is a collection of DynamicRecords
        - DynamicRecords (like a dataframe but with more ELT functionalities) are self describing, with a schema
        - Integrates with Python and scala apps
    - Transformations with Glue ETLs:
        - To run a transformation job on the data, run a Glue job that runs code
        - When running the job, AWS spins up a group of servers that runs the code and makes the transformations
        - Bundled transformations:
            - Filters to specify a function to folder records
            - Joins are used to enrich data
            - Map is used to add/delete fields or perform external lookups
        - ML transformations: find matches ML allows identifying duplicate / matching records in the dataset, even when records don't have common unique name and no fields match exactly
        - Format conversions: CSV, JSON, Arvo, Parquet, ORC, XML
        - ResolveChoice deals with ambiguities on a DynamicFrame and returns a new one:
            - 2 fields with the same name
            - make_cols: creates a new column for each unique type
            - cast: casts all values to a specified type
            - make_struct: creates a structure that contains each data type
            - project: projects every type to a given type
- Glue Scheduler: used to schedule jobs
- Glue Triggers: used to automate job runs based on events
- Modifying Glue Data Catalog from ETL jobs: ETL scripts can update schema and partition
    - Add new partitions / updating table schema: rerun the crawler or use enableUpdateCatalog/UpdateBehavior
    - Creating new tables: enableUpdateCatalog with SetCatalogInfo
    - Restricting: S3 only and limited to JSON, CSV, Arvo (Parquet formats and nested schemas not supported)
- Glue Development Endpoints: develop ETL scripts in a notebook and create ETL job to run the script with Spark
    - Endpoint is in a VPC controlled by security groups, connect via:
        - Apache Zeppelin on local machine, Zeppelin Notebook Server on EC2 via Glue, SageMaker notebook, terminal window, Pycharm professional or use Elastic IPs to assess a private endpoint address
- Running Glue Jobs: time based schedules and cron style
    - Jobs run on virtual resources: all resources needed to run ETL jobs are provisioned and managed in its own isolated service account to protect data in transit and at rest
        - Using these virtual resources allows us to segregate our data from other customer data
        - Provide input data sources and output data targets in VPC
        - Traffic governed by VPC: traffic in/out and within the Spark env is determined by networking policies
    - Running more DPU speeds up job processing but there is a threshold that adding more won't help
    - Job Bookmarks: persist state from job runs and prevents reprocessing of old data
        - When bookmarks are enabled, the Glue Job already knows which files it has scanned over and transformed so only new files will be scanned
        - It is a way to process new data without reprocessing old data: tracks already processed data by persisting the state of information from the job
        - Allows processing new data only when rerunning on a schedule
        - Works with S3 sources in a variety of formats and with relational DBs via JDBC (if primary keys in sequential order): only handles new rows (not updated rows)
        - By default, bookmarks are disabled which means the job always processed the entire dataset
        - CW Events: Launch a Lambda or SNS notification when ETL succeeds of fails, then income EC2 instance to do further processing or activate a Step Function
- Glue Studio: visual interface for ETL workflows
    - Visual job editor for defining / creating ETL workflows: create DAGs for complex workflows
    - Transform, sample, join data while supporting partitioning
- Glue Databrew: visual data preparation tool
    - UI for pre processing large datasets with input from S3, data warehouse or database, then output to S3
    - Over 250 ready made transformations but can create recipes of transformations that can be saved as jobs
    - Define data quality rules or create datasets with custom SQL from Redshift
    - Security: integrate with KMS, SSL in transit, IAM can restrict who can do what, CW, CloudTrail
- Elastic Views: build materialized views from Aurora, RDS, DynamoDB
    - These views can be used by Redshift, ElasticSearch, S3, DynamoDB, Aurora, RDS with SQL interface
    - Handles any copying or combining / replication data and Monitors for changes and continuously updates
- Security: configure Glue to only access JDBC through SSL
    - Data Catalog is encrypted by KMS and resource policies protect data catalog resources
    - Connection passwords are encrypted by KMS: data written by Glue has encryption modes

LakeFormation

- Built on top of Glue: easy to set up a secure data lake by loading data and monitoring dataflows
  - Set up partitions, encryption, manage keys, define/monitor transformation jobs, access control, auditing
  - No cost for LakeFormation but underlying services incur charges: Glue, EMR, Athena, S3
  - Does not support manifests in Athena or Redshift queries
  - IAM permissions on KMS encryption keys are needed for encrypted data catalogs in Lake Formation
  - IAM needed to create blueprints and workflows
- Building a data lake: create IAM user for data analysis, create Glue Connection to data sources, create S3 bucket for Lake, register S3 path in LakeFormation for Data Catalog/ grant permissions, use a blueprint for a workflow, run workflow, grant select read permissions
- Cross Account Lake Formation Permission: recipient must be set up a Data Lake admin
  - Use AWS resource access manager for accounts external to your organization
  - IAM permissions for cross account access
- Governed tables support ACID transactions across multiple tables: Incurs additional charges based on usage
  - Safely and concurrently do row level action without users: strict transactional guarantees under the good
  - New type of S3 table where have choice of governed can't be changed
  - Works with streaming data and can query with Athena (storage optimization with automatic compaction)
  - Granular access control with row and cell level security: both for governed and S3 tables

EC2 Instances

- Instance Groups:
  - Spot Instances: need to be able to tolerate loss, low cost, need a checkpointing feature
  - Reserved Instances: long running clusters
  - On Demand Instances: for the remaining workloads
  - Autoscaling van be leveraged for error and automated for DynamoDB, auto scaling groups
- Instance Types:
  - General Purpose: t2, t3, m4, m5
  - Compute Optimized: c4, c5 for batch processing, distributed analytics, machine and deep learning inference
  - Memory Optimized: r4, r5, xi, xid for high performance database, in memory database, real time big data analytics
  - Accelerated Computing: p2, p3, g3, f1 for GPU instances ,machine or deep learning, high performance computing
  - Storage Optimized: hi, i3, d2 for distributed file system, NFS, MapReduce, Apache Kafka, Redshift

EMR (Elastic Map Reduce)

- MapReduce is a technique to distribute workloads across different computing nodes so they can process different data and return information quicker than just on a single node
  - Maps workload evenly across nodes: files are split across secondary nodes (handled by primary), where each secondary node has mut;iple copies of each broken up files
  - Use customer code to process and analyze massively large datasets
- Hadoop:
  - MapReduce is the framework for distributed data processing that maps data to key/value pairs
    - Reduces intermediate results to final output, large supplement by Spark
  - YARN manages cluster resources for multiple data processing frameworks
  - HDFS distributes data blocks across the clutter in a redundant manner (ephemeral on EMR)
- EMR is a managed Hadoop framework that runs on EC2 instances (includes Spark, HBase, Presto, Flink, Hive)
  - Run distribution processing frameworks with flexibility of custom code while defining compute, size, memory
  - Charges by the hour plus EC2 charges and provisions new nodes if a core node fails
  - Add or remove task nodes on the fly: increases processing capacity, but not HDFS capacity
  - Resize a running clusters core nodes: increases both processing and HDFS capacity
  - Code nodes can be added or removed (removing code node risks data loss)
  - Expensive: pay for running EC2 instances and S3 usage
- Launch EMR Cluster: EMR cluster is a collection of EC2 instances (nodes) running Hadoop
  - Each node has a rule within the cluster called node types
  - Master Node: manages cluster by running software components to coordinate the distribution of data and tasks among other nodes for processing
    - Tracks status of tasks submitted to cluster andmonitors cluster health, core, task nodes
    - Master node is a single EC2 instance that is a leader node (need at least one)
    - In a single node cluster, have only a master node (Multiple primary nodes to avoid single point of failure)
    - Tracks and directs HDFS, responsible for resource management
  - Core Nodes: hosts HDFS data and runs tasks for the primary node
    - Scaled up and down, but with some risks
    - Coordinate data storage, knows how/where to store data

        - Multi node clusters have at least one core node
        - Able to add task nodes to perform parallel computational tasks on data to help
    - Task Nodes: run tasks but does not host data (optional)
        - No risk of data loss when removing (good use of spot instances)
        - Provide power to perform parallels computation tasks on data
        - Cluster resides in a single AZ because nodes in a cluster can communicate with one another fast so no need to traverse as much Internet
            - Block replication, finding files, communication between nodes, etc can happen faster
        - Transient cluster: as soon as job is run, cluster is terminated:
            - Create steps that run as soon as cluster is spun up
            - Total number of EMR processing hours per day < 24 -> shut down cluster when not in use
            - Good if not using HDFS primary data storage (EMRFS with S3), if processing job is intensive/iterable
    - Long running EMR cluster: good if frequently running processing obs so beneficial to keep cluster running
        - Good if jobs have output dependent on one another, more cost effective to store data on HDFS
- Usages:
    - Transient cluster terminates once all steps complete: loading data, processing, storing, shut down to save $
    - Long running cluster must be manually terminated:
        - Data warehouse with periodic processing on large datasets
        - Spin up task nodes using spot instances for temporary capacity
        - Use reserved instances on long running clusters to save month
        - Termination protection is on by default, with auto termination off
    - Frameworks and apps are specified at cluster launch
    - Connect directly to master and run jobs directly or submit ordered steps via console
        - Process data in S3 or DFS, output data to S3 and when defined, steps involved from console
- EMR/AWS Integration: IAM to configure permissions, CloudTrail to audit requests made
    - EC2 for instances that hosts the nodes, S3 to store input/output data
    - VPC to configure virtual network in which instances launched
    - CW to monitor cluster performance / configure alarms, Data Pipeline to schedule / start clusters
- EMR Storage:
    - HDFS: very fast, keeps multiple copies stored across cluster instances for redundancy (through hdfs://)
        - Files stored as blocks (128MB default size)
        - Ephemeral: HDFS data lost when cluster is terminated
        - Useful when caching intermediate results or workloads with significant random I/O
        - Hadoop tries to process data where it is stored on HDFS: split files into multiple chunks of smaller files
        - Use HDFS data block to assign tasks and uses file compression when possible
    - EMRFS: access S2 as if it were HDFS by R/W files directly to S3 (through s3://)
        - Allows feature rich persistent storage after cluster termination
        - EMRFS Consistent View: optional for S3 consistency
            - Uses DynamoDB to track consistency, may need to tinker with R/W capacity on DynamoDB
    - Local File System: ephemeral and suitable only for temporary data (buffers, caches, etc)
        - Instance storage: located through ~/ on disks that are attached to the host machine and used for very high I/O performance and high IOPS at a low cost
            - EBS volumes are located through ~/ and is used to extend HDFS (ephemeral)
        - EBS for HDFS: allow use of EMR or EBS/ only types (m4, c4) - Deleted when cluster is terminated
            - EBS volumes can only be attached when launching a cluster
            - If you manually detach an EBS volume, EMR treats that as a failure and replaces it
- Managed scaling: create custom scaling rules based on CW metrics but limited to instance groups only
    - Supports instance groups and instance fleets: scales spot, on demand instances in a savings plan within same cluster (available for Spark, Hivem Yarn workloads)
    - When reserving cluster annually, can't have less core nodes then the replication factor
    - When resizing cluster using AWS scaling, can automatically increase/decrease the number of instances in core/task nodes (depending on your workload)
        - Set min and max limit for the number of instances in your cluster
        - Create custom autoscaling policy: set min, max, on demand limit and max core nodes
    - Scale up strategy: first add core nodes, then task nodes up to max units specified
    - Scale down strategy: first removes task nodes, then core nodes
        - Spot nodes are always removed before on demand
- EMR Serverless: choose EMR release and runtime (Spark, Hive, Presto)
    - Submit queries and scripts via job run requests
    - EMR manages underlying capacity but you can specify default to worker sizes and pre initialized capacity
        - EMR compute resources needed for the job and schedule workers accordingly

        - All within one region, across multiple AZs
    - No need to estimate how many workers are needed for workloads: auto provisioned as needed
    - Pre initialize capacity: Spark adds 10% overheard to memory requested for drivers and executors
        - Be sure initial capacity is at least 10% more than requested by the job
- Security: IAM policies/roles, Kerberos for secure user authentication, SSH
    - Using EC2 key pair for SSH credentials and attach IAM roles to EC2 instances
    - EC2 security groups: one for master node and one for cluster node (core or task node)
    - EMR supports LUKs for at rest encryption
    - EMR always uses HTTPS to send data between S3 and EC2 but can also encrypt input data before uploading to S3 and decrypt it when EMR fetches data from S3 -> Predefined bootstrap actions to configure cluster on startup
- Choosing instance types: chosen according to workload (AWS limits to 20 EC2 instance per region)
    - Master Node: m5.xlarge if less than 50 nodes, large if greater than 50 nodes
    - Core and Task Nodes: m5.xlarge is usually good
        - m5 has good balance of CPU, disk space, I/O (for improved performance, use m4.xlarge)
        - c/z family for batch processing, CPU based
        - g/p family for graphics processing, GPU based
        - r family for Spark apps with in memory caching
        - h/i family fo large HDFS and MapReduce jobs requiring high I/O performance and high IOPS
        - If cluster waits a lot on external dependence, then t2.medium
    - Spot Instances: good for task nodes, only use on core/master if testing or cost sensitive (risk partial data loss)
- Choosing right number of instances: 1 primary node is fine, 3 primary nodes to maintain high availability
    - Core and Task Nodes: depends if doing a lot of processing (running asks) or storing data (HDFS)
        - 4-9 Core Nodes, then replication factor is 4 - 9 nodes
        - 1-3 Core Nodes, then replication factor is 1
        - Able to override the default replication factor upon cluster launch or later in HDFS use
    - AWS suggests using a smaller cluster of large nodes to reduce failure of possibilities and reduce the amount of maintenance (use on demand instances if data is important, spot to save money)
    - For long running apps: primary node should be on demand, core nodes as on demand, task nodes as spot
    - For cost driven apps: primary, core, task nodes should all be spot instances
    - For data critical apps: primary, core nodes should be on demand, task nodes can be spot
    - For app testing: entire cluster can be run on spot instances

Apache Spark
- Distributed processing framework for big data that includes in memory caching and optimizes query execution
    - Not meant for OLTP: supports Java, Scala, Python, R
    - Allows code reuse across batching, interactive queries (Spark SQL), real time analysis, ML, graph processing
- Spark components: all run on Spark console that handles memory management, fault recovery, scheduling, distributed and monitor jobs and interact with storage
    - Spark Streaming is integrated with Kinesis, Kafka on EMR for real time streaming analytics
        - With Spark structured streaming, picture data as a table that keeps growing
        - Query this data using windows of time
    - Spark SQL: up to 100x faster than MapReduce
    - MLib: classification, regression, clustering, pattern mining, collaborative filtering
    - GraphX: graph processing, ETL, analysis, interactive graph computation
- Spark apps are run as independent processes on a cluster:
    - Spark context (driver program) coordinates them through cluster manager: sends app code, tasks to executor
    - Executes run computations and store data: use a dataset that is a table of data (treat data on cluster)

Apache Hive on EMR
- Execute SQL code on underlying, unstructured data that resides in YARN:
    - Scalable: works with big data on a cluster, good for data warehouse
    - Easy OLAP queries, highly optimized and highly extensive
- Hive maintains a metastore that impacts a structure you define on unstructured data stored in HDFS:
    - Metastore stored on MySQL on masternode by default: external meta stores offer better resiliency, integration
- Load table partitions from S3, write tables in S3, load scripts from S3, DynamoDB as an external table

PIG on EMR
- Alternative interface to MapReduce: highly extensible with user defined function
- Able to use multiple file systems and use TARs/ scripts from S3
- Writing mappers and reducers by hand takes a long time: pig introduces pig latin (scripting language that uses SQL like syntax to define map and reduce steps)

HBase

- It is a non relational database for PB scale data: based on Google's BigTable on top of HDFS
- In memory with Hive integration: similar to DynamoDB (both are noSQL databases intended for the same thing)
    - HBase is efficient storage of sparse data, good for high frequency counters, high write/update throughput and more integration with Hadoop -> Store and backup data on S3 via EMRFS

Presto on EMR

- Connect to many different big data databases/data stores at once and query across them (Athena uses this under hood)
    - Interactive queries at PB scale: familiar SQL syntax
    - Optimized for OLAP: analytical queries, data warehousing
- Presto connectors: HDFS, S3, Cassandra, MongoDB, HBase, SQL, Redshift

Zeppelin and EMR Notebook

- Hosted notebook on cluster to run Python code: run Spark code interactively to speed up development cycle
- Execute SQL queries directly against Spark SQL and query results can be visualized in charts/graphs:
    - Makes Spark feel more like a data science tool
- EMR notebook is similar to Zeppelin, with more AWS integration: backed up to S3, provision clusters from DB:
    - Hosted inside a VPC and accessed only via AWS console

S3 Dist CP

- Tool for copying large amounts of data from S3 to HDFS or from HDFS into S3
- Uses MapReduce to copy in a distributed manager
- Suitable for parallel copying large number of objects: across buckets, across accounts

Hue, Splunk, Flume, Other tools

- Hue is Hadoop user experience which is a graphical front end for the apps on EMR cluster
    - IAM integration so Hue users can inherit IAM roles (data between HDFS and S3)
- Splunk makes machine data accessible, usable, valuable to everyone
    - Operations tool to visualize EMR and S3 data using EMR Hadoop cluster
- Flume is a way to stream data into your cluster, made from start with Hadop in mind
- MxNet is like Tensorflor, a library for building and accelerating NNs
- Ganglia for monitoring, Mahout for ML, Accumulo for noSQL database, Sqoop for relational DB connector
- HCatalog for table, storage management for Hive metadata, Kinesis Connector to directly access Kinesis streams
- Tachyon is a Spark accelerator, Derby is an open source relational DB in Java, Ranger is Hadoop's data security manager

Step Functions

- Design workflows, easy visualization, audit of history of workflows: max execution time of 1 year
    - Ability to wait for arbitrary amount of time and Advanced error handling and retry mechanism outside the code

AWS Data Pipeline

- Schedule tasks for processing big data: web service that processes and moves data between AWS services
    - Destinations: S3, RDS, DynamoDB, Redshift, EMR
    - Manages task dependencies, retries, notifies on failures, cross region pipelines, precondition checks
    - Data pipeline activity is an action initiated on your behalf in the pipeline: EMR, Hive, COPY, SQL, scripts
- Tool to automate movement / transformation of our data:
    - Helps process/move data between AWS compute/storage services and on premise data sources
    - Create ELT workflows to automate processing / movement of data at scheduled intervals
- It is a container that consists of:
    - Data node: where data is coming from, end destination for data
    - Activities are ways for steps of a pipeline to define work on perform
    - Preconditions are conditional statements that must be true before activity can run
        - Schedule are set up when pipeline should run
- AWS provisions and terminates EC2 instances or EMR clusters that actually do transformation, processing / moving of data from one source to another (terminates automatically when finished):
    - Run it within compute resources and on premise by installing task running server on local network to access local DB securely and pulled data pipeline for next task to run
    - When it recognizes a task to run, it runs on task runner installed on premise

DOMAIN 4: ANALYSIS

Kinesis Data Analytics

- System for continuously querying data streams: real time analytics using SQL queries
  - Receive data from source and set up time windows to aggregate data across
  - Source is kinesis data streams or Firehose streams or S3
  - Destination is Kinesis Data Streams or Firehose Streams, and from there to S3 or Redshift
    - If no data is received in the destination, then it could be an issue with IAM role or mismatched name for output stream or destination service is currently unavailable
- The default capacity of processing app in terms of memory is 32GB:
  - A single KPU provides 4GB of memory and default limit for KPUs is 8
- Errors are sent to error streams: if record arrives late to app during stream processing, it s written to error stream
- Reference tables can be an inexpensive way to join data for quick lookups: mapping is stored into S3
- Integrates with Lambda so it can be a destination as well:
  - Allows flexibility for post processing and opens up access to other services as destinations
- Uses Apache Flink under the good: Flink is a framework for processing data streams
  - Integrate Kinesis Data Analytics with Flink so use Flink app (instead of SQL) from scratch and load it into Kinesis Data Analytics via S3 (makes it serverless)
- Use cases: streaming ETL, continuous metric generation, responsive analytics
- Cost Model: Pay only for resources consumed (not cheap)
  - Charged by KPUs consumed per hour, serverless so scales automatically
  - Use IAM permissions to access streaming sources and destinations
- Random Cut Forest is SQL function for anomaly detection (identify outliers) on numeric columns in a stream
- Security: attach IAM role so it can read from Kinesis DA and references sources / write to output destinations

Athena

- Serverless, interactive query service for S3 via SQL that uses Presto under the hood:
  - Full RDBMS service backed in S3: SQL, fully featured interface with its own API
  - Supports CSV, JSON, ORC, Parquet, Arvo and unstructured / semi structured / structured data
  - Connects to external data sources with Athena federated queries and uses data source connectors (translates between external data source and Athena) to run on lambda to run these federated queries
  - Use cases: ad hoc queries of web logs, querying staging data before loading to Redshift, analyze logs in S3, integration with Jupyter, integration with QuickSight, ODBC/JDBC with other visualization tools
  - Integrates with Glue to put structure: Glue Crawler runs over S3 to extract schema and builds Glue Data Catalog that Athena uses to query data (from there, connect with Redshift or QuickSight)
  - Not used for: highly formatted reports, ETL, visualizations
- Cost model: pay as you go ($5 per TB scanned, successful or canceled queries count but failed queries don't)
  - No charge for DDL save money using columnar storage (Glue and S3 have their own charges)
- Workgroups: to organize users, teams, apps or workflows which integrates with CW, IAM, SNS
  - Control query access and track cost by workgroup
  - Each workgroup can have its own query history, data limits (limits how much data queries may scan by workgroup), IAM policies, encryption settings
- Security: access control using IAM, ACLs, S3 bucket policies
  - Encrypt results at rest in S3 staging directly (SSE-S3, SSE-KMS SSE-C)
  - Cross account access using S3 bucket policies, TLS encrypted in transit between Athena and S3
- Performance: uses columnar storage
  - Small number of large files performs better than a large number of small files
  - Use partitions: if adding partitions after table creation, use MSCK REPAIR TABLE
- ACID Transactions: provides strict guarantees about transactions
  - Add table_type = iceberg in the table command to benefit from periodic compaction to preserver performance
  - Concurrent users can safely make row level modifications (Removes need for custom record locking)
  - Provides time travel operations to recover data recently deleted with select statements

RedShift

- Fully managed, PB scale data warehouse service for fast querying, enterprise level reporting
  - 10x better performance via ML, parallel query execution, columnar storage, columnar compression
  - Designed for OLAP, cost effective with SQL, ODBC, JDBC interfaces
  - Scales up or down in demand: built in replication / backups
  - Running complex SL queries with multiple joins and subqueries
  - Pull data from many different data sources into a common format that is stored for long periods of time
  - Monitoring via CloudWatch and Cloudtrail

    - Integrates with S3, DynamoDB, EMR/EC2, DMS, Data Pipeline
    - Use cases: accelerate analytic workloads, unified data warehouse / data lake, data warehouse modernization, analyze global sales data, store historical data, analytic ad impressions and clicks, aggregate gaming data
    - Anti patterns: small datasets, OLTP, unstructured data, BLOB data
- Cluster is core component: leader node and more than 1 compute nodes (1 to 128 nodes, depending on the type)
    - User sends query to leader node, generates a query plan to generate execution scripts to complete that query
        - Execution scripts go to compute nodes: compute nodes go to their slices and each slice has a piece of data that it needs to return to the leader node which combines that data into query repair
        - Slice divides the compute, memory, storage of a node into separate pieces
            - There is a default number of slices per node (usually 2)
            - Each slice has its own compute, memory, storage
            - Slices receive a query script from leader node, executes script and returns the result
    - Leader node manages communication with client program and compute nodes: receive queries from clients, parses them, develops execution plans to process the queries
        - Coordinates processing with compute nodes, aggregates results, returns them to leader
    - Compute nodes execute steps in plans from leader node and transmitting data among themselves to complete queries, sends aggregations abc to leader node
        - Each compute node has its own compute/storage, depending on chosen node type
            - Dense storage for large DW using HDDs at a low price point for high performance
        - Within compute nodes, there are node slices so each portion of workload is split by slice
            - Number of slices per node is determined by node slice per cluster
- Node types:
    - Dense Compute: more compute / memory attached but storage attached to these nodes is going to be faster
        - For queries that need to be performed quickly with near real time analytics
    - Dense Storage: dense compute but more of it
    - RA3 Nodes: same amount of storage (backed by S3) with local storage cache to provide hot data for queries
        - RA3 nodes are optimized for managed storage: decouple compute / storage capabilities
- Durability: replicates all data within the cluster and backs it to S3 (asynchronously replicated to another region)
    - Automated snapshots, failed drives, nodes are automatically replicated, limits to a single AZ

- Scaling: supports vertical (increase node instance type) and horizontal (increase number of nodes) on demand
    - During scaling, a new cluster is created where the old one remains available for reads
    - CNAME is flipped to new clutter with a few minutes of downtime
    - Data is moved in parallel to new compute nodes
- Redshift Spectrum allows querying exabytes of unstructured data in S3 without loading:
    - Limitless concurrency, horizontal scaling, separate storage and compute resources
    - Supports wide variety of data formats and gzip/snappy compression
    - Interface to create foreign tables in Redshift cluster from data stored in S3
    - Uses external keywords for schema/table creation and supports select/insert operations (not update/delete(
    - Query comes from Redshift cluster and is sent to Redshift Spectrum which sends a data request through access control to data store (Glue, Athena, S3)
    - High utility service to save money
- Distribution Styles: data in table is distributed across many compute nodes and slices within those nodes
    - When data is loaded into a table, Redshift distributes table rows to compute nodes/slices by distribution type
        - Want to distribute data uniformly and minimize data movement during query execution
    - AUTO: Redshift figures it out based on data size (default)
    - EVEN: rows are distributed across slices in round robin
        - Regardless of column value, leader nodes distributes in a round robin fashion
        - Best when table does not participate in joins or no choice in other distribution styles
    - KEY: rows are distributed based on the column
        - Leader node puts matching values on same node slices and on same column stored together
    - ALL: entire table is copied to every node (best for slow moving tables that are not often updated)
        - Copying entire table is distributed to every node: every row is collated in every join or the table
        - Multiple storage required by number of nodes in cluster: takes a long time to load data from multiple tables
- Sort Keys: rows are stored on disk in sorted order based on the column designated as a sort key (like an index) to make it fast for range queries
    - Single Sort Key uses a single column to sort the data (best if querying primarily by 1 particular column)
    - Compound Sort Key are made with all columns defined in sort key definition, in the order they are defined
    - Interleaved Sort Key gives equal weight to each column or subset of columns (useful if different queries relies on different columns)

- Importing or exporting data:
  - Import using COPY command: read large amounts of data from multiple data files or streams simultaneously and read in data from S3, EMR, DynamoDB, remote hosts
    - S3 requires a manifest file and IAM role
    - If data is already in another table in Redshift, use INSERT INTO…SELECT or CREATE TABLE AS
    - Copy can decrypt data as it is loaded from S3 (use hardware accelerated SSL to keep it fast)
    - Gzip, izip, bzip2 compression is supported for speed up
    - Automatic compression: analyzes data being loaded, figures out optimal compression schema for storing it
    - Special case: narrow table (many rows, few columns)
      - Load with a single copy transaction if possible
      - Otherwise, hidden metadata columns consume too much space
  - Export using UNLOAD command to unload from a table into S3 files
  - Enhanced VPC routing forces all copy/unload traffic from cluster to repositories through VPC
- Copy snapshot from Redshift to another region (like cross region snapshot replication for backup):
  - KMS encrypted Redshift cluster snapshots:
    - In destination AWS region: create KMS key, specify a unique name for snapshot copy grant and specify KMS key id for which you are creating copy grant
    - In source region, enable copying of snapshots to the copy grant you just created
- DBLink: connect Redshift to PSQL in RDS instance
  - Copy and sync data between PSQL and Redshift: launch a Redshift cluster and RDS instance in same AZ, configure VPC to allow Redshift connection to RDS instance (allow DBLink connection with Redshift)
- Redshift Workload Management (WLM) helps prioritize workloads so that short, fast queries are not stuck behind long queries by creating query queues for each type (improves system performance)
- Concurrency Scaling: automatically adds cluster capacity to handle increase in concurrent read queries if you have burst access to Redshift cluster
  - Support virtually unlimited concurrent users and queries
  - WLM queries manage which queries are sent to the concurrency scaling cluster
  - Automatic WLM can define up to 8 queues (default is 5 queues with even memory allocation) so large queries will have concurrency lowered and small queries will have concurrency raised
    - Easy query queue can be configured according to priority, concurrency scaling mode, user groups, query groups and query monitoring rules
  - Manual WLM comes with 1 default queue with concurrency level of 5 (5 queries at once):
    - Super user queue with concurrency level of 1
    - Define up to 8 queues, up to concurrency level of 5:
      - Each can have defined concurrency scaling mode, concurrency level, user groups, query groups, memory, timeout, query monitoring rules
      - Enable query groups hopping for timed out queries to hop to next queue to try again
  - SQA (Short Query Acceleration) will prioritize short running queries over long running ones so short queries run in a dedicated space and won't wait in queues behind long queries:
    - Use in place of WLM queues for short queries
    - Use ML to predict a queries execution time and configure how many seconds is short
- VACUUM command recovers space from deleted rows:
  - VACUUM FULL is default to restore all rows and reclaim space from delete rows
  - VACUUM DELETE ONLY will only reclaim deleted space
  - VACUUM SORT ONLY will only restore the table
  - VACUUM REINDEX to reinitialize interleaved sort keys and then perform cleanup
  - Redshift automates some of the vacuuming options: monitors activity of cluster to determine when / what
- Deep Copy creates a new table that is an exact replica of old table: requires old table not be in active use
  - Sorts / removes any rows marked for deletion, then truncate the source table, takes all the data out, renames the new table and get clean table
    - Same effect as VACUUM FULL without being resource intensive
- Backup and restore: use a snapshot as a point in time backup of the entire cluster onto S3
- Resizing Redshift Clusters:
  - Elastic resize to quickly add or remove nodes of the same type (source cluster goes into read only):
    - Change node types, but not without dropping connections since it creates a whole new cluster
    - Cluster is down for a few minutes but tries to keep connections across the downtime
    - Limited to doubling or halving some node types
    - Utilizes a snapshot (creates one if no recent snapshot available): use to create any cluster
    - Cluster endpoint is changed to point to resized cluster
    - Takes minutes: snapshots are lazy loaded from S3 onto being a cluster
    - Constraints: cannot be used on a single node cluster and cluster must be within VPC

    - Classic Resize to change node types and/or the number of nodes (for single node cluster)
        - Cluster is read only for hours to days: moves the user schemes, table, data from source to resized cluster
            - Takes long time but it changes the cluster endpoint to the resized cluster (resume operations)
        - Snapshot, restore, resize to keep cluster available during a classic resize
            - Copy cluster and resize new cluster
- Cross region data sharing to share live data across Redshift clusters without copying: requires using RA3 node type for secure, across region / account data sharing
- Redshift Data Lake Export to dump Redshift query into S3 in Parquet format:
    - Parquet is 2x faster to unload and consumes 6x less storage
    - Compatible with Redshift Spectrum, Athena, EMR, SageMaker
    - Automatically partitioned
- AQUA is advanced query accelerator to make it faster for data in S3 to be queried from Redshift
    - Pushes reduction / aggregation queries closer to data for up to 10x faster, no extra cost, no cost changes
    - Benefits from high bandwidth connection to S3
- Security Concerns: using a hardware security module (HSM), then must use a client and a server certificate
    - If migrating unencrypted cluster to HSM encrypted cluster, create new encrypted cluster and move data to it
    - Defining access privileges for users or groups: use the grant or revoke commands in SQL
- Redshift Serverless: automatic scaling / provisioning for workload to optimize cost / performance
    - Uses ML to maintain performance across variable and sporadic workloads
    - Easy spin up of dev and text environment
    - Easy ad hoc business analysis
    - Need IAM role with a policy allowing redshift-serverless:*
    - Capacity measured in Redshift Processing Units (RPUs): pay for RPU hours per second and storage
        - Define base and maximum RPUs
- Security: VPC provides network isolation, cluster, security groups
    - Encryption in flight using JDBC driver enabled with SSL
    - Encryption at rest using KMS or an HSM device: need to establish a connection

DOMAIN 5: VISUALIZATION

QuickSight

- Offers fast, easy, serverless, cloud powered business analytics service for BI insight
    - Allows employees to build visualizations, perform ad hoc analysis, quickly get business insight from data
    - Data sources: Redshift, Aurora, rDS, Athena, EC2 hosted database, files on S3 or on-premise
        - Data preparation allowed limited ETL
    - Use cases: interactive, ad hoc exploration and visualization of data for dashboards and kpiS
        - Analyze and visualize data from logs in S3, on premise DBs, AWS, etc
    - Anti patterns: highly formatted reports, ETLs
- SPICIE imports datasets for fast, parallel, in memory and machine calculator engine that uses columnar storage, in memory and machine code generation to accelerate interactive queries on large datasets
    - Each year gets 10GB of SPICE by default: highly available and durable
    - Scales to hundreds and thousands of users
    - Accelerate large queries that times out in direct query mode (hitting Athena directly) but if it takes more than 30 minutes to import your data into SPICE, then it will timeout
- Security: offers MFA on your account with VPC connectivity (adds QuickSights IP address range to your DB security groups) and row level security:
    - Private VPC access: dat access to create IAM policies to restrict what data in S3 Quicksight can access
    - Resource access: ensure QS is authorized to use S3/Athena (managed in QS console)
    - To point QS to fetch DB stored on EC2 instance in VPC, add QS IP range to allowed IPs of the hosted DBs
    - Standard Edition: IAM users, email based accounts
    - Enterprise Edition: active directory, federated login, supports MFA, encryption at rest and in SPICE
- Integration with Redshift: QS can only access data stored in same region as the one QS is running in
    - Issue if QS is running in one region and Redshift in another region
    - VPC configured to work to access AWS
    - Solution: creates a new SG with inbound rule authorizing accessed from IP range of QS servers in that region
- User Management: users are defined via IAM or email
    - Active Directory connector with QS Enterprise Edition: all keys are managed by AWS and cannot use customer provided keys
- Pricing: annual subscription for standard and enterprise (pay extra for SPICE capacity)
- Dashboards are a read only snapshot of an analysis to share with other QS access
    - Share with embedded dashboards within an app (authenticate with active directory, SSL, Cognito)

- ML Insights: suggested insights to display ready to use insights
  - ML powered anomaly detection uses random cut forest to identify top contributors to changes in metrics
  - ML powered forecasting uses random cut forest to detect trends by excluding outliers / imputing missing values
- Answers business questions with NLP as add on: set up topics associated with data sets (must be NLP friendly)
- Visual Types:
  - Bar charts for comparisons and distribution, Line graphs for changes over time
  - Scalar plots and heat maps for correlation, Pie graphs / tree maps for aggregation
  - Pivot tables for tabular data, KPIs to compare key value to target value
  - Geospatial charts (map with dots), Donut charts for percentage of total amount
  - Gauge charts to compare values in a measure, Word clouds for word or phrase frequency
  - Tree maps for hierarchical aggregation

DOMAIN 6: SECURITY

Encryption

- Encryption in Flight (TLS/SSL): data is encrypted before being sent and decrypted after being received for communication between client and a server over a network
  - TLS Certificates help with encryption (HTTPS): ensures no main in the middle attack
- Server Side Encryption At-Rest: data is encrypted after being received by the server
  - Data is decrypted before being sent back to the client: stored in an encrypted form with a key
  - Encryption and decryption keys must be managed somewhere, and the server must have access to it
  - The encryption / decryption happens on the server
- Client Side Encryption: data is encrypted by the client and never decrypted by the server:
  - Data is decrypted by a receiving client: server should not be able to decrypt data
  - Leverage envelope encryption if want to encrypt data client side before uploading to S

IAM

- IAM policies dictate what permissions are given on other services:
  - Attach policies directly on a user: users can be grouped to connect to more than one user at a time
  - Attach policies to rules which can be assumed by users or services
  - Provides access to AWS services through trust relationships without an outside identity provider and supports federation through SAML 2.0
  - AWS using sign on can federate with external IdP or act as a central identity provider to federate external IdPs
  - IAM controls access to AWS API and integrates IAM controls with external IdPs

KMS

- Store and manage encryption keys, white encrypting and decrypting data
  - If keys expire or become invalid, use KMS to rotate those encryption keys
  - Send data to KMS that needs to be encrypted but latency can add up: works for infrequent encryptions but to not have network latency
- AWS managed software for encryption: supports symmetric and asymmetric encryption
  - Helps to store and manage encryption keys
  - Envelope encryption is used to encrypt data at rest:client wants to encrypt data so KMS generates a key that is encrypted with KMS an stores in the service, then generates a table of what data is encrypted with that
- Automatic Key Rotation for Customer Managed Keys (CMKs):
  - Use a CMK that can be generated within KMS or supplied by the customer to ensure full control of encryption
  - If enabled, key rotation happens every 1 year but previous key is kept active so can decrypt old data
  - New key has the same CMK ID but only the key is changed
- Manual Key Rotation if they should be rotated more often:
  - Ne key has a different CMK but keep previous key active to decrypt old data
  - Better to use aliases to ride the change of key for the application

CloudHSM

- AWS provisions encryption hardware: It is a dedicated hardware for you to manage your own encryption keys entirely
  - HSM device is tamper resistant: supports both symmetric and asymmetric encryption
  - No free tier available
  - Supports symmetric and asymmetric encryption
  - At rest encryption does not exist for RDS
- CloudHSM clusters are spread across multi AZ that must be setup
- Must use CloudHSM client software: good option to use with SSE-C encryption
- Redshift supports CloudHSM for database encryption and key management

Secrets Managers
- Stores text based security tokens, where secret access is controlled via IAM and secrets can be encrypted at rest
- Utilizes a secrets vault to keep authentication material out of our code so code becomes more flexible

STS and Cross Account Access
- STS (Security Token Service) allows granting limited and temporary access to AWS resources
  - Token is valid for up to 1 hour (must be refreshed)
  - Cross account access allows users from one AWS account to access resources in another
  - Federation (Active directory): uses SAML
    - Provides a nonAWS user with temporary AWS access by linking users Active Directory credentials
    - Allows SSO which enables users to log in to AWS console without assigning IAM credentials
  - Federation with 3rd party providers / Cognito (mainly used in web/mobile apps)
  - To gain access to a role in another AWS account, use sTS to gain temporary credentials
- Cross Account Access: Define IAM role for another account to access and define which accounts can access it
  - Use STS to retrieve credentials and impersonate IAM role you have access to using AssumeRole API
  - Temporary credentials can be valid between 15 minutes to 1 hour

Identity Federation
- Federation lets users outside of AWS to assume temporary role for accessing AWS resources:
  - Users assume identity provided access role
  - Federation assumes a form of 3d party authentication (no need to create IAM users)
- SAML federation for enterprises: use to integrate with Active Directory, SDFS with AWS
  - Provides access to AWS console or SLI through temporary credentials
  - No need to create an IAM user for each of your employees
- Customer Identity broker app for enterprises: use only if identity provider is not compatible with SAML
  - Identity broker must determine the correct IAM policy
- AWS Cognito (federated identity pools) for public applications:
  - Provide direct access to AWS resources from the client side
  - Do this by: log into federated identity providers or remain anonymous
    - Get temporary AWS credentials back from federated identity pool
    - These credentials come with predefined IAM policy stating their permissions

CloudTrail
- Provides governance, compliance, audit for AWS accounts (enabled by default)
- Provide history of events / API calls made within AWS account via console, SDK, CLI, AWS
- Get logs from CloudTrail into CW: CloudTrail logs have SSE-S3 encryption when in S3
- If a resource is deleted in AWS, look for it in CloudTrail first: shows past 90 days of ACIDity
- Default UI shows create, modify, delete events so need to enable CloudTrail logs to get a derailed list of all the events you want: able to store these events in S3 for further analysis and can be region specific or global

VPC Network Security
- VPC is network space that lives within a region with top level CIDR defined and AZs are within that region:
  - Subnets are within AZs and are all interconnected (some can have a path out to the Internet)
  - VPC Flow Logs generate a log of network traffic
- First layer of network security is NACL: attached at subnet edges and at stateless
  - Subnet ca be associated with 1 NACL
  - NACL rules: actions can be allow or deny
    - Source / destination specified with CIDR notation
    - Rule parts can be specified as ranges
    - Rules can apply to single protocol, IPv4 and IPv6 traffic type can be specified
    - Rules are checked in order by the number and once matched, the checking is stopped
- Next layer is security groups: attached at the resource network interface
  - Only have allow rules: they are stateful and keep track of connections flowing through them
  - Create inbound and outbound rules that specify source or destination CIDr

VPC Endpoints
- Endpoints allow you o connect to AWS services using a private network, instead of www network
  - Scale horizontally, are redundant and remove the needs of IGW to access AWS services
  - Use to access the AWS service from pirate subnets without the need to have an outgoing internet connection
- Gateway: provisions a target and must be used in a route table (only S3 and DynamoDB)

- Interface / VPC Private Link: provisions an ENI (private IP addresses) as an entrypoint but must attach security group for most AWS services

DOMAIN 7: OTHER

AWS AppSync
- Creates a graph QL API on top of DynamoDB, Aurora, Lambda, ElasticSearch, HTTP
  - graphQL is an alternative to REST, etc for applications
  - Creates graphQL endpoints for your apps to talk to

Amazon Kendra
- Enterprise search with natural language: combines data from file systems, share point, etc into one searchable repository (ML powered, uses thumbs up/down feedback)
- Relevance tuning to boost strength of document freshness, new counts, etc

AWS Data Exchange
- Find, subscribe to and use 3rd party data in the cloud
- Once subscribed to a data product, issue the AWS data exchange API to load data directly into S3 and then analyze it with a wide variety of AWS analytics and ML services

Amazon AppFlow
- Fully managed integration service that enables you to security transfer data between SaaS apps and AWS:
  - Sources: Salesforce, SAP, Zendesk, Slack, etc
  - Destinations: AWS services (S3, Redshift) or non AWS (Snowflake or Salesforce)
  - Frequency: on a schedule, in response to events or on demand
  - Has data transformation capabilities like filtering and validation
  - Encrypted over public internet or privately over AWS Private Link
  - No need to spend time writing the integrations and leverage APIs immediately