Architecture Deep Dive

* Monolithic Architecture
  + Fails together
  + Scales together
  + Billed together
* Tiered Architecture
  + Each tier can be scaled independently
  + Each tier connects to each
  + Enables horizontal scaling of individual tiers since a single endpoint (LB) is used to direct traffic between the tiers
  + Each tier has to be running something for the app to function
    - Ex. Upload expects and REQUIRES Processing to respond
* Event Driven Architecture
  + Producers, Consumers, or BOTH
  + No constant running or waiting for things
  + Producers generate events when something happens
    - Clicks, errors, criteria met, uploads, actions
  + Events are delivered to consumers
    - Generally via Event Router
    - Actions are taken and the system returns to waiting
  + Mature event-driven architecture only consumes resources while handling events
    - serverless

AWS Lambda In-depth – Part 1

* Function-as-a-Service (FaaS) – short running & focused
* Lambda function – a piece of code lambda runs
* Functions use a runtime – e.g. Python 3.8
* Functions are loaded and run in a runtime environment
* The environment has a direct memory (indirect vCPU) allocation
* You are billed for the duration that a function runs
* A key part of Serverless architectures
* When a function is invoked, deployment package is sent to a Lambda environment. – 50 MB zipped, 250 MB unzipped
  + Once arrived, executes within the runtime environment
* Common Runtimes – Python, Ruby, Java, Go, C#
* Custom runtimes such as Rust are possible using Layers
* Stateless – every time a function is invoked, a new runtime environment is provided
  + Above is default, can be configured for multiple invocations in a single runtime envionment
* You directly control the memory allocated for Lambda functions whereas vCPU is allocated indirectly
  + – 128 MB - 10240 MB
* 1769 MB = 1vCPU
* Runtime environment has disk memory mounted – 512 MB storage available as /tmp – default.
  + Can scale up to 10240 MB
* Lambda function can run max 900s (15minutes)
  + Function timeout
* Use step functions for longer function flows
* Execution Roles – IAM roles assumed by the Lambda function which provides permissions to interact with other AWS services
* Common uses
  + Serverless Applications (S3, API Gateway, Lambda)
  + File Processing (S3, S3 Events, Lambda)
  + Database Triggers (DynamoDB, Streams, Lambda)
  + Serverless CRON (EventBridge/CWEvents + Lambda)
  + Realtime Stream Data Processing (Kinesis + Lambda)

Lambda In-depth Part 2

* Networking modes
  + Public or VPC
* Public – by default, functions are given public networking. They can access public AWS services and the public internet
  + E.g. SQS DynamoDB
* Public networking offers the best performance because no customer specific VPC networking is required
* Public Lambda functions have no access to VPC based services unless public IPs are provided and security controls allow external access
* Typically the desired network configuration
* Private (VPC)
  + Lambda is configured to execute in a private subnet
* Private Lambda functions running in a VPC obey all VPC networking rules
  + No access to internet based endpoints unless proper networking and permissions are configured
* Use GWLB or VPC endpoints to access public endpoints (AWS + external)
* NatGW and Internet GW are required for VPC lambdas to access internet resources
* Private lambda needs ec2 network permissions (execution role)
* Security
  + Lambda Execution roles are IAM roles attached to lambda functions which control the PERMISSIONS the Lambda function RECEIVES
  + Lambda Resource Policy controls WHAT service and accounts can INVOKE lambda functions
    - E.g., sns, s3, or external services
    - Only configurable via the CLI
* Logging –
  + Lambda uses CloudWatch, CloudWatch Logs & X-Ray
  + Logs from Lambda executions – CloudWatch Logs
  + Metrics – invocation success/failure, retries, latency – stored in Cloud Watch
  + Lambda can be integrated with X-Ray for distributed tracing
  + CloudWatch Logs requires permissions via Execution Role – log information into CloudWatch Logs

Lambda In-depth Part 3

* Invocation
  + Synchronous invocation
  + Asynchronous invocation
  + Event Source mappings
* Synchronous Invocation – waits for complete/response pass/fail
  + CLI / API invoke a lambda function, passing in data and wait for a response
    - Lambda function responds with data or fails
  + Lambda + API Gateway
    - Client communicates with API GW – Proxied to Lambda function
  + Result (Success or Failure) – returned during the request
  + Errors or Retries have to be handled within the client
  + Synchronous Invocation is typically used when a human is directly or indirectly invoking a Lambda function
* Asynchronous Invocation – no waiting for complete
  + Typically used when AWS services invoke Lambda functions
  + Example - S3 isn’t waiting for any kind of response. The event is generated and S3 stops tracking
  + If processing of the event fails, Lambda will retry between 0 and 2 times (configurable). Lambda handles the retry logic.
  + The lambda function code needs to be idempotent reprocessing as a result should have the same end state
    - Not additive/subtractive – use declarative
  + Events can be sent to dead letter queues after the automatic (configured) retries repeatedly fail processing.
  + Destinations – events processed by lambda functions can be delivered to another destination
    - SQS, SNS, Lambda & EventBridge – where successful or failed events can be sent.
* Event Source Mapping Invocation –
  + Typically used on streams or queues which don’t support event generation to invoke lambda (polling required)–
    - Kinesis
    - DynamoDB Streams
    - SQS
  + Source Batches – data that is read (polled) from the source then processed by Lambda – size considerations – lambda times out after 900s (15 mins).
  + Because the services aren’t generating events, the data (source batch) requires permissions from the Lambda Execution role
  + Event Source Mapping Permissions from the lambda execution role are used by the event source mapping to interact with the event source
  + SQS queues or SNS topics can be used for any discarded or failed event batches – DLQ Failed Events (Failed Batch)
* Lambda Versions –
  + Lambda functions have versions – v1, v2, v3
  + A version is the code + the configuration of the lambda function
  + It’s immutable – it never changes once published & has its own Amazon Resource Name
  + $Latest points at the latest
  + Aliases (ex. Dev, Stage, Prod) point at a version – can be changed
* Execution Context –
  + The environment a lambda function runs in
  + A cold start is a full creation and configuration including function code download
    - 100s of ms – slow if interacting with humans, but OK for processing events from S3, SQS, etc.
  + A warm start, the same execution context is reused. A new event is passed in but the execution context creation can be skipped.
    - 1-2 ms
  + A Lambda invocation can reuse an execution context but has to assume it can’t. If used infrequently contexts will be removed. Concurrent executions will use multiple (potentially new) contexts
  + Provisioned Concurrency
    - AWS will create and keep X contexts warm and ready to use – Improving start speeds
  + Use the /tmp space to predownload something in the context
    - Or DB connections

CloudWatch Events and EventBridge –

* If X happens, or at Y time(s) - do Z
* EventBridge is CloudWatch Events v2 (\*)
* A default Event bus for the account – used by both
* In CloudWatch Events this is the only bus (implicit)
* EventBridge can have additional event busses
* Rules match incoming events – or schedules
* Rule matches – Route the events to 1+ Targets – e.g Lambda
* Event Pattern Rule – event matches In Event Bus
* Schedule Rule – chron format
* Events are JSON format

Serverless Architecture –

* Serverless isn’t one single thing
* You manage few, if any servers – low overhead
* Applications are a collection of small & specialized functions
* Functions run in Stateless and Ephemeral environments – duration billing
* Event-driven – consumption only when being used
* FaaS is used where possible for compute functionality
  + No consistent use for compute
* Managed services are used where possible
  + S3, DynamoDB, OIDC

Simple Notification Service (SNS) –

* Public AWS Service – Network connectivity with Public Endpoint
* Coordinates the sending and delivery of messages
  + Messages are <= 256KB payloads
* SNS topics are the base entity of SNS
  + Permissions and configurations
* A publisher sends messages to a TOPIC
* TOPICS have Subscribers which receive messages
  + Message formats –
    - HTTP(s) Endpoints
    - Email – JSON
    - SQS Queues
    - Mobile Push Notifications
    - SMS Messages
    - Lambda Functions
* SNS uses across AWS notification – CloudWatch & CloudFormation
* ASG can also be configured to scale based on SNS notifications
* Can be accessed from public internet and other vpcs with proper networking configuration
* By default, all subscribers receive all messages published to the topic
  + Configure filter on subscribers to control
* Fanout – single SNS topic with multiple SQS queues as subscribers
  + Create multiple related workloads
* Features
  + Delivery Status – HTTP, Lambda, SQS
  + Delivery Retried – Reliable Delivery
  + HA and Scalable – Regionally Resilient
  + Server Side Encryption – SSE
  + Cross-Account access – via TOPIC Policy (resource policy)

Step Functions –

* Long – running serverless workflow-based applications.
* Some problems with Lambda
  + Lambda is FaaS
  + 15 minute max execution time
  + Can be chained together – gets messy at scale
  + Runtime Environments are stateless
* Step Function State Machines
  + Serverless workflow - START –> STATES -> END
  + States are THINGS that occur
  + Maximum Duration – 1 year
  + Standard Workflow (Default) – 1 year execution limit
  + Express Workflow – 5 minute
    - High volume event processing workloads
    - IOT, Streaming, Mobile app backend
    - Highly transactional
  + Started via API Gateway, IOT Rule, EventBridge, Lambda
  + Generally used for backend processing
  + Amazon States Language (ASL) – JSON Template
  + IAM Role is used for permissions
  + States –
    - Succeed & Fail –
    - Wait – waits for certain period of time or specific date and time
    - Choice – allows the state to change based on the input
      * Email Only
      * Email and SMS
      * SMS Only
    - Parallel –
    - Map – takes action based on the item/input
    - Task – single unit of work performed by a State Machine – integrates with:
      * Lambda
      * Batch
      * DynamoDB
      * ECS
      * SNS
      * SQS
      * Glue
      * SageMaker
      * EMR
      * Step Functions

API Gateway 101 –

* Create and Manage APIs
* Endpoint/entry-point for applications
* Sits between applications & integrations (services)
* Highly available, scalable, handles authorization, throttling, caching, CORS (secure cross domain), transformations, OpenAPI spec (third-party API integration) direct integration with a range of AWS services.
* Public Service
  + Can connect to services/endpoints in AWS or on-premises
* Supported types:
  + HTTP APIs
  + REST APIs
  + WebsSocket APIs
* Phases –
  + Request
    - Authorize, Transform, Validate
  + Client
    - Action taken via integrated service – DDB, SNS, Step Functions, HTTP Endpoints, Lambda
  + Response
    - Transform, Prepare, Return
* CloudWatch Logs can store and manage full state request and response logs.
* CloudWatch can store metrics for client and integration sides
* API Gateway cache can be used to reduce the number of calls made to backend integrations and improve client performance
* API Gateway Authentication
  + Cognito User Pools
    - Client authenticates with cognito and receives a token if successful auth, passed to services
  + Lambda Based Authorization
    - Formerly known as Custom Authentication
    - Client calls API Gateway with a bearer token (ID)
    - Lambda authorizer called – checks local user store or external ID provider
      * IAM policy and principal identifier
    - API Gateway handles return request via Lambda integration or returns error 403 ACCESS\_DENIED to client
* API Gateway Endpoint Types
  + Edge Optimized
    - Any incoming requests are routed to the nearest CloudFront POP (Presence)
  + Regional
    - Clients in the same region
    - Doesn’t use CF network
    - Low overhead
  + Private
    - Only accessible within a VPC via an interface endpoint
* API Gateway Stages
  + APIs are deployed to stages – each sage has one deployment
  + Rollback supported
  + Canary deployments
    - If enabled – deployments are made to the canary not the stage
    - Canary = sub part of a Stage
    - Stages enabled for canary deployments can be configured so a percentage of traffic is sent to the canary.
      * This can be adjusted over time or the canary can be promoted to make it the new base stage
* API Gateway – ERRORS
  + 4XX – Client Error – Invalid request on client side
  + 5XX – Server Error – Valid request, backend issue
  + 400 – Bad Request – Generic
  + 403 – Access Denied – Authorizer denies OR WAF Filtered
  + 429 – API Gateway can throttle – this means you’ve exceeded that amount (configured)
  + 502 – Bad Gateway Exception – bad output returned by lambda (backend compute error)
  + 503 – Service Unavailable – backing endpoint offline? Major Service issues
  + 504 – Integration Failure/Timeout – 29s (Backed by Lambda/Limitation of Lambda)
* API Gateway – Caching
  + Caching is configured per Stage
  + Cache TTL default is 300 seconds
    - Configured 0 and max 3600s
  + Can be encrypted
  + Cache size – 500MB to 237 GB
  + Calls are only made to backend integrations if request is a cache miss
  + Benefits – Reduced load & cost + improved performance

Simple Queue Service (SQS) –

* Public, Fully Managed, Highly – Available Queues
  + Standard or FIFO
* Messaged up to 256 KB in size
  + Alternatively, use S3 and link it if data is larger – not best case, but workaround
* Received messaged are hidden (VisibilityTimeout)
  + Then either reappear (retry) or are explicitly deleted
* Dead-Letter queues can be used for problem messages
  + Allow you to perform different sets of actions based on the type of message
* Use
  + Decouple application components
  + Allow asynchronous messaging
  + Implementation of worker pools
  + ASGs can scale and Lambdas invoke based on Queue length
* Standard = at least once
* FIFO = exactly once
* FIFO (Performance) - 3,000 messages per second with batching, or up to 300 messages per second without
  + Not ideal for highly performant applications due to limited scaling capabilities
* Standard – nearly infinite + ability to scale greatly
* Billed based on “requests”
* 1 request
  + 1-10 messages
  + Up to 64 KB total
* Short polling (immediate) vs. Long (waitTimeSeconds) Polling
* Short polling
  + One request – 0 or more messages, if 0 then it still consumes a request and immediately returns empty request
* Long Polling
  + Specify a wait time seconds
    - Up to 20 seconds
    - If messages are available upon request, it will process
    - If no messages are available, it will wait for a maximum of 20 seconds for a message to arrive to the queue
* 14 day max for messages in queue
* Encryption at rest (KMS) & in-transit
  + In-transit encryptyed by default
* Access to a queue is based on Identity Policy or Queue Policy
  + Use Queue (resource) Policy for external account access

SQS Standard vs. FIFO

* FIFO
  + 300 TPS w/o Batching or 3,000 TPS with
    - Batching allows up to 10 messages to be processed per request
  + Exactly-Once Processing – duplicates are removed
  + Message order is strictly preserved – First In First Out
  + Useful for –
    - Workflow ordering
    - Command ordering
    - Price adjustments
* Standard
  + Scalable, as wide as required – Near Unlimited TPS
  + Best-Effort Ordering – no rigid preservation of message order
  + At-Least-Once Delivery, there could be on occasion be more than one copy of a message
  + Useful for –
    - Decoupling
    - Worker pools
    - Batch for future processing

SQS Delay Queues –

* Provide an initial period of invisibility for messages. Predefine periods can ensure that processing of messages doesn’t begin until this period has expired.
* Visibility Time Out vs. Delay
  + VisibilityTimeOut
    - SendMessage -> ReceiveMessage -> Processing -> DeleteMessage Or Not
    - Reappears on Queue if not Deleted. Allows for reprocessing
    - Default VisibilityTimeOut is 30 seconds
      * VisibilityTimeOut Min = 0, Max = 12 hours
      * Set on Queue or Per-Message
  + Delay
    - SendMessage -> ReceiveMessage -> No Message
    - A Delay Queue has a DelaySeconds set
    - Messages added to the queue will be invisible for DelaySeconds
    - DelaySeconds Min = 0, Max = 15 Minutes
    - Message Timers allow a per-message invisibility to be set, overriding any queue setting. Same Min/Max (0s-15min)
      * Not supported on FIFO Queues
* Delay Queues –
  + Perform a set of tasks before processing
* Visibility Queues –
  + Automatically reprocess any failed / unsuccessful

SQS Dead-Letter Queues

* Retrieve Policy –
  + Specifies the Source Queue, the Dead-Letter Queue, and the conditions where messages will be moved from one to the other
    - Define maxReceiveCount
  + maxReceiveCount –
    - When ReceiveCount is greater than maxReceiveCount & a message isn’t deleted – it’s moved to the Dead-Letter Queue.
* Allows you to configure an alarm for any messages sent to a DLQ
* Separate area to examine logs / content for diagnosis and mitigation for problematic messages
* All SQS queues have a RetentionPeriod – when a message is added to a queue, it has an enqueue time stamp.
  + When moving to DLQ, original enqueue timestamp is used
* Enqueue timestamp of message is unchanged. Retention period of a DLQ is generally longer than the Source Queues.
* A single DLQ can be used for multiple Source Queues

Amazon Kinesis Data Streams

* Kinesis Concepts
  + Kinesis is a scalable streaming service
  + Producers send data into a kinesis stream
  + Streams can scale from low to near indinite data rates
  + Public service & highly available by design (Regional)
  + Streams store a 24-hour moving window of data – (offers some level of persistence)
    - Default
    - Can be increased to a maximum of 365 days – additional costs
    - Data persists during this window
      * Kinesis Data Record (1MB)
      * Stored across shards
  + Supports multiple producers and multiple consumers
  + Multiple consumers access data from the moving window
    - Great for analytics and dashboards
  + Scales using shards
    - A stream starts off by using 1 shard
    - Each shard adds 1 MB ingestion / 2 MB consumption
  + Kinesis Data Firehose – moves data from stream enmasse to another AWS service
    - E.g., S3 + EMR
* SQS vs Kinesis
  + SQS 1 Production Group, 1 Consumption Group
  + SQS – Decoupling and Asynchronous communications
  + SQS – No persistence of messages, no window
  + Kinesis – Huge scale ingestion of data
  + Kinesis – Multiple consumers – rolling window
  + Kinesis – Data Ingestion, Analytics, Monitoring, App Clicks

Amazon Kinesis Data Firehose

* Stream based delivery service capable of delivery high throughput streaming data to supported destinations in near real-time
* Fully managed service to load data for data lakes, data stores, and analytics services
* Automatic scaling – fully serverless – resilient
* Near Real Time delivery (~60 seconds)
* Supports transformation of data on the fly (Lambda)
  + Can add latency depending on the complexity of the processing
* Billing – data volume through firehose
* Valid Destinations
  + HTTP Endpoints (third-party)
  + Splunk
  + RedShift
  + ElastiSearch
  + S3
* Can directly accept data from producers or obtained from a Kinesis Data Stream
* Solution for data persistence in Kinesis. By default, Kinesis doesn’t persist (store) data outside of the rolling window (24 hour default – maximum 365 days). Use Kinesis Data Firehose to move the data from Kinesis Data Stream to a data store.
* Producers can send data directly to Kinesis Firehose – couple with Kinesis Data Stream for Real Time processing. By default, ~60 second delay for data processed by Kinesis Data Firehose.
* Producers can send records to data streams or send directly at firehose. Firehose can also read from a data stream as a consumer.
* Firehose offers near realtime delivery. Delivery when buffer size in MB (1) or Buffer interval in seconds (60) passes.
* Lambda Transformation Function (Blueprints)
  + Optionally store the unmodified data in an S3 bucket
  + Transformed records sent to Valid Destinations
* When using RedShift – Delivered to S3 as an intermediary, RedShift COPY into RedShift.
* Common use cases –
  + Persistence to data coming into Kinesis Data Stream
  + Store data in a different format – Lambda Transformation Function (Blueprints)
  + Store data in a Valid Destination – HTTP Endpoint (Third-Party), Splunk, RedShift, ElastiSearch

Amazon Kinesis Data Analytics

* Real time data processing product
* Easiest way to analyze streaming data, gain actionable insights, and respond to business and customer needs in real time.
* Uses SQL
* Ingests from Kinesis Data Streams or Firehose
  + Optionally pull in referenced data from S3
* Valid Destinations –
  + Firehose
    - Including valid firehose destinations
      * S3, Redshift, ElasticSearch, & Splunk
    - Data becomes NRT and not Real Time
  + AWS Lambda – Real-time
  + Kinesis Data Streams – Real-time
* Sits between input and output streams, adjust data using SQL between input and output
* When and Where (use cases)
  + Streaming data needing real-time SQL processing
  + Time-series analytics – elections / e-sports
  + Real-time dashboards – leaderboards for games
  + Real-time metrics – Security & Response teams
  + Complex manipulation of data in real-time
    - Firehose only offers NRT using Lambda

Amazon Kinesis Video Streams

* Securely stream video from connected devices to AWS for analytics, Machine Learning, playback, and other processing.
* Kinesis Video Streams automatically provisions and elastically scales all the infrastructure needed to ingest streaming video data from millions of devices.
* Ingest live video data from producers
* Producers
  + Security Cameras, Smartphones, Cars, Drones, Time-serialized audio, Thermal, Depth, and RADAR data.
* Consumers can access data frame-by-frame – OR as needed
* Can persist and encrypt (in-transit and at rest) data
* Can’t access data directly via storage – only via APIs
* Integrates with other AWS Services – e.g., Rekognition (Facial Recognition) and Connect (VM or Audio Recordings)
* Kinesis Video Streams – 1 Per camera
* Kinesis Data Stream receives analysis data of the video (from Rekognition), and vs. the face collection (from Rekognition) identifying any Detected or Matched faces.
  + Lambda Analysis on every record
  + Send notification for unknown detections
* Useful for Live video streaming analytics – GStreamer, RTSP, etc.

Amazon Cognito – User and Identify Pools

* Authentication, Authorization, and User Management for web/mobile apps
* User Pools – Sign-in and get a JSON Web Token (JWT)
  + Most AWS services cannot use JWTs
  + Don’t grant access to AWS services
  + User directory management and profiles, sign-up and sign-in (customizable web UI), MFA, and other security features
  + Customized workflows and user migration – Lambda Triggers
* Identity Pools – Allow you to offer access to Temporary Credentials
  + Unauthenticated Identities – Guest Users
  + Federated Identities – SWAP – Google, Facebook, Twitter, SAML2.0 & User Pool for short term AWS Credentials to access AWS Resources
* User Pools –
  + Authenticate (JWT)
  + User Pool Token used as a proof of authentication
  + User pool tokens (JWT) can be used to access self-managed server based resources
  + User Pool tokens can grant access to APIs via Legacy lambda Custom Authorizers
    - Alternatively, directly via API Gateway
* Identity Pools –
  + External Identity
  + Authenticate to 3rd party IDP directly and get a token
  + Application passes token to Identity Pool
    - Configured to support each type of token
  + Authenticated Role
  + Unauthenticated Role
  + Cognito assumes IAM role defined in Identity Pool and returns temporary AWS credentials
  + Temporary credentials (Authenticated or Unauthenticated Role) to access AWS resources.
  + Swap Identity tokens from external IDP for temporary access to AWS resources
    - Web ID Federation
* User & Identity Pools combined
  + User pool abstracts requirement to manage/configure support for external ID
  + Benefit –
    - less admin overhead, near
    - unlimited max number of users (bypass the 5,000 max IAM user limit)

AWS Glue

* Fully managed serverless extract, transform, and load (ETL) service that makes it easy for customers to prepare and load data for analytics.
* Point AWS Glue to your data stored on AWS, AWS Glue discovers the data and stores the associated meta data (e.g., table definition and schema) in the AWS Glue Data Catalog. Once cataloged, the data is immediately searchable, queryable, and available for ETL.
* Serverless ETL (Extract, Transform, & Load)
* Vs. Datapipeline (which can do ETL), but uses compute(EMR Servers)
* Moves and transforms data between source and destination
* Crawls data sources and generates the AWS Glue Data Catalog
* Data Sources
  + Stores
    - S3, RDS, JDBC Compatible & DynamoDB
  + Streams
    - Kinesis Data Stream & Apache Kafka
* Data Targets
  + S3, RDS, JDBC (Java Database Connectivity) Databases
* Data Catalog
  + Collection of meta data
  + Persistent metadata about data sources in a region
  + One catalog per region per account
  + Avoids data silos – improves visibility
  + Amazon Athena, Redshift Spectrum, EMR & AWS Lake Formation all use Data Catalog
  + Configure crawlers for data sources – give credentials + point at sources
* Crawlers connect to datastores, determine schema and create metadata in the data catalog
* When resources are required, glue allocates from a AWS Warm Pool to perform the ERL processes
* Glue Jobs can be initiated manually or via events using Event Bridge

Amazon MQ 101

* SNS and SQS are AWS Services – using AWS APIs
* SNS Provides TOPIC and SQS provides QUEUES
  + Public Services, Highly Scalable, AWS Integrated
* Many ORGS already use Topics and Queues
  + And want to Migrate into AWS
* SNS and SQS won’t work out of the box
  + We need a standards compliant solution for migration
* Amazon MQ
  + Open-source message broker
  + Based on Managed ApacheMQ
  + JMS API Protocols –
    - AMQP
    - MQTT
    - OpenWire
    - STOMP
  + Provides QUEUES and TOPICS
  + One-to-one OR One-to-many
  + Provided message broker servers
    - Single Instance – Test, Dev, Cheap
    - HA Pair – Active/Standby
  + VPC Based – NOT A PUBLIC Service – Private networking required
    - Requires VPN or Direct Connect
  + No AWS native integration – delivers ActiveMQ product which you manage
* Amazon MQ vs SNS+SQS
  + SNS or SQS for most new implementations (default to this)
  + SNS or SQS if AWS integration is required – logging, permissions, encryption, service integration)
  + Amazon MQ if you need to migrate from an existing system with little to no application change
  + Amazon MQ if APIs such as JMS or protocols such as AMQP, MQTT, OpenWire, and STOMP are needed
  + Remember you need private networking for Amazon MQ – VPN or Direct Connect

Amazon AppFlow

* Fully Managed Integration Service
* Exchange data between applications (connectors) using flows
* Flow consists of Source and Destination connectors
* Sync data across applications
* Aggregate data from different sources
* Public Endpoints, but works with PrivateLink (Privacy)
* AppFlow Custom Connector SDK (build your own)
* Use cases –
  + Contact records from Salesforce => Redshift
  + Support Tickets from Zendesk => S3
* Connections store configuration & credentials to access applications
* Connections can be reused across many flows – they are defined separately
* Configure Source to Destination field Mapping
  + Optional – Data Transformation
  + Optional – Configure Filters and Validation
* Frequency –
  + Schedule
  + In response to a business event
  + On-demand
* Automatically encrypts data in flight, and allows users to restrict data from flowing over the public internet for SaaS applications that are integrated with AWS PrivateLink
  + Reduces exposure to security threats