Section 3: AWS Overview

- Global services:
    - IAM, Route 53 (DNS service)
    - CloudFront (Content Delivery Network)
    - WAF

Section 4: IAM & AWS CLI

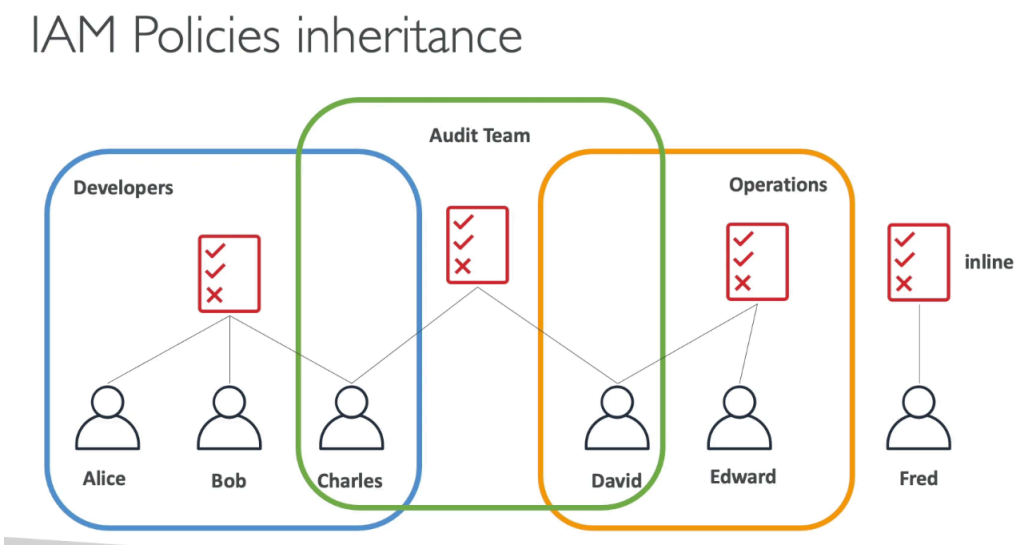
IAM

- Root account created by default
- Users are people within the org and can be grouped
- Groups only contain users, not other groups
- Users don't have to be in a group and users can be in multiple groups

Permissions

- Policy document (JSON) defines what users or groups can do
    - Least privilege principle

Policies



Password Policy

- Strong passwords = high security
- Allow IAM users to change passwords
- Require users to change password after some time

- Prevent password reuse

MFA
- Protect root and IAM users
- MFA = password + security device

Access AWS

- Console, CLI, SDK
    - CLI is a tool that allows AWS interaction in terminal
        - CloudShell is the in console terminal
    - Direct access to public APIs of AWS services
    - SDK is language specific APIs that embed within application to access and manage AWS services
- Access keys generated via Console

IAM Roles

- Some AWS services need to perform actions on your behalf
    - IAM entity that defines permissions to make requests to AWS services and will be used by AWS service
- Assign permissions to AWS services with IAM roles
- Common Roles:
    - EC2 instance roles
    - Lambda function roles

IAM Security Tools

- IAM Credentials report (account level)
    - Report that lists all account users and status of credentials
- IAM Access advisor (user level)
    - Shows the service permissions granted to a user and when those services were last accessed

IAM Guidelines & Best Practices

- Don’t use root account
- One physical user = one AWS user
- Assign users to groups and assign permissions to groups
- Strong password and MFA
- Create and use roles for giving permissions to AWS services

Shared Responsibility Model for IAM

- You: Users, groups, roles, policies management and monitoring

Summary

IAM Section – Summary

- **Users:** mapped to a physical user, has a password for AWS Console
- **Groups:** contains users only
- **Policies:** JSON document that outlines permissions for users or groups
- **Roles:** for EC2 instances or AWS services
- **Security:** MFA + Password Policy
- **AWS CLI:** manage your AWS services using the command-line
- **AWS SDK:** manage your AWS services using a programming language
- **Access Keys:** access AWS using the CLI or SDK
- **Audit:** IAM Credential Reports & IAM Access Advisor

Section 5: EC2 Fundamentals

EC2 Basics

- Elastic Compute Cloud = infrastructure as a service
    - Rent VMs (EC2)
    - Storing data on virtual drives (EBS)
    - Distributing load across machines (ELB)
    - Scaling services using auto scaling group (ASG)

EC2 sizing & config options

- OS, CPU, RAM, Storage, network, WAF, bootstrap script
    - Bootstrapping means launching commands when machine starts (run only once at instance first start)
    - Runs as root user (sudo)

EC2 Instance Types

- Naming convention: m5.2xlarge
    - M: instance class
    - 5: generation
    - 2xlarge: size within the instance class

General Purpose

- Great all around workload: balance between compute, memory, networking

Compute Optimized

- Great for compute intensive tasks that require high performance processors

Memory Optimized

- Fast performance for workloads that process large data sets in memory

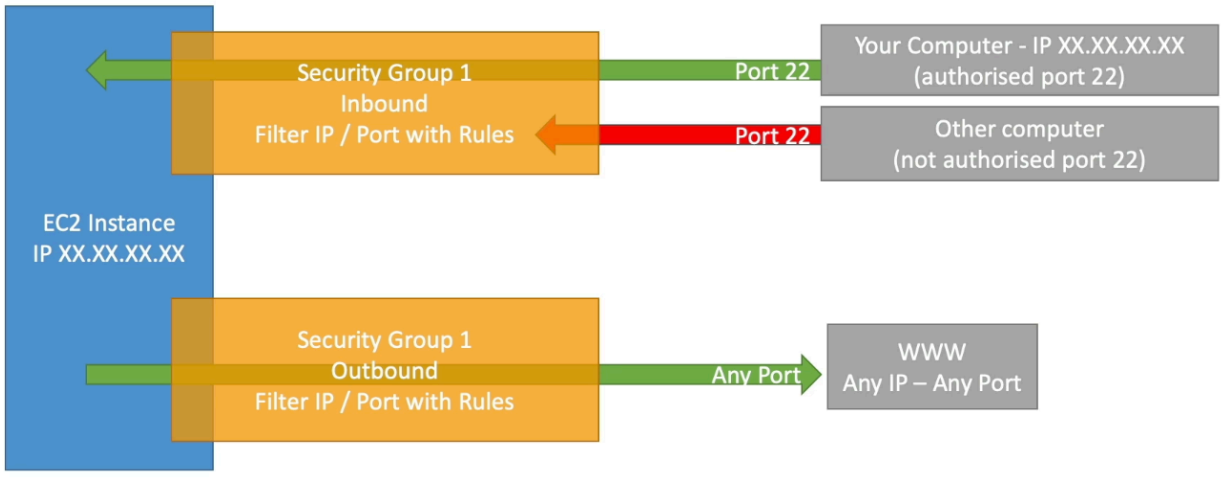
Storage Optimized

- Storage intensive tasks that require high sequential read and write access to large data sets in local storage

Introduction to Security Groups

- Control how traffic is allowed in/out of EC2 instances
- SG only contain ALLOW rules
- SG rules can reference by IP or SG
- SG act as firewall on EC2 instances
    - Regulate:
        - Access to ports
        - Authorized IP ranges
        - Control inbound and outbound network traffic

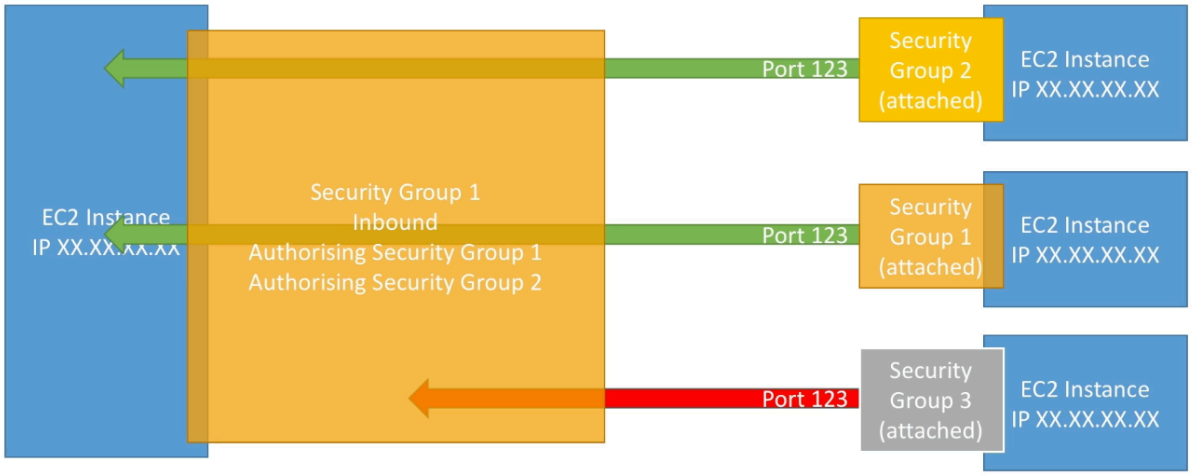
Security Groups Diagram



SG Good to know

- Can be attached to multiple instances
- Locked down to region/VPC combination
- Lives “outside EC2
- Good to maintain one separate SG for SSH access
- If application not accessible (timeout), then it’s security group issue
- Connection refused error, then it’s application error or not launched
- All inbound traffic is BLOCKED by default
- All outbound traffic is authorized by default

Referencing other security groups Diagram



Classic Ports to Know

- 22 == SSH - log into Linux instance
- 21 = FTP (file transfer protocol) - upload files to file share
- 22 = SFTP (secure file transfer protocol) - upload files using SSH
- 80 = HTTP - access unsecure websites
- 443 = HTTPS - access secure websites
- 3389 = RDP (remote desktop protocol) - log into windows instance

EC2 Instance Purchase Options

- On demand instances
    - Short workload, predictable pricing, pay by second (Linux or windows), after first minute
    - All other OS - billing per hour

    - Highest cost, no upfront payment, no long-term commitment
    - Recommended for short-term and un-interrupted workloads where you can't predict how the application will behave

- Reserved (1 & 3 years): long workloads
    - Convertible reserved instances - long workloads with flexible instances
        - Can change instance type, family, OS, scope and tenancy
    - Up to 72% discount compared to on demand
    - Reserve specific instance attributes (instance type, region, tenancy, OS)
    - Payment options: no upfront, partial, all upfront
    - Reserved instance's scope - regional or zonal (reserve capacity in AZ)
    - Recommended for steady state usage applications
    - Buy or sell in reserved instance marketplace

- Savings Plans (1 & 3 years) - commitment to an amount of usage, long workload
    - Commit to certain type of usage
    - Usage beyond EC2 savings plan billed at on demand price
    - Locked to specific instance family and region
    - Flexible across instance size, OS, tenancy

- Spot instances - short workloads, cheap, can lose instances if max price less than current spot price (less reliable)
    - Most cost efficient
    - Workloads that are resilient to failure
        - NOT DB OR CRITICAL JOBS

- Dedicated hosts - book entire physical server, control instance placement
    - Compliance requirements or licenses
    - Purchasing by on demand or reserved
        - Most expensive

- Dedicated instances - no other customers will share hardware
    - May share hardware with other instances in same account
    - No control over instance placement (can move after stop/start)
    - You get your own instance on your own hardware

- Capacity reservations - reserve capacity in specific AZ for any duration
    - No time commitment or discounts
    - Charged at on demand rate whether you run instances
    - For short-term, uninterrupted workloads in specific AZ

Which purchasing option is right for me?

- **On demand:** coming and staying in resort whenever we like, we pay the full price
- **Reserved:** like planning ahead and if we plan to stay for a long time, we may get a good discount.
- **Savings Plans:** pay a certain amount per hour for certain period and stay in any room type (e.g., King, Suite, Sea View, …)
- **Spot instances:** the hotel allows people to bid for the empty rooms and the highest bidder keeps the rooms. You can get kicked out at any time
- **Dedicated Hosts:** We book an entire building of the resort
- **Capacity Reservations:** you book a room for a period with full price even you don't stay in it

EBS Volume

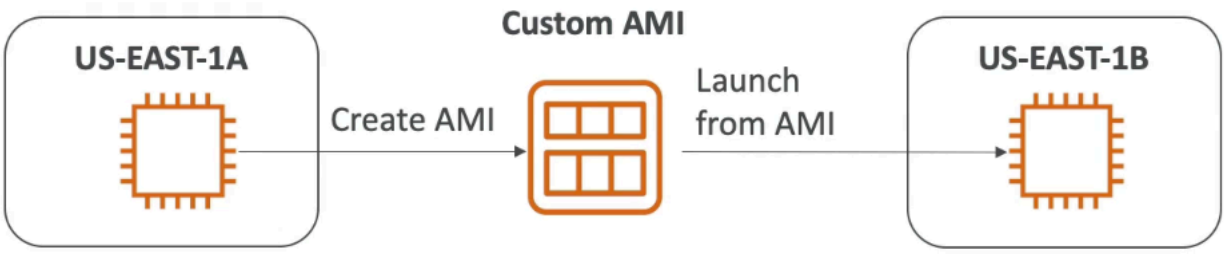
- Network drive attached to instances while they run
    - Allows instances to persist data, after termination
    - Only mounted to one instance at a time and bound to specific availability zones
        - Detached from instance to another quickly (within same AZ)
            - To move volume across, a snapshot is needed
    - "Network USB sticks"
        - Via network, thus latency
    - Has provisioned capacity (can increase, bill increases)
- Delete on termination by default happens to root EBS volume, but others not deleted

Snapshots

- Backup of EBS volume at a point in time
- Not necessary to detach volume to do snapshot
- Can copy across AZ or region
- Snapshot archive
    - Cheaper and takes 24-72 hours to restore from archive
- Recycle bin for EBS snapshots
    - Setup rules to retain deleted snapshots to recover after accidental deletion (retention of 1 day to 1 year)
- Fast Snapshot Restore (FSR)
    - Force full initialization of snapshot to have no latency on first use ($$$)

Amazon Machine Image (AMI) Overview

- Customization of EC2 instance
    - Add own software, configuration for faster boot/config
    - Start instance → customize → Stop instance → Build AMI → launch
- Built region specific
    - Launch from public AMI (AWS provided), personal, marketplace AMI



EC2 Instance Store

- High performance hardware disk
    - Better I/O performance
    - EC2 instance store lose storage if EC2 instance is stopped
        - Good for cache, temporary content due to risk of data loss

EBS Volume Types

- 6 types
    - gp2/gp3 (SSD)
        - General purpose SSD volume that balances price and performance
        - GP3 can independently increase IOPS, while GP2 GP and IOPS linked

EBS Volume Types Use cases

General Purpose SSD

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

    - IO1/IO2 block express (SSD)
        - High performance SSD for critical, low latency, high throughput workloads
            - For over 32k IOPS, need EC2 Nitro with IO1 or 2
        - EBS Multi Attach only available here
    - STL (HDD)
        - Low Cost HDD volume for frequently accessed throughput intensive workloads
    - SCL (HDD)
        - Lowest cost HDD volume for less frequently accessed workloads
- Only gp2/gp3 and IO1/IO2 block express can be used as boot volumes (where root OS is running)

Provisioned IOPS SSD

- Critical apps with sustained IOPS performance
    - DB workloads (sensitive to storage performance and consistency)

EBS Volume Types Use cases

Provisioned IOPS (PIOPS) SSD

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

EBS Volume Types Use cases

Hard Disk Drives (HDD)

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

EBS – Volume Types Summary

	General Purpose SSD volumes		Provisioned IOPS SSD volumes	
Volume type	gp3	gp2	io2 Block Express [3]	io1
Durability	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)		99.999% durability (0.001% annual failure rate)	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)
Use cases	• Transactional workloads • Virtual desktops • Medium-sized, single-instance databases • Low-latency interactive applications • Boot volumes • Development and test environments		Workloads that require: • Sub-millisecond latency • Sustained IOPS performance • More than 64,000 IOPS or 1,000 MiB/s of throughput	• Workloads that require sustained IOPS performance or more than 16,000 IOPS • I/O-intensive database workloads
Volume size	1 GiB - 16 TiB		4 GiB - 64 TiB [4]	4 GiB - 16 TiB
Max IOPS per volume (16 KiB I/O)	16,000		256,000 [5]	64,000
Max throughput per volume	1,000 MiB/s	250 MiB/s [1]	4,000 MiB/s	1,000 MiB/s [2]
Amazon EBS Multi-attach	Not supported		Supported	
NVMe reservations	Not supported		Supported	Not supported
Boot volume	Supported			

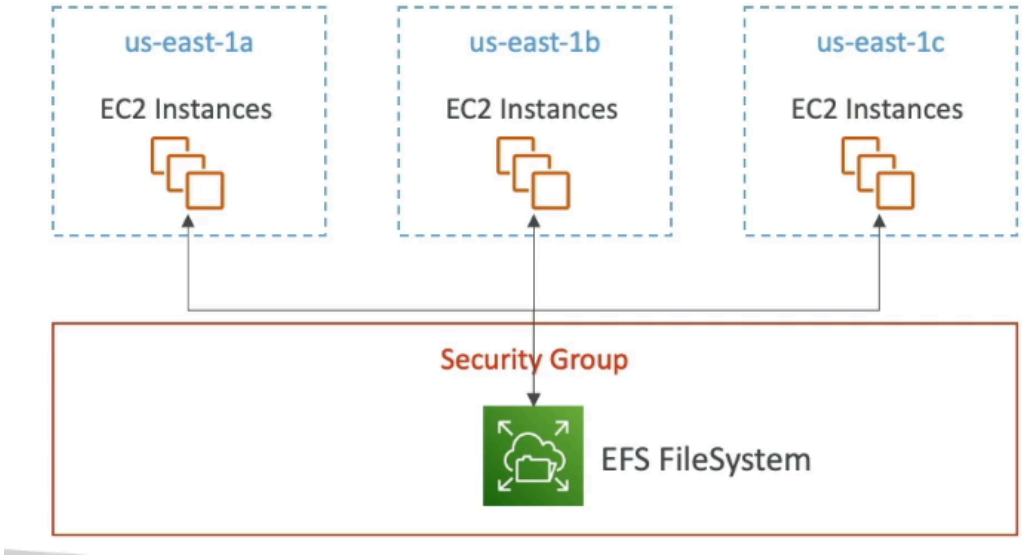
	Throughput Optimized HDD volumes	Cold HDD volumes
Volume type	st1	sc1
Durability	99.8% - 99.9% durability (0.1% - 0.2% annual failure rate)	
Use cases	• Big data • Data warehouses • Log processing	• Throughput-oriented storage for data that is infrequently accessed • Scenarios where the lowest storage cost is important
Volume size	125 GiB - 16 TiB	
Max IOPS per volume (1 MiB I/O)	500	250
Max throughput per volume	500 MiB/s	250 MiB/s
Amazon EBS Multi-attach	Not supported	
Boot volume	Not supported	

EBS Multi Attach

- Attach same EBS volume to multiple EC2 instances in same AZ
- Each instance has full read/write permissions
- Up to 16 EC2 Instances at a time
- Must use a file system that's cluster aware

Amazon EFS (Elastic File System)

- Managed network file system mounted on EC2
- Works with EC2 instances in multi AZ, highly available, scalable
- Use cases: content management, web serving, data sharing
- NFSv4.1 protocol
- Uses SG to control access to EFS
- Only compatible with Linux based AMI
    - POSIX file system
    - Scales automatically, pay per use, no capacity planning
- Encryption at rest using KMS
- Performance mode and throughput mode settings

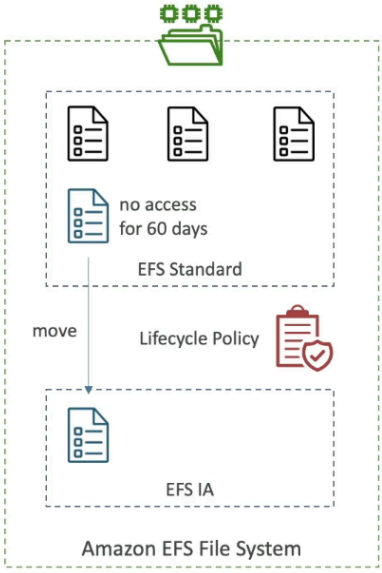


EFS – Performance & Storage Classes

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

EFS – Storage Classes

- Storage Tiers (lifecycle management feature – move file after N days)
  - Standard: for frequently accessed files
  - Infrequent access (EFS-IA): cost to retrieve files, lower price to store. Enable EFS-IA with a Lifecycle Policy
- Availability and durability
  - Standard: Multi-AZ, great for prod
  - One Zone: One AZ, great for dev, backup enabled by default, compatible with IA (EFS One Zone-IA)
- Over 90% in cost savings



EFS vs EBS

- EBS volumes
  - One instance (except multi attach IO 1/IO 2)
  - Locked at AZ level

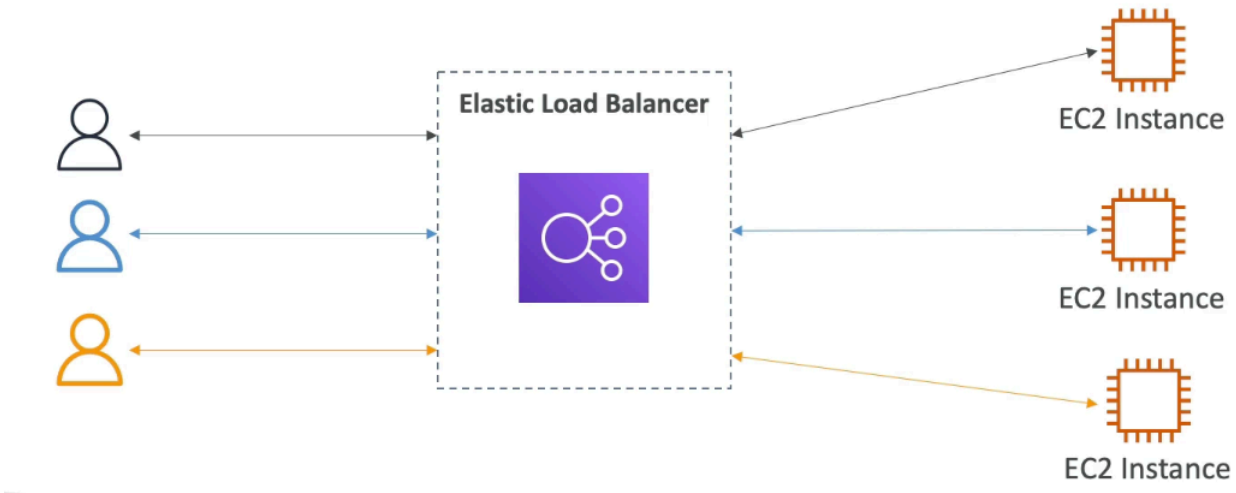
        - GP2: IO increases if disk size increases
        - GP3 & IO1: increase IO independently
        - Migrate EBS across AZ by taking snapshot and restore at another AZ
            - Backups use IO, so should be done when low traffic
        - Root EBS volumes terminated by default if EC2 instance is terminated
- EFS
    - Mounts to many instances across AZ
    - Only for linux
    - Higher price than EBS

Section 7: AWS Fundamentals: ELB + ASG

Scalability & High Availability

- Scalability means an application can handle greater loads by adapting
    - Vertical and horizontal (elasticity)
        - Vertical means upgrading size of instance (like DB)
            - Hardware limit
        - Horizontal means more instances
            - Implies distributed systems like web apps
    - Linked, but different to availability
- High Availability
    - Running app in at least 2 AZ (prevent data center loss)
    - Active or passive

Elastic Load Balancing (ELB) Overview



- Forward traffic to multiple server downstream
    - Helps spread load across multiple downstream instances
- Expose single point of access (DNS) to app

        - Seamlessly handle failures of downstream instances
        - Health checks on instances
            - Verifies if EC2 instance works and can forward traffic
            - Done on port an route (/health)
        - SSL termination (HTTPS)
        - Enforce stickiness with cookies
        - High availability
        - Separate public traffic from private traffic
- Managed LB meaning AWS upgrades, availability, etc

3 types
- Application load balancer: HTTP, HTTPS, websocket
- Network LB: TCP, TLS (secure TCP), UDP
- Gateway LB: Operates at layer 3 network layer IP protocol

Security Groups

Load Balancer Security Group:

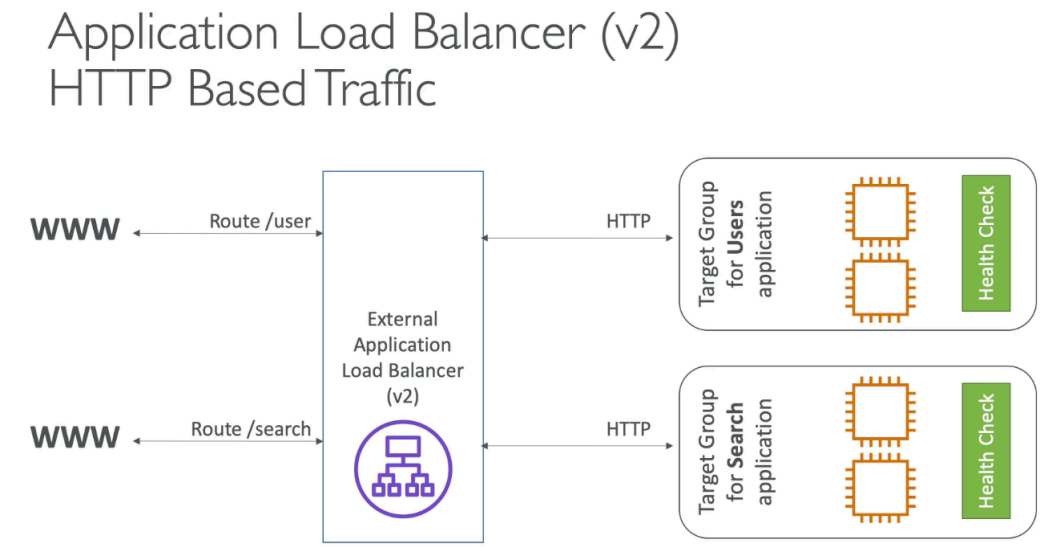
Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	Allow HTTP from an...
HTTPS	TCP	443	0.0.0.0/0	Allow HTTPS from a...

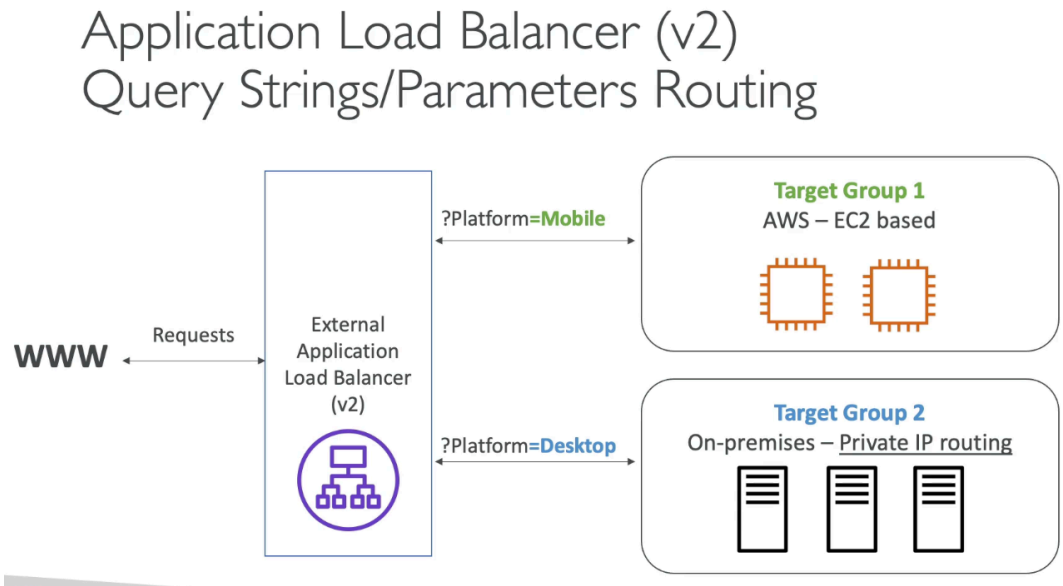
Application Security Group: Allow traffic only from Load Balancer

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	sg-054b5ff5ea02f2b6e (load-b	Allow Traffic only...

Application Load Balancer

- Layer 7 HTTP (supports HTTP/2 and websocket, and redirects from HTTP to HTTPS)
- Load balance to multiple HTTP apps across same or different machines
- Routing tables to different target groups
    - Based on path, hostname, query string or headers
- Good for container based apps
- Has port mapping to redirect to a dynamic port in ECS

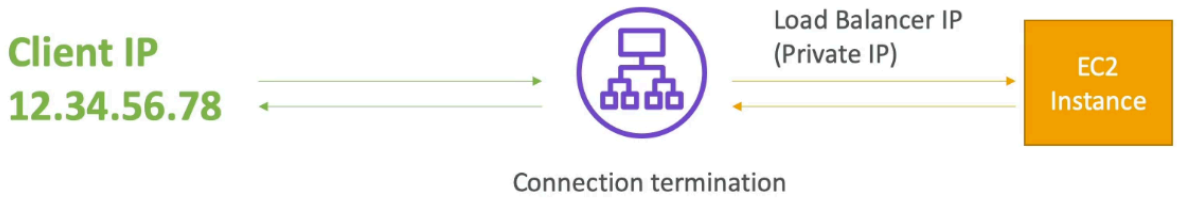




Target Groups

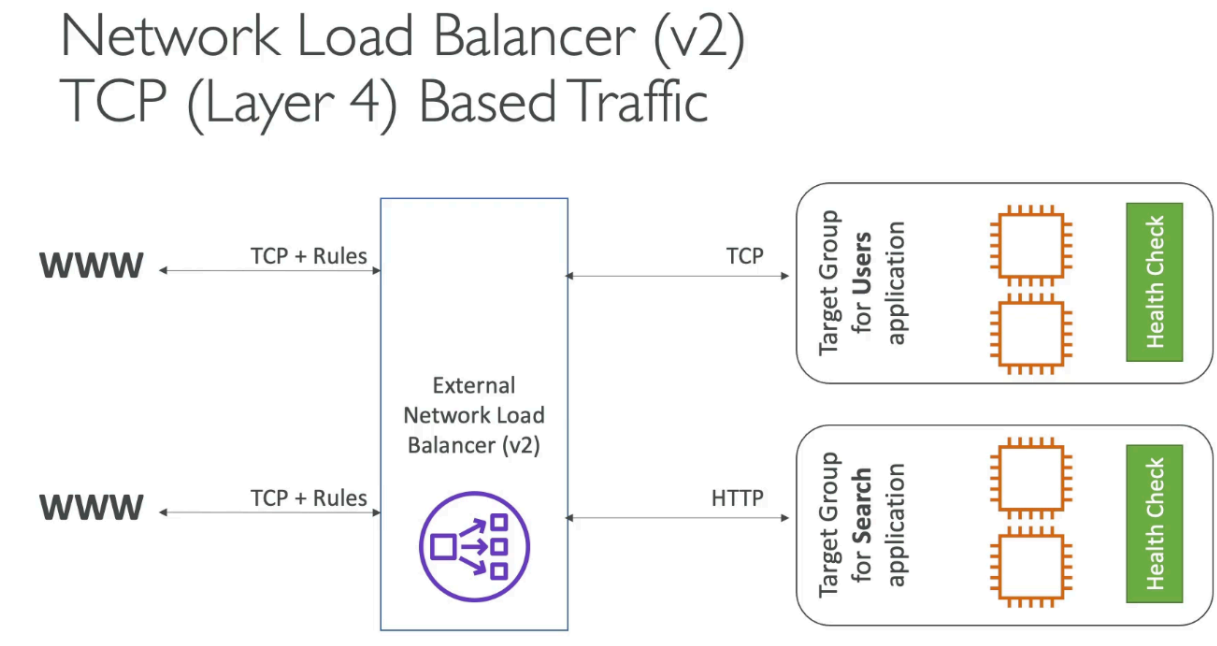
- EC2 instances (can be managed by auto scaling groups) - HTTP
- ECS tasks - HTTP
- Lambda functions - HTTP request translated into JSON event
- IP address - must be private
- ALB can route to multiple target groups and health checks done at target group level

Good to know



- Fixed hostname
- Application servers don't see the IP of client directly
    - True IP of client is inserted in header X-Forwarded-For
    - Can also get port, X-Forwarded-Port and proto (X-Forwarded-Proto)

Network Load Balancer



- Layer 4 allows to forward TCP/UDP traffic instances
    - High performance to handle millions of requests and 100 ms vs 400 ms for ALB
- One static IP per AZ and supports assigning elastic IP to each AZ
    - Expose app to a set of IPs, whitelisting specific IP

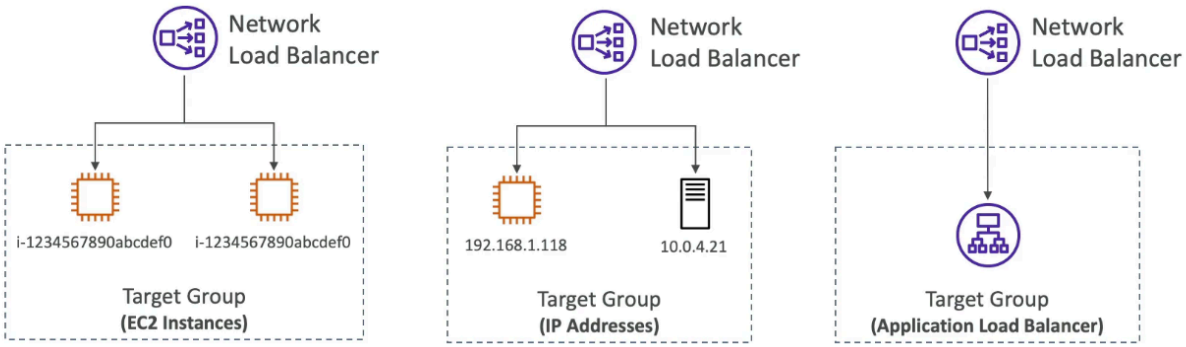
Target Groups

- EC2 instances
- IP addresses - must be private IP
- Application Load balancer

- Why do this? NLB gets fixed IP addresses and ALB gets all the rules around handling HTTP traffic
- Health checks by NLB support TCP, HTTP, and HTTPS protocols

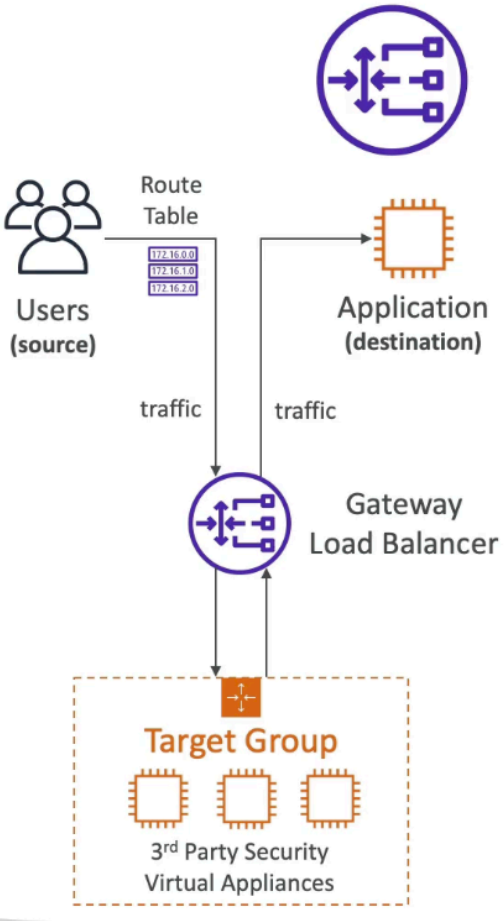
Network Load Balancer – Target Groups

- EC2 instances
- IP Addresses – must be private IPs
- Application Load Balancer



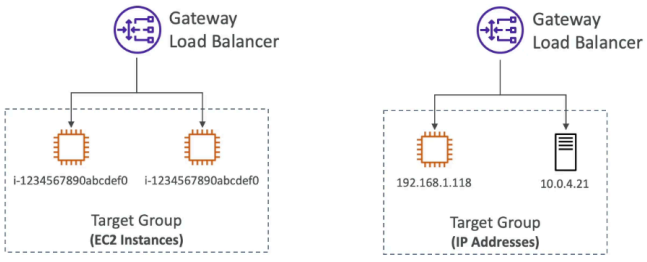
Gateway Load Balancer

- Deploy and manage 3rd party network apps in AWS
  - Firewalls, intrusion detection, etc…
- Operates at layer 3 (network layer) - IP Packets
- Combines:
  - Transparent network gateway - single entry/exit for all traffic
  - Load balancer - distributes traffic to all virtual apps
- Uses GENEVE protocol on port 6081

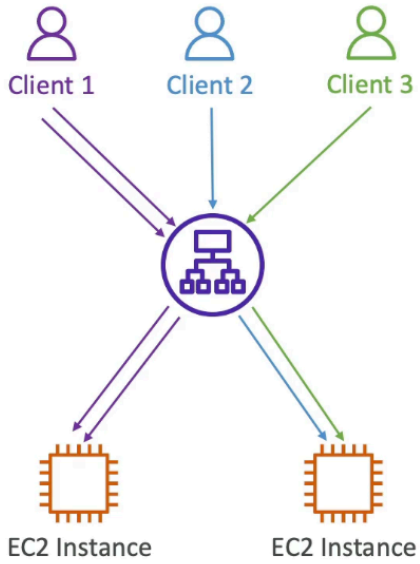


Gateway Load Balancer – Target Groups

- EC2 instances
- IP Addresses – must be private IPs



Elastic Load Balancer - Sticky Session (Session Affinity)



- Same client is always redirected to the same instance behind a LB
    - May bring imbalance to load over backend instances
    - Done at target group level
- A “cookie” used for stickiness and has an expiration date you control
- Use case: make sure the user doesn’t lose session data

Sticky Sessions - Cookie Names

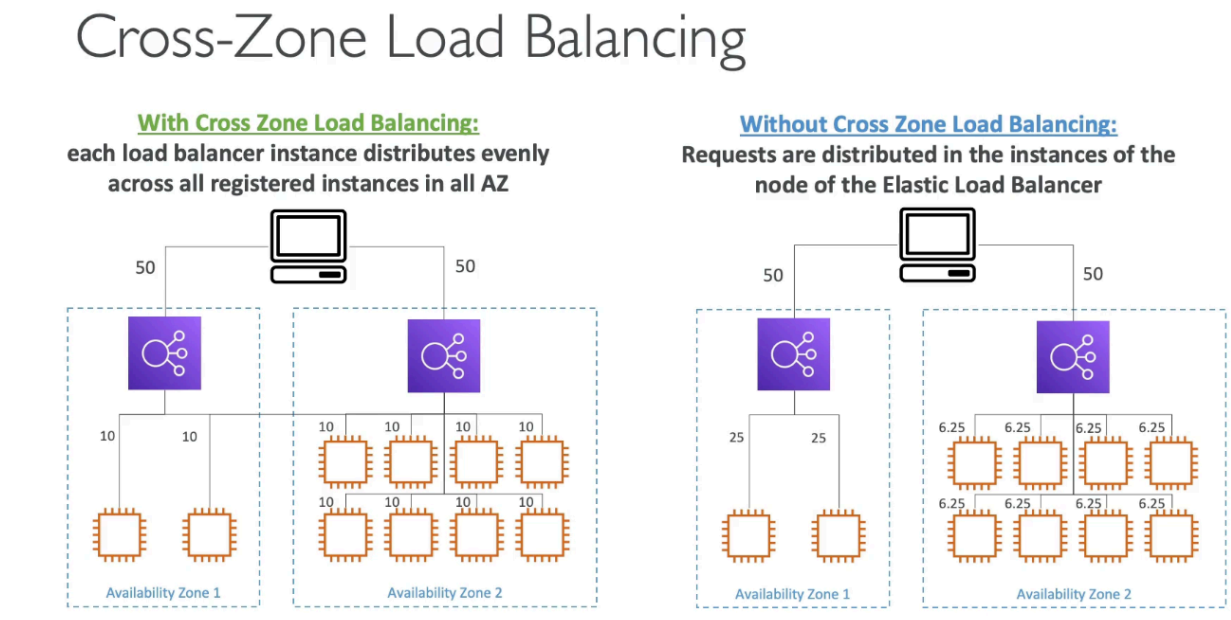
Application based cookies;

- Custom cookie
    - generated by target
    - Include any custom attributes required by app
    - Cookie name must be specified individually for each TG
- Application Cookie
    - Generated by LB, cookie name is AWSALBAPP

Duration based cookies

- Generated by LB, name is AWSALB for ALB, AWSELB for CLB

ELB - Cross Zone LB



- Each LB instance distributes evenly across all instances in all AZs
    - Client distributes evenly to ALBs, but each ALBs will evenly distribute regardless of AZ
- ALB
    - Enabled by default (disabled at TG level)
    - No charges for inter AZ data
- NLB & Gateway LB
    - Disabled by default, pay to use

ELB - SSL Certificates

SSL/TLS Basics

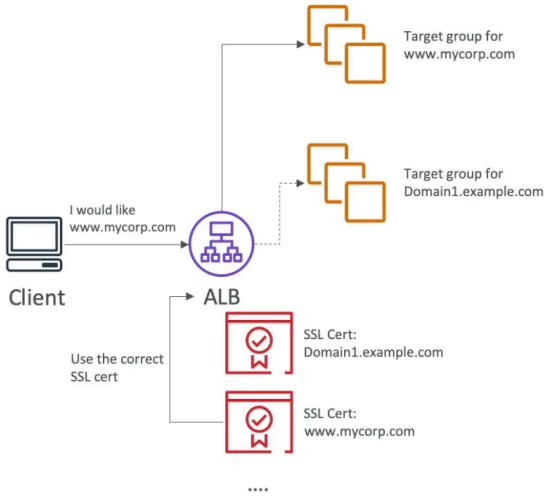
- SSL Cert allows traffic between client and LB to be encrypted in transit (in flight encryption)
    - SSL = secure sockets layer, used to encrypt connections
    - TLS = transport layer security, newer version, mainly used
- Public SSL certs issued by certificate authorities
- SSL Certs have expiration date (you set) and must be renewed

LB SSL Certificates

- Load balancer uses X.509 certificate and managed via ACM
- HTTPS listener:
    - Must specify default certificate
    - Clients can use SNI (server name indication) to specify hostname reached
    - Ability to specify security policy to support older versions
    -

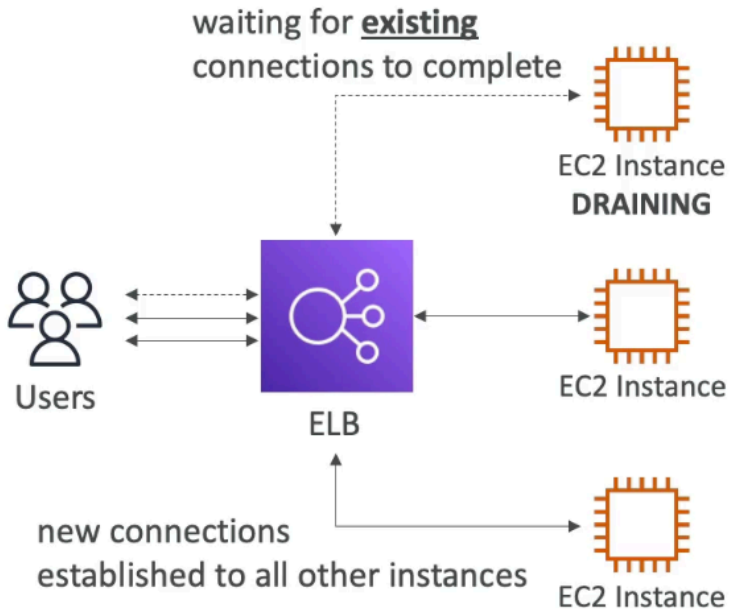
SSL - Server Name Indication (SNI)

- Solves problem of loading multiple SSL certificates to one web server (to serve multiple websites)
    - Newer protocol and requires client to indicate the hostname of the target server in initial SSL handshake
    - Only works for ALB, NLB, CloudFront



ELB - Connection Draining

- ALB/NLB called deregistration delay
- Time to complete “in-flight requests” while the instance is deregistering or unhealthy (default 300 seconds or disabled - only low values if requests are short)
    - Stops sending new requests to EC2 instance that is deregistering



Auto Scaling Groups (ASG)

- Goal to scale in/out instances to match load
    - Min and max EC2 instances and automatically register new instances to LB
    - Recreate EC2 instance in case of unhealthy instances
- Launch template for various attributes
- Can scale based on CloudWatch alarms based on metric that are computed for overall instances

Scaling Policies

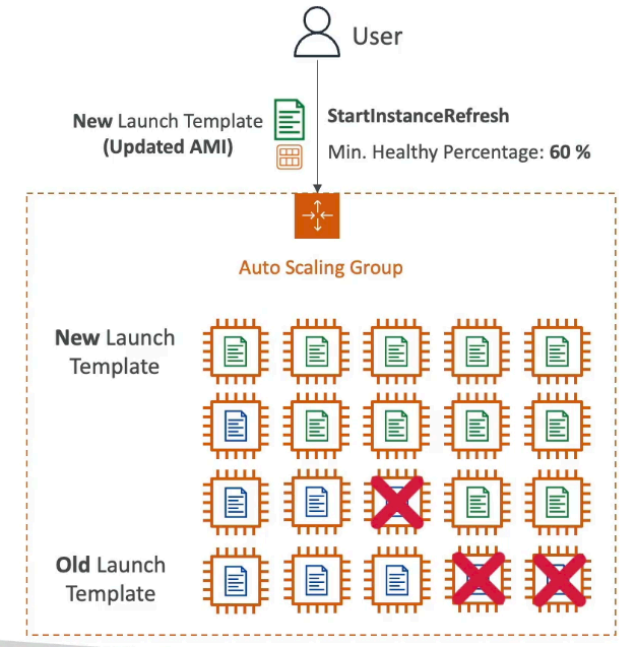
- Dynamic Scaling
    - Target Tracking Scaling
        - Based on metric
    - Simple/Step scaling
        - When CloudWatch alarm is triggered, add or remove
- Scheduled Scaling
    - Based on time and predictability of usage
- Predictive Scaling
    - Continuous forecast load and schedule scaling ahead

Metrics to scale on:

- CPU utilization: avg CPU usage across instances
- RequestCountPerTarget: number of request per instance is stable
- Average network in/out: if app is network bound
- Custom metric via CloudWatch

Scaling Cooldowns

- After scaling occurs, cooldown period of 300 seconds where ASG will not launch or terminate instances to allow metrics to stabilize



Instance Refresh

- Goal: update launch template and recreate all EC2 instances
- Setting of minimum healthy %: tells how many instances can be deleted over time
    - As new instances are created, old ones are terminated
    - Specify warm up time (how long until instance is ready to use)

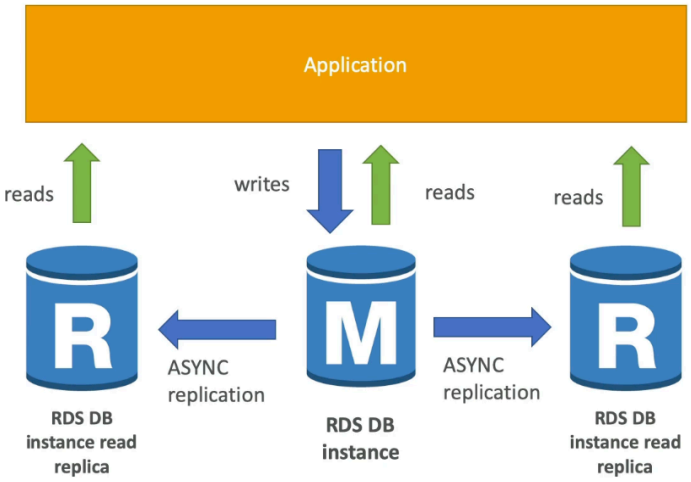
Section 8: AWS Fundamentals: RDS + Aurora + ElastiCache

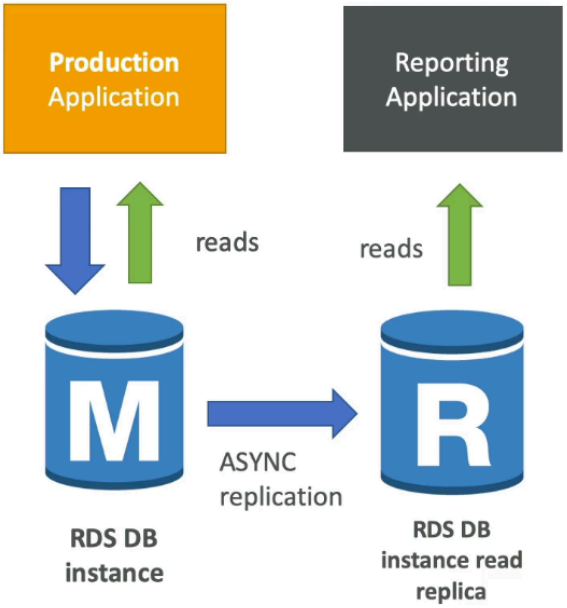
RDS (Relational DB Service) Overview

- Managed DB service using SQL
    - Create DB managed by AWS: PostGres, MySQL, etc…
- RDS vs DB on EC2
    - Managed by AWS for automated provisioning, OS patching, backups/restore, monitoring, read replicas, multi AZ for disaster recovery, scaling capabilities
    - Cannot SSH into RDS instances
- Storage Auto Scaling
    - Dynamically increase storage, done automatically
    - Avoid manual scaling and have a maximum storage threshold
    - Automatically modify if:
        - Free storage < 10% of allocated storage
        - Low storage lasts at least 5 minutes
        - 6 hours passed since last modification
    - Useful for apps with unpredictable workloads

RDS Read Replicas for read scalability

- Up to 15 read replicas within or cross AZ/region
- Async replication between main and read replicas





- Read replicas can be promoted to its own DB, out of replication mechanism

Use Cases:

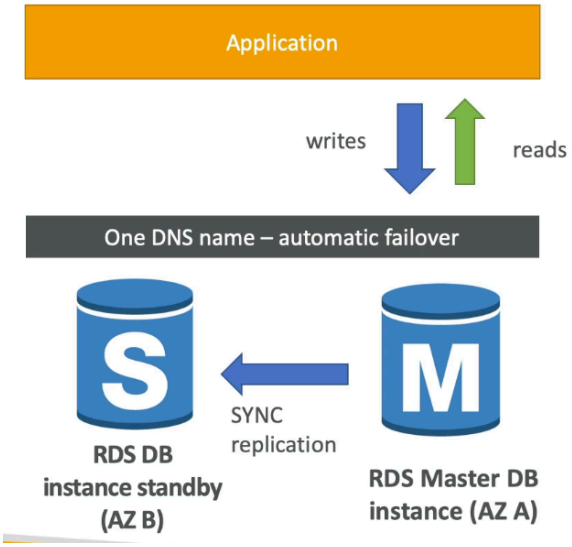
- Production DB on normal load. A new team wants to run analytics on DB, so a read replica is created to run new workloads

Read replicas network costs:

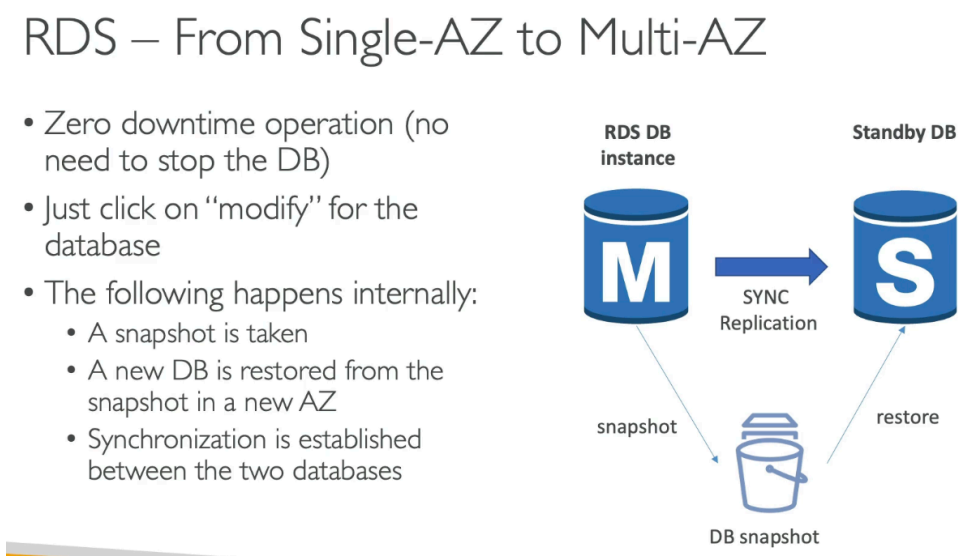
- For RDS replicas within the same region from one AZ to another AZ, no fee to move data. For cross region will incur replication fees

RDS Multi AZ (Disaster recovery)

- Synchronous replication to standby instance in a different AZ
- One DNS name - automatic failover to standby to increase availability
- Not used for scaling



From Single AZ to Multi AZ



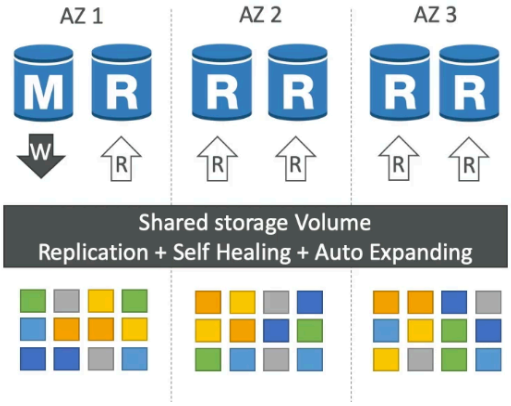
- 0 downtime, just enable multi AZ and it will have a standby DB with synchronous replication
- Internal process:
    1. Snapshot taken
    2. New DB is restored from snapshot in new AZ
    3. Synchronization is established between the 2 DB

Amazon Aurora

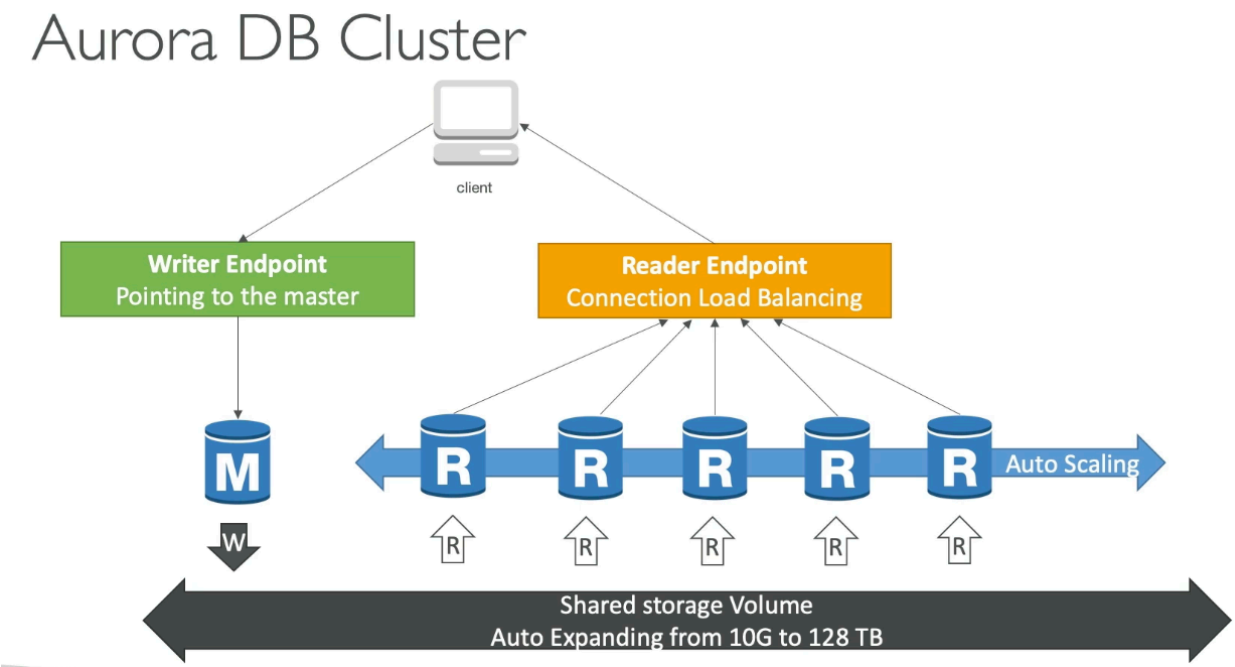
- Proprietary tech from AWS, but compatible w/ Postgres and MySQL
    - Cloud optimized, good performance over base postgres/mysql
- Automatically grows in increments of 10 GB up to 128TB
- 15 read replicas and faster than MySQL, failover is instantaneous, high availability
    - Slightly higher cost than RDS, but more efficient

High Availability and Read Scaling

- 6 copies of data across 3 AZ:
    - 4/6 copies needed for writes, 3/6 needed for reads, self-heal with peer to peer replication
    - Storage striped across 100s of volumes
- One Aurora instance takes writes (master) with automated failover for master < 30 sec
    - Master + up to 15 read replicas in cross origin replication



Aurora DB Cluster



- DNS name called writer endpoint, which points at the master, which will automatically write to the master instance, even if master changes.
- Reader endpoint: connection load balancing and connects automatically to all the read replicas so anytime the client connects to reader endpoint, it will load balance across all read replicas
    - LB happens at connection level

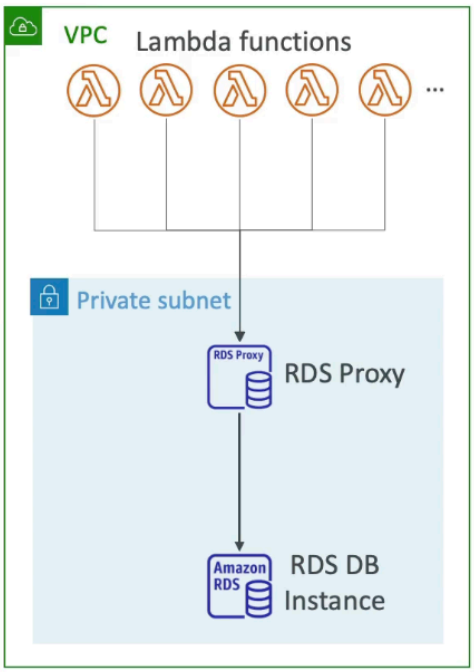
Features of Aurora

- Auto failover
- Backup and recovery
- Isolation and security
- Industry compliance
- Push button scaling
- Advanced monitoring
- Automated patching with 0 downtime
- Routine maintenance
- Backtrack: restore data and any point in time without backups

RDS & Aurora Security

- At rest encryption: master and replica encryption via KMS (defined at launch time)
    - If master not encrypted, read replicas cannot be encrypted
    - To encrypt an un-encrypted DB, go through DB snapshot and restore as encrypted

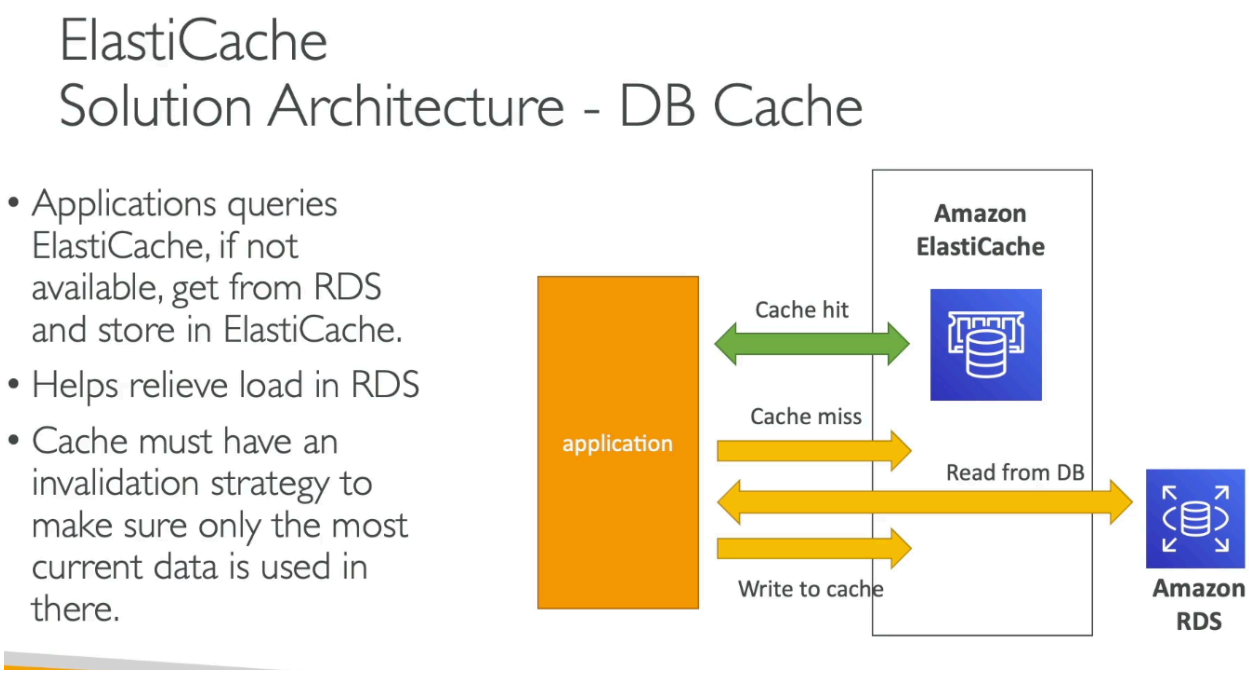
- In flight encryption: TLS ready by default, clients must use AWS TLS root certificates
- IAM Auth: roles to connect to DB instead of user/pass
- SG: control network access
- No SSH available except RDS custom service
- Audit logs can be enabled and sent to CloudWatch (longer retention)



Amazon RDS Proxy

- Fully managed DB proxy for RDS and Aurora
  - RDS can already be reached directly, but proxy allows apps to pool and share DB connections with the DB. Basically, a proxy pools all connections and sends less connections to RDS DB instance at a time
    - Improves efficiency by reducing stress on resources and minimizes open connections
    - Never publicly accessible, must be accessed via VPC
  - Serverless, autoscaling, highly available (multi AZ)
  - Reduce RDS/Aurora failover time
  - Enforces IAM auth for DB and credentials stored via Secrets Manager

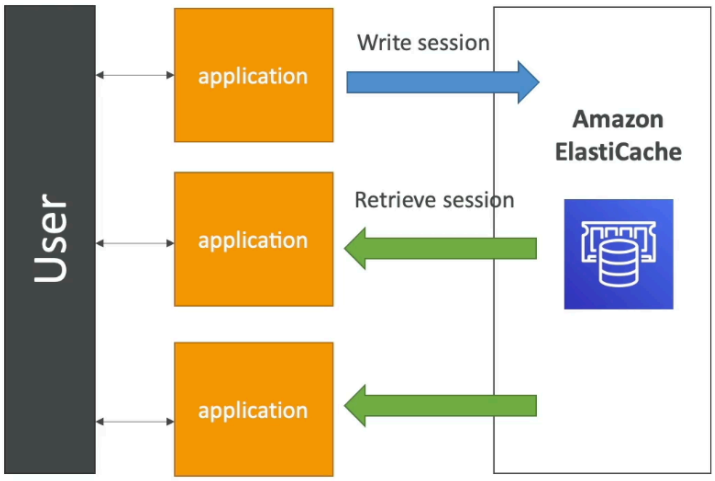
ElastiCache Overview



- AWS managed Redis or mem-cached
- Caches are in-memory DB with high performance, low latency to help reduce load off DB for read intensive workloads and makes application stateless
- Needs cache invalidation method to make sure most current data is cached

ElastiCache
Solution Architecture – User Session Store

- User logs into any of the application
- The application writes the session data into ElastiCache
- The user hits another instance of our application
- The instance retrieves the data and the user is already logged in



ElastiCache - Redis vs Memcached

Redis

- Multi AZ with auto-failover
- Read replica to scale read and high availability
- Data durability using AOF persistence
- Backup and restore features
- Supports sets and sorted sets
- Max 5 read replicas in redis cluster with cluster mode disabled

Memcached

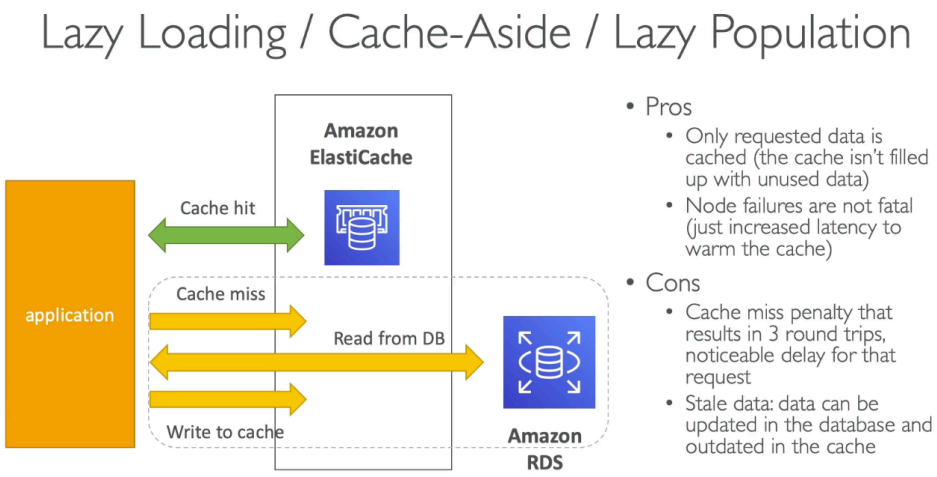
- Multi-node partitioning of data (sharding)
- No high availability (replication)
- Non-persistent
- No backup and restore
- Multithreaded architecture

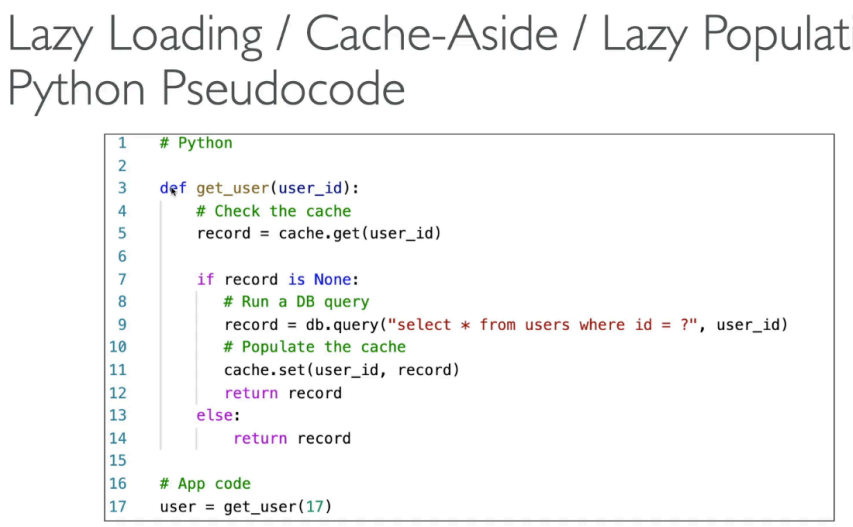
ElastiCache Strategies

Caching Implementations Considerations

- Is it safe to cache? Data may be out of date, eventually consistent
- Is caching effective for data? Pattern: changing slowly, few keys frequently needed. Anti patterns: data changes rapidly, all large key space frequently needed
- Is data structured well for caching? Key value cache, cache aggregation results

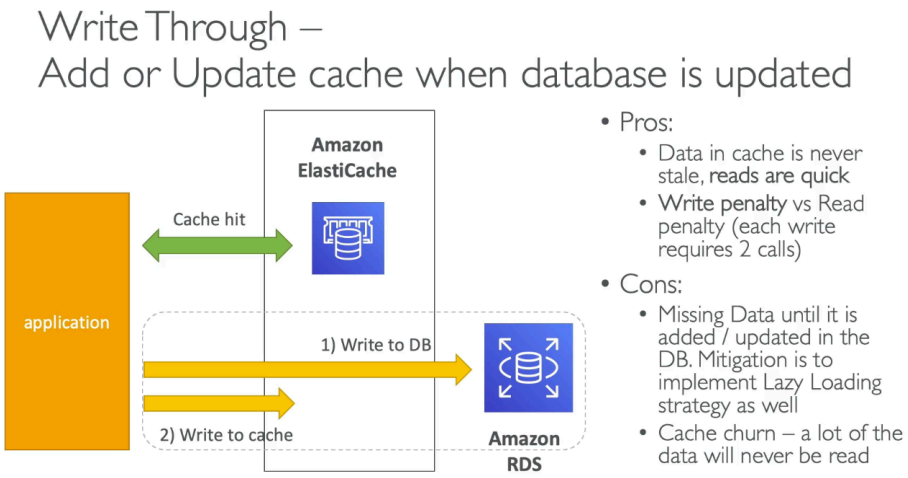
Lazy Loading / Cache-Aside / Lazy Population

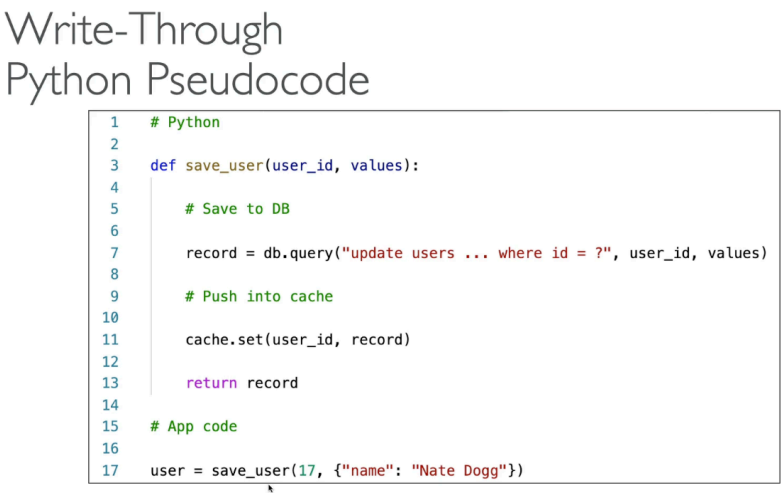




- Pros
  - Only requested data is cached (no unused data cached)
  - Node failures not fatal (only increased latency to warm cache)
- Cons
  - Cache miss penalty that results in 3 round trips (noticeable delay)
  - Stale data: data can be updated in DB and outdated in cache

Write Through - Add/update cache when DB updated





- On cache miss, write to RDS directly, then write to cache
- Pros:
    - Data in cache never stale, reads are quick
    - Write penalty vs read penalty (each write requires 2 calls instead of 3)
        - Better from a user perspective. On social media, fetching profile fast, posting is slow (post = write to DB)
- Cons:
    - Missing data until it is added/updated to DB
        - Implement lazy loading to mitigate
    - Cache churn - a lot of data will never be read (if cache small)

Cache Evictions and Time to live (TTL)

- Cache eviction occurs in 3 ways:
    1. Delete explicitly

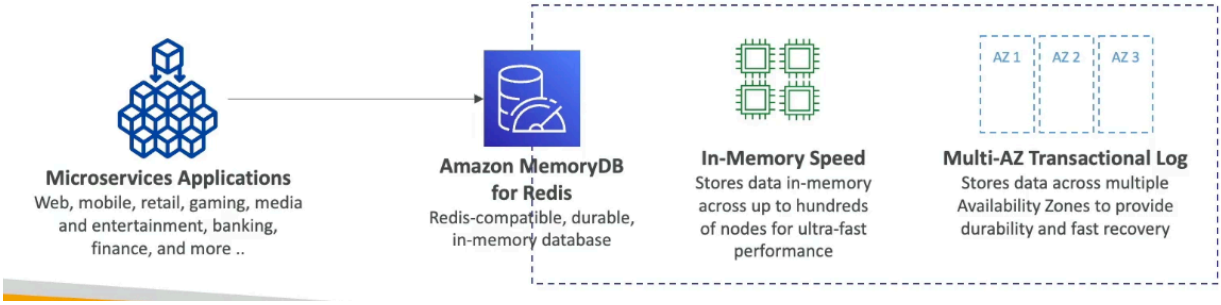
2. Item is removed because memory is full and not recently used (LRU)
3. Set time to live (TTL)

- TTL is helpful for any kind of data and ranges from seconds to days. Balances cache storage, but if evictions are too common due to memory, scale the cache

Final Thoughts

- Lazy loading is easy to implement, helps on read side
- Write through is usually combined with lazy loading as targeted for queries or workloads that benefit such optimization
- Setting TTL good except when using write through
- Only cache data that makes sense

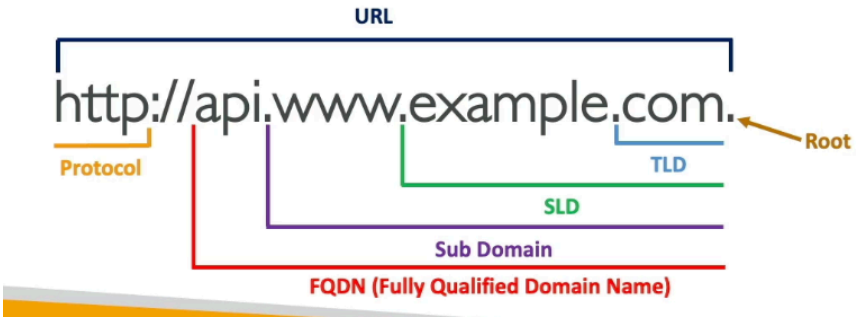
Amazon MemoryDB for Redis



- Redis compatible, durable, in memory DB service
- A memory DB that has redis compatible API
- Very fast performance and durable in memory data storage with multi AZ transactional log
- Scales seamlessly from 10s GB to 100s TB storage

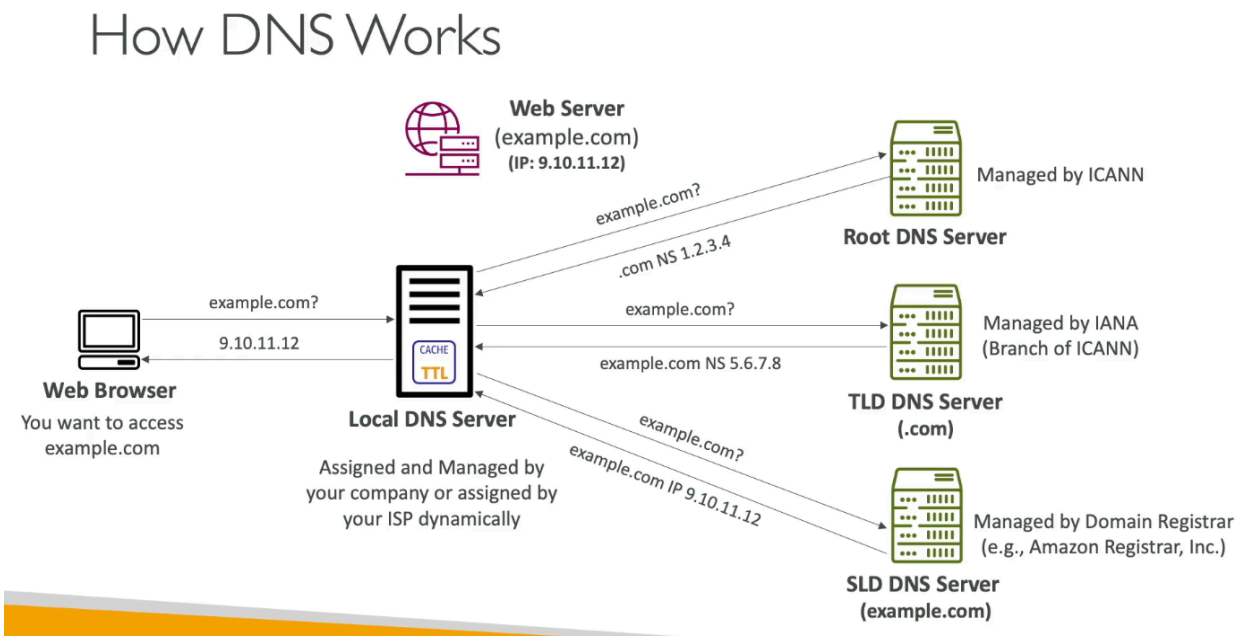
Section 9: Route 53

What is DNS?



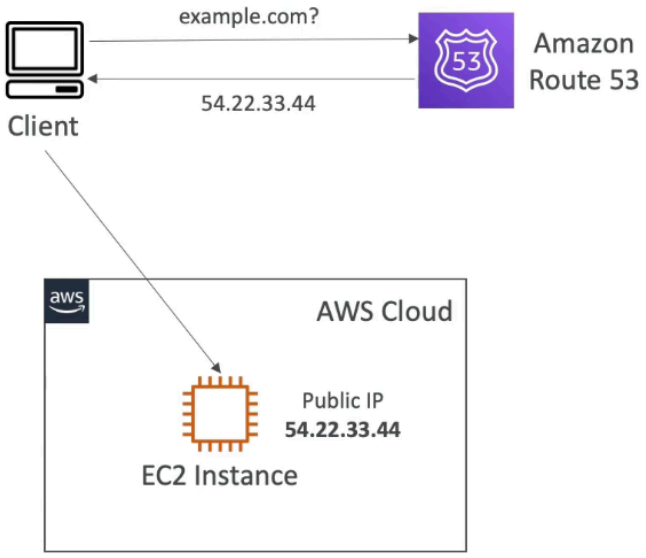
- Domain name system → translates human friendly hostnames into machine IP addresses (e.g. google.com → 123.456.78.90
    - Backbone of internet and has hierarchical naming structure: .com, x.com, www.x.com
- Domain registrar: register domain name
- DNS records: A, AAAA, CNAME, NS
- Zone file: contains DNS records → match hostname to IP
- Name server: resolves DNS queries (authoritative or non authoritative)
- Top level domain (TLD): .com, .org, etc..
- Second level domain (SLD): google.com

How DNS works



Route 53 Overview

- Highly available, scalable, fully managed and authoritative DNS
    - Authoritative = customer can update DNS records
- Also a domain registrar (can register domain)
- Ability to check health of resources
- Only service with 100% availability SLA



Route 53 Records

- How you want to route traffic for a domain
- Record contains:
    - Domain/subdomain name - example.com
    - Record type - A, AAAA
    - Value - 123.456.78
    - Routing policy - how Route 53 responds to queries
    - TTL - amount of time record cached at DNS resolvers
- Supports the following DNS record types:
    - Must know: A / AAAA / CNAME / NS

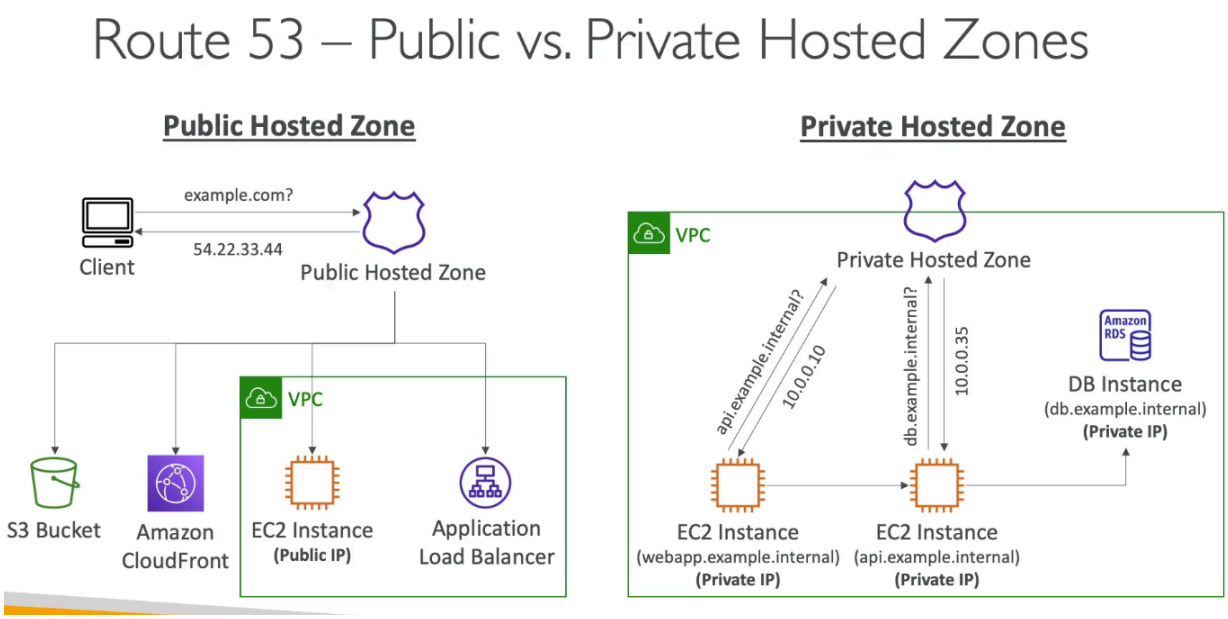
Record Types:

- A: maps a hostname to IPv4 (example.com → 123.456.78)
- AAAA: maps hostname to IPv6
- CNAME - map hostname to another hostname
    - Target is a domain name which must have A or AAAA record
    - Can't create a CNAME record for top node of DNS namespace (zone apex)
        - Ex: can't create for example.com, but can for www.example.com
- NS: name servers for hosted zone
    - This is DNS names or IP addresses of servers that can respond to DNS queries for hosted zone
    - Control how traffic is routed to domain

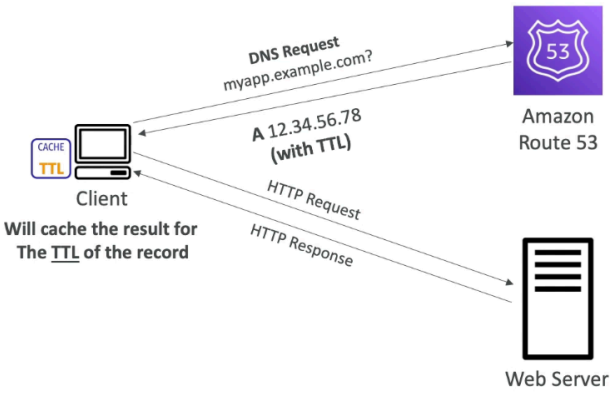
Hosted Zones

- A container for records that define how to route traffic to a domain and subdomains
- Public hosted zones: contains records that specify how to route traffic on internet (public domain names)
- Private hosted zones: records that specify how you route traffic within 1+ VPCs (private domain name)
- $0.50/month per hosted zone

Public vs Private Hosted Zones



Records TTL (time to live)

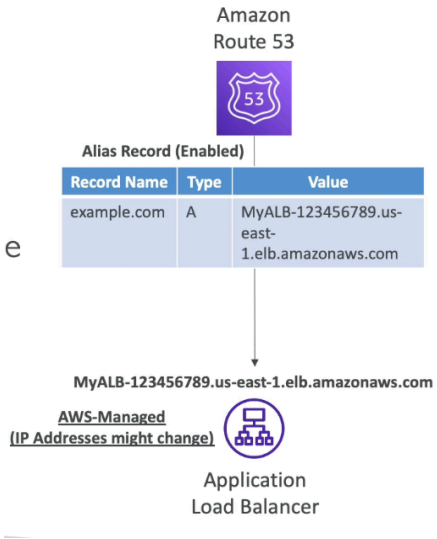


- If the client requests the same host name again, the client will not query the DNS system if it is cached and still within TTL. Done because we don't want to query the DNS often
  - High TTL:
    - Less traffic on Route 53
    - Possibly outdated records
  - Low TTL:
    - More traffic on Route 53 ($$$)
    - Records are outdated for less time
    - Easy to change records
- Except for Alias records, TTL is mandatory for each DNS record

CNAME vs Alias

- AWS resources exposes AWS hostname (map hostname to domain name)
- CNAME:
  - Points a hostname to any other hostname (example.com → example1.com)
  - Only works for non-root domain (aka something.domain.com, not domain.com)
- Alias:
  - Points a hostname to AWS resource
  - Works for root and non root domains

- Free of charge and have native health check
- Extension to DNS functionality
- Automatically recognizes changes in resource's IP address
- Unlike CNAME, it can be used for top node of DNS namespace (zone apex) e.g: example.com
- Always of type A / AAAA
- Can't set TTL (done automatically)
- Cannot set ALIAS record for EC2 DNS name



Route 53 – Alias Records Targets

- Elastic Load Balancers
- CloudFront Distributions
- API Gateway
- Elastic Beanstalk environments
- S3 Websites
- VPC Interface Endpoints
- Global Accelerator accelerator
- Route 53 record in the same hosted zone

- You cannot set an ALIAS record for an EC2 DNS name

Routing Policies

- Define how Route 53 responds to DNS queries
  - DNS does not route any traffic, only responds to DNS queries (translate hostnames to IP)

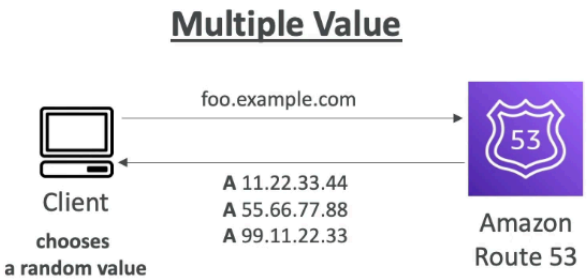
Simple

- Typically route traffic to a single resource
- Can specify multiple values in same record

- If multiple values are returned, a random one is chosen by the client
- When Alias enabled, specify only 1 AWS resource
- Can't associate with health checks

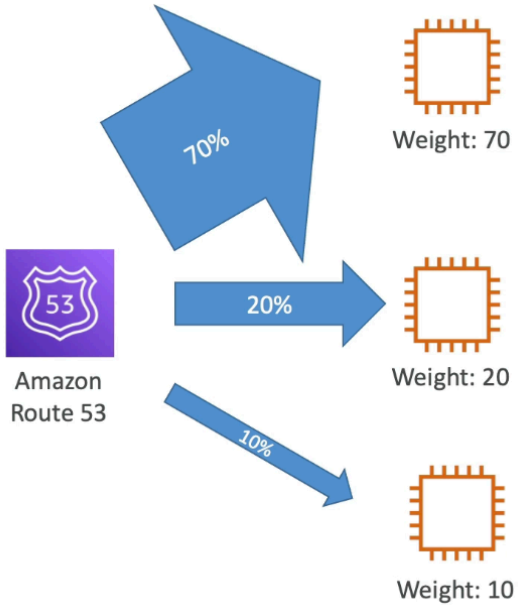
Weighted

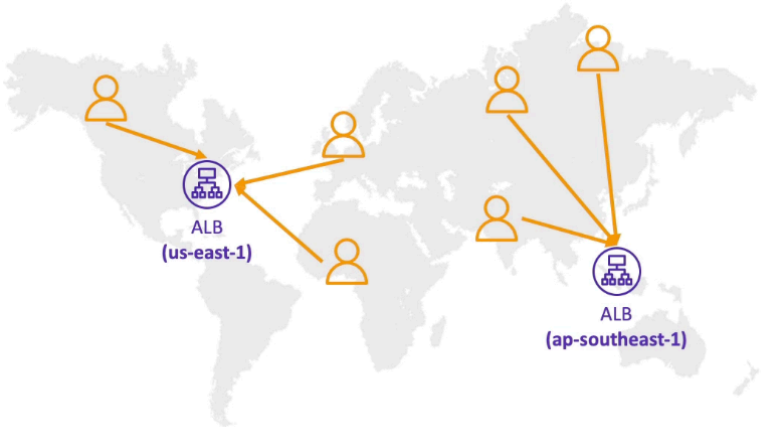
- Control % of requests that go to each specific resource
- Assign each record a relative weight
  - Traffic % = weight for specific record / sum of all weights for all records
  - No need to add up to 100
  - Weight = 0, no traffic
    - If all records have 0, all records are returned equally
- DNS records must have the same name and type
- Can be associated with health checks
- Use case: LB between regions, testing application versions…



Latency-based

- Redirect to the resource that has the least latency (helpful when latency is priority)
  - Latency based on traffic between users and AWS regions
- Can be associated with health checks (has failover capability)





Health Checks

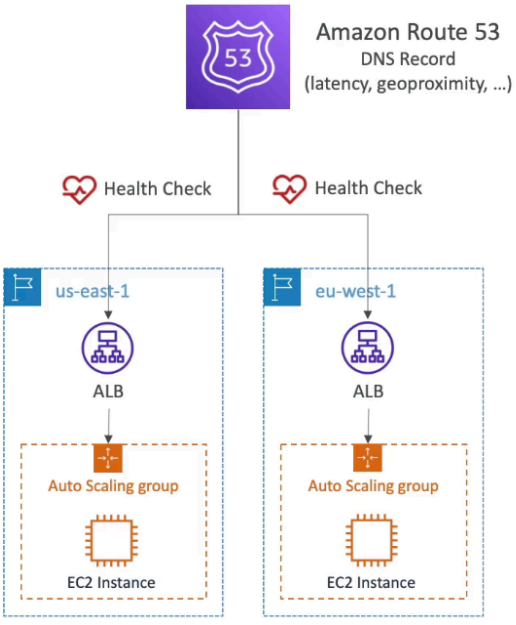
- Only for public resources
- Automated DNS failover
    - Monitor endpoint, other health checks, CloudWatch alarms
- 15 health checkers from all regions
    - Threshold for healthy/unhealthy
    - Interval for check
    - Supports HTTP/S, TCP
    - If > 18% is healthy, it is healthy; otherwise unhealthy
        - Passes when 2xx or 3xx status code is returned
        - Can be setup to pass/fail based on the text in first 5120 bytes of the response
    - Ability to choose which locations Route 53 uses
    - Configure your router/firewall to allow incoming requests from Route 53 health checkers

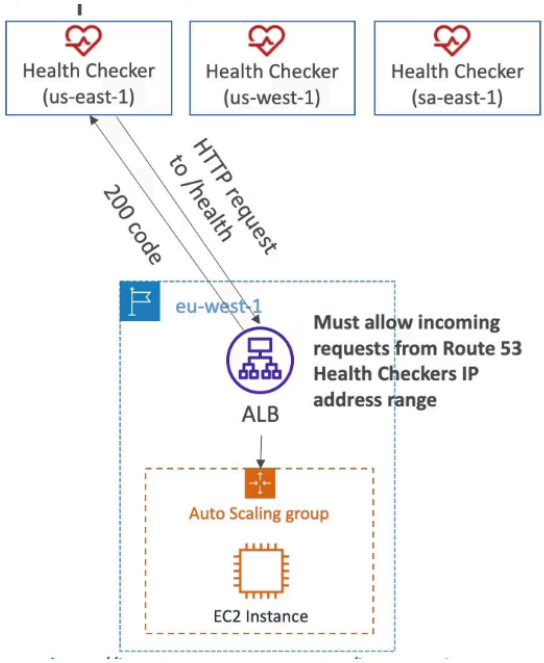
Calculated Health checks

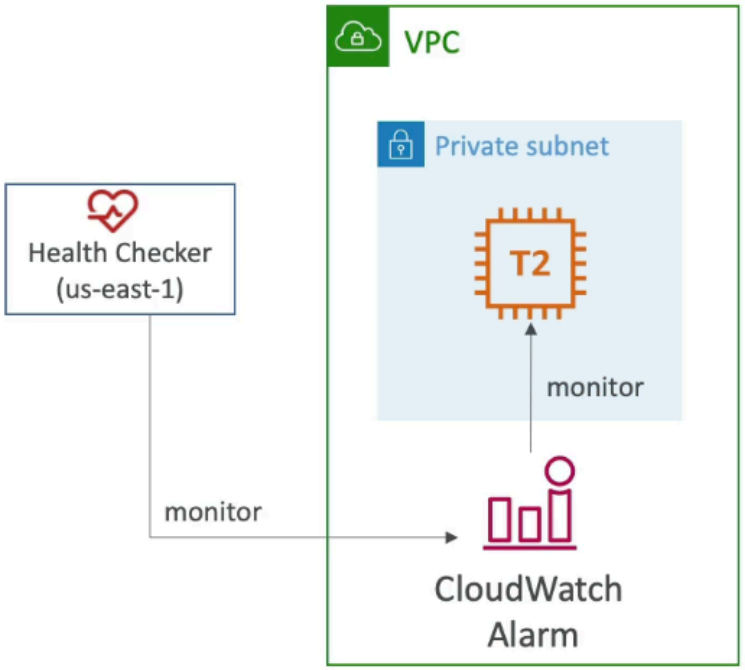
- Combine the results of multiple health checks into a single health checks
- Can use OR, AND, NOT
- Monitor up to 256 child health checks
- Specify how many health checks need to pass to make the parent pass
- Usage: perform maintenance without causing all checks to fail

Private Hosted Zone Health Checks

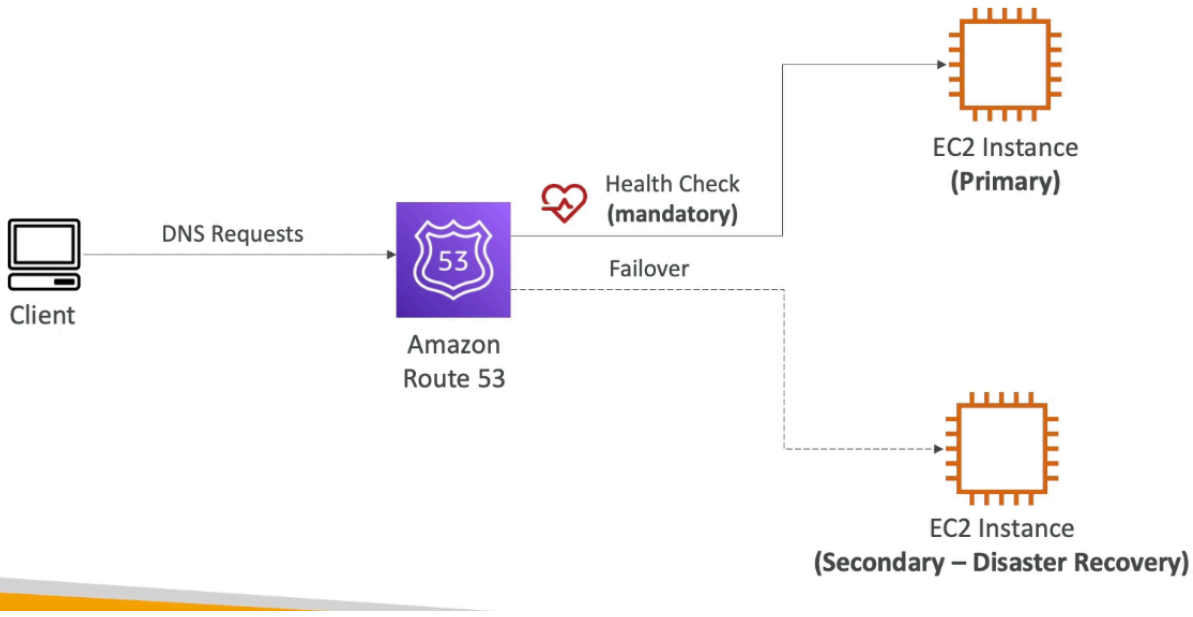
- Route 53 health checks are outside VPC, cannot access private endpoints
- Create a CloudWatch metric and associate an alarm, then create a health check that checks the alarm itself







Failover (Active - Passive)

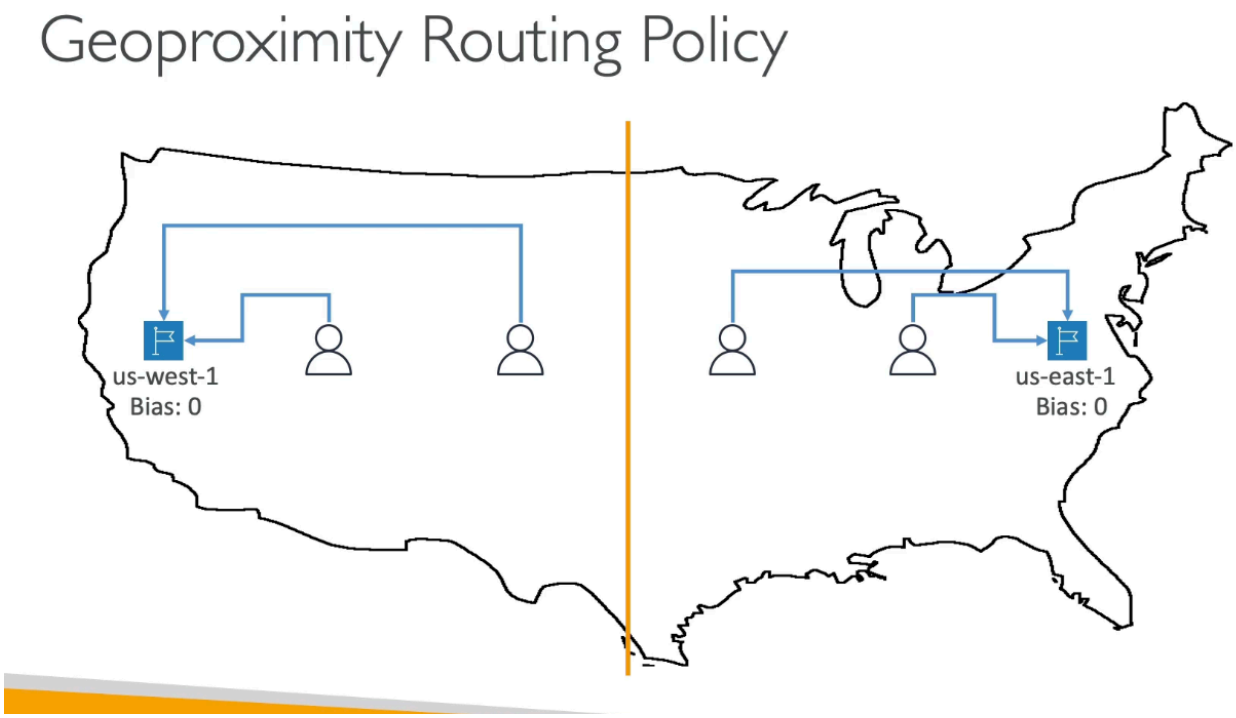


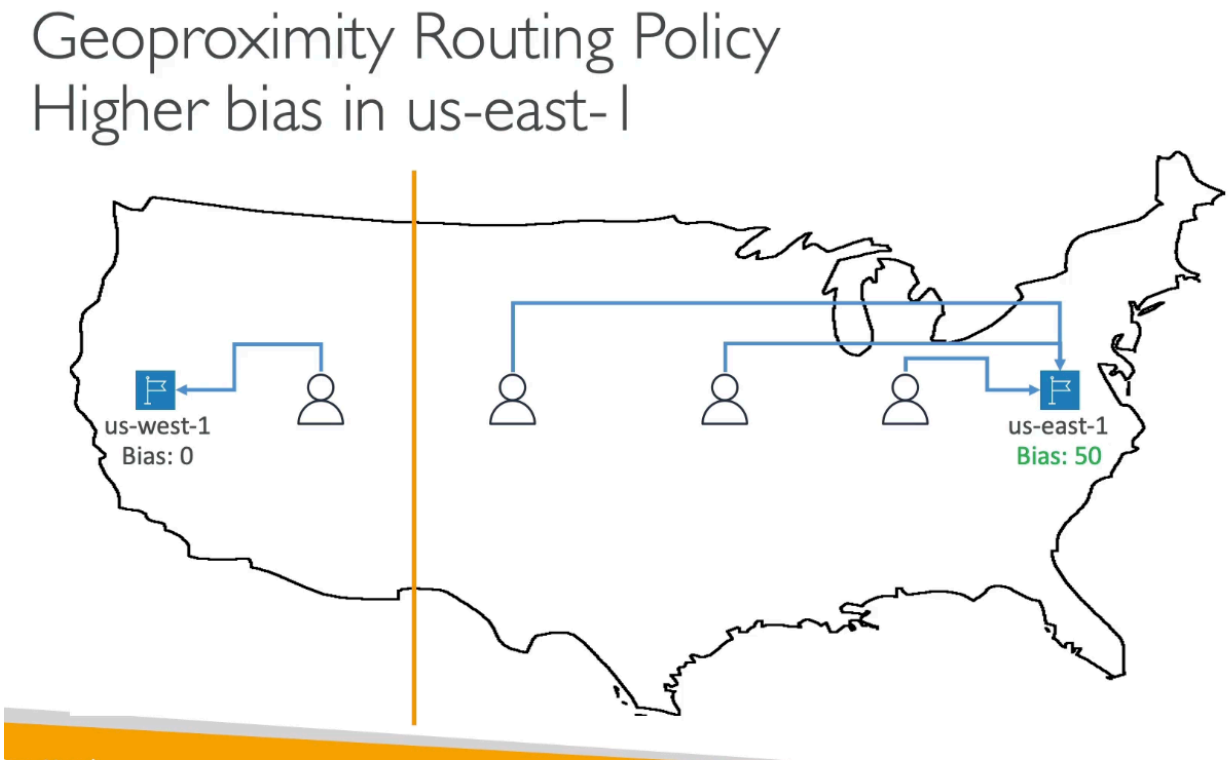
Geolocation

- Based on user location (continent, country, state), default record if no match
    - Restricts content distribution
    - Associated with health checks

Geo Proximity Routing

- Route traffic based on geographic location of users and resources
- Ability to shift more traffic to resources based on defined bias
- To change the size of geographic location, specify bias value:
    - To expand (1 to 99) - more traffic to resource
    - To shrink (-1 to -99) - less traffic to resource
- Resources can be:
    - AWS resources (specify AWS region)
    - Non AWS resources (specify latitude and longitude)
- You must use Route 53 Traffic flow to leverage bias



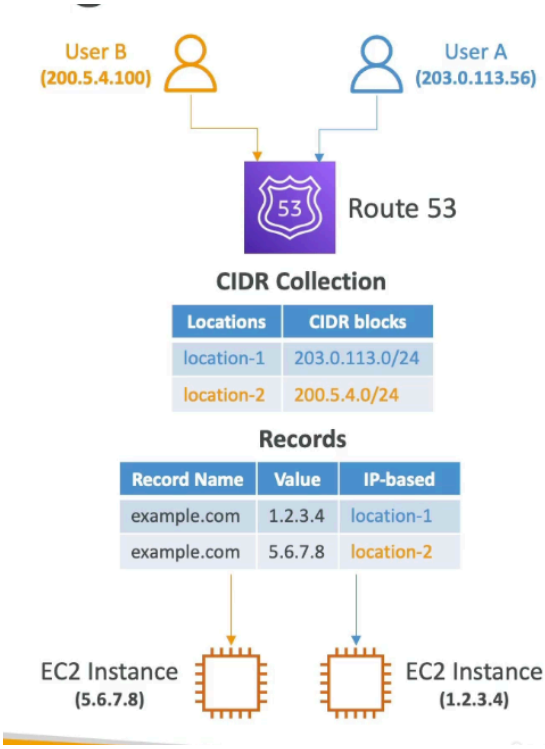


Traffic Flow

- Simplify the process of creating and maintaining records in large and complex configurations
    - Visual editor to manage complex routing decision trees
    - Configurations saved as traffic flow policy and can be applied to different Route 53 hosted zones (different domain names)

IP Based

- Routing based on clients' IP addresses
- Provide list of CIDRs and corresponding endpoints/locations (user IP to endpoint mapping)
- Use cases: optimize performance, reduce network costs



Multi Value Routing

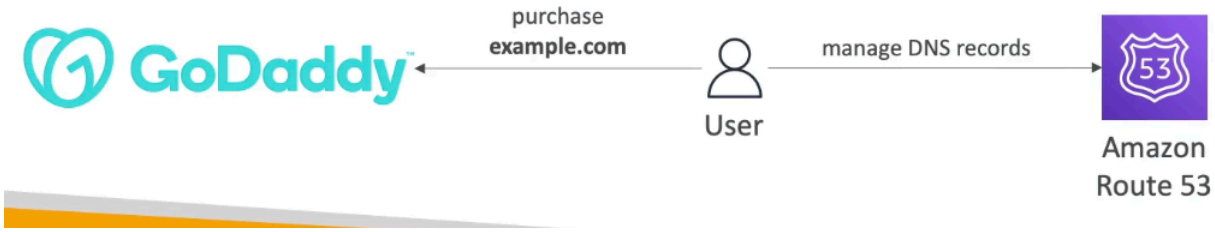
- Use when routing traffic to multiple resources
- Route 53 return multiple values / resources
    - Can be associated with health checks (return only values for healthy records)
    - Up to 8 healthy records are returned for each multi-value query
- Not a substitute for having ELB
- Different than simple routing with multiple records because simple doesn't have health checks

3rd Party Domains & Route 53

Domain Registrar vs DNS Service

- You can buy / register domain name with Domain Registrar
- Domain registrar usually provides a DNS service to manage DNS records, but can use another DNS service to manage DNS records
- Domain registrar != DNS service, but every domain registrar usually comes with some DNS features

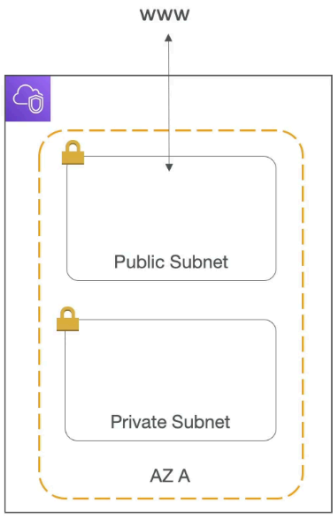
1. Create Hosted Zone in Route 53
2. Update NS records on 3rd party to use Route 53 Name Servers



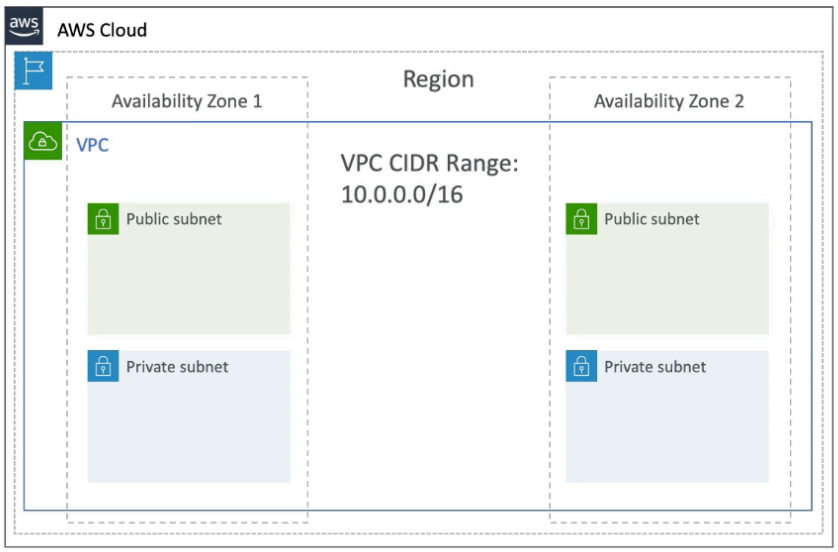
Section 10: VPC Fundamentals

VPC & Subnets Primer

- VPC: private network to deploy resources (regional resource)
- Subnets allow partitioning network inside VPC (AZ resource)
    - Public subnet is a subnet that is accessible from the internet
    - Private subnet is a subnet that is not accessible from the internet
- To define access to internet and between subnets, route tables are used



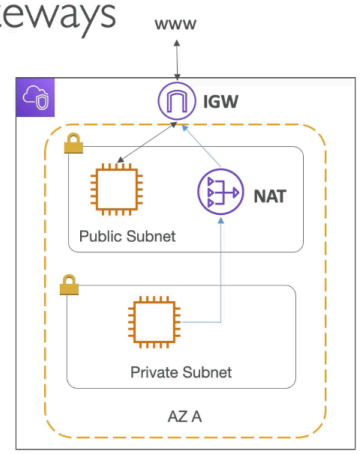
VPC Diagram



Internet Gateway & NAT Gateways

Internet Gateway & NAT Gateways

- Internet Gateways helps our VPC instances connect with the internet
- Public Subnets have a route to the internet gateway.
- NAT Gateways (AWS-managed) & NAT Instances (self-managed) allow your instances in your Private Subnets to access the internet while remaining private

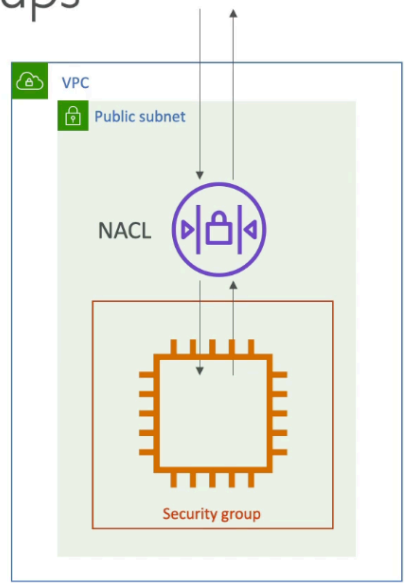


- Internet Gateway helps VPC instances connect with the internet
    - Public subnets have a route to internet gateway
- NAT Gateways (AWS managed) & NAT instances (self managed) allow instances in private subnets to access the internet while remaining private
    - NAT is created inside the public subnet and is a middle man between Internet GW and private subnet

Network ACL & Security Groups

Network ACL & Security Groups

- NACL (Network ACL)
  - A firewall which controls traffic from and to subnet
  - Can have ALLOW and DENY rules
  - Are attached at the Subnet level
  - Rules only include IP addresses
- Security Groups
  - A firewall that controls traffic to and from an ENI / an EC2 Instance
  - Can have only ALLOW rules
  - Rules include IP addresses and other security groups



- NACL: firewall which controls traffic from and to subnet
  - Allow and Deny rules
    - Rules only include IP addresses
  - Attached at subnet level
  - The first line of defense at subnet level
- Security Groups
  - Firewall that controls traffic to and from ENI / EC2 instance at INSTANCE level
  - Only allow rules
    - IP addresses and other SGs

Network ACLs vs Security Groups

Security Group	Network ACL
Operates at the instance level	Operates at the subnet level
Supports allow rules only	Supports allow rules and deny rules
Is stateful: Return traffic is automatically allowed, regardless of any rules	Is stateless: Return traffic must be explicitly allowed by rules
We evaluate all rules before deciding whether to allow traffic	We process rules in number order when deciding whether to allow traffic
Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on	Automatically applies to all instances in the subnets it's associated with (therefore, you don't have to rely on users to specify the security group)

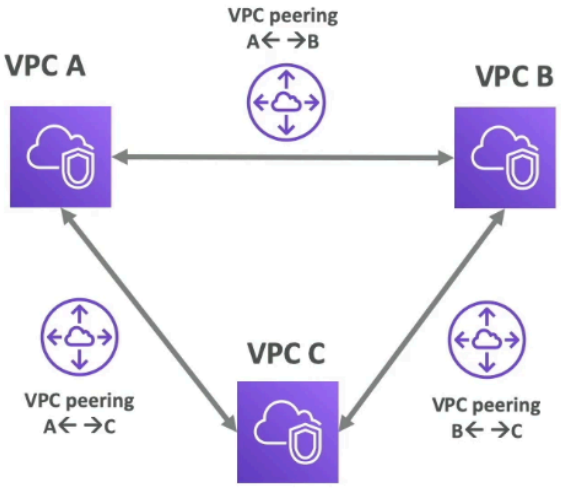
https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html#VPC_Security_Comparison

VPC Flow Logs

- Capture info about IP traffic going into interfaces:
  - VPC flow logs
  - Subnet flow logs
  - Elastic Network Interface flow logs
- Helps monitor / troubleshoot connectivity issues
- Captures network info from AWS managed interfaces and can be sent to other AWS resources (CloudWatch, S3)

VPC Peering

- Connect 2 VPC privately via AWS network, making them behave as if they were in the same network
- Must not have overlapping CIDR (IP address ranges)
- Connection is NOT transitive (must be established for each VPC that need to communicate with one another)



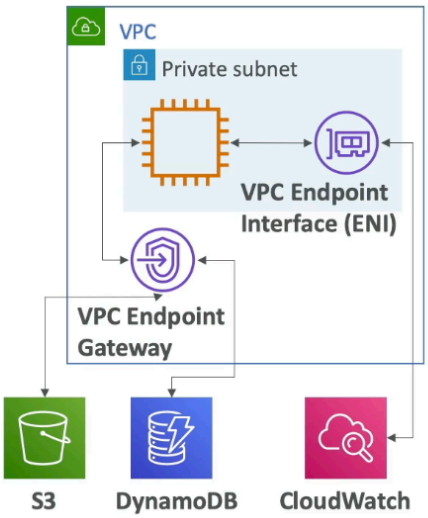
VPC Endpoints

VPC Endpoints

- Endpoints allow you to connect to AWS Services using a private network instead of the public www network
- This gives you enhanced security and lower latency to access AWS services

- VPC Endpoint Gateway: S3 & DynamoDB
- VPC Endpoint Interface: the rest

- Only used within your VPC

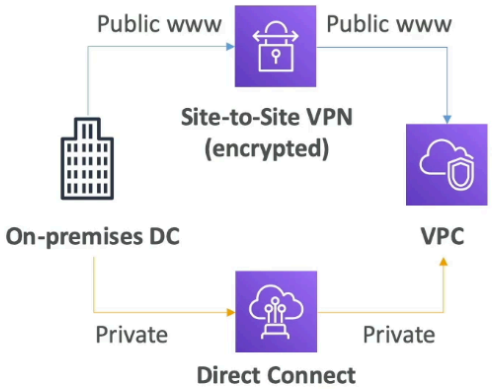


- Endpoints allow connection to AWS services using a private network instead of the public www network
    - VPC Endpoint Gateway: only for S3, DynamoDB
    - VPC Endpoint Interface: the rest
- Gives security and lower latency to access AWS services
- Only used within VPC

Site to Site VPN & Direct Connect

Site to Site VPN & Direct Connect

- Site to Site VPN
    - Connect an on-premises VPN to AWS
    - The connection is automatically encrypted
    - Goes over the public internet
- Direct Connect (DX)
    - Establish a physical connection between on-premises and AWS
    - The connection is private, secure and fast
    - Goes over a private network
    - Takes at least a month to establish

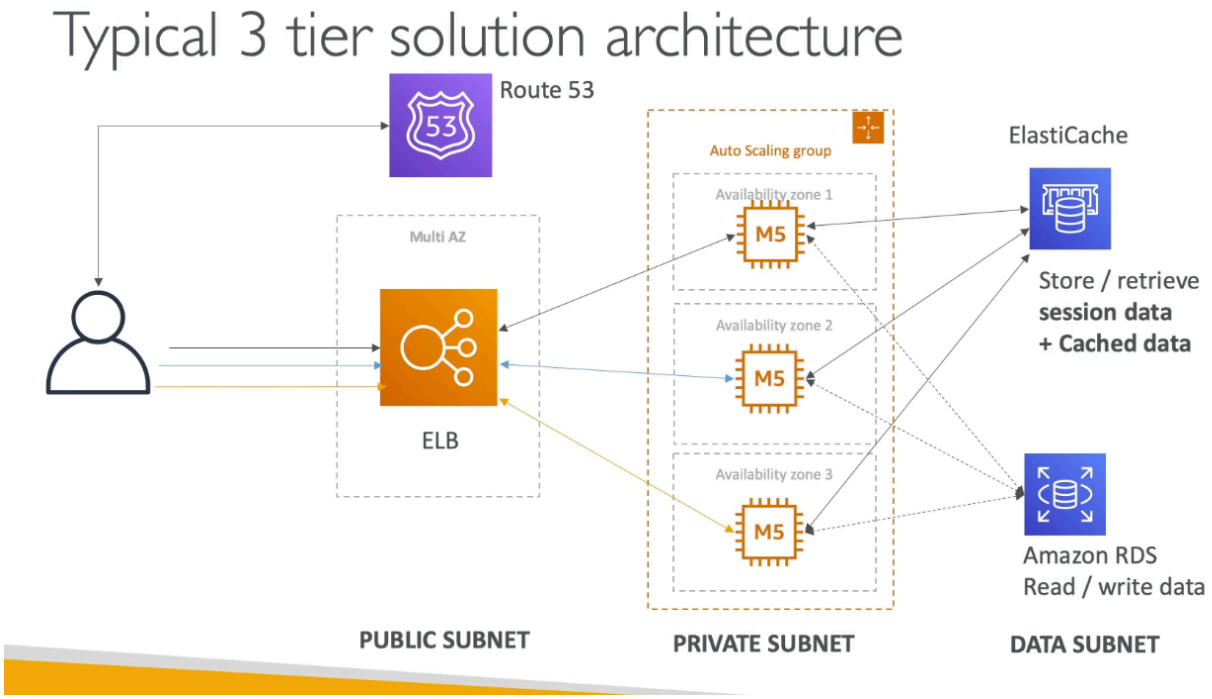


VPC Closing Comments

VPC Closing Comments

- VPC: Virtual Private Cloud
- Subnets: Tied to an AZ, network partition of the VPC
- Internet Gateway: at the VPC level, provide Internet Access
- NAT Gateway / Instances: give internet access to private subnets
- NACL: Stateless, subnet rules for inbound and outbound
- Security Groups: Stateful, operate at the EC2 instance level or ENI
- VPC Peering: Connect two VPC with non overlapping IP ranges, non transitive
- VPC Endpoints: Provide private access to AWS Services within VPC
- VPC Flow Logs: network traffic logs
- Site to Site VPN: VPN over public internet between on-premises DC and AWS
- Direct Connect: direct private connection to a AWS

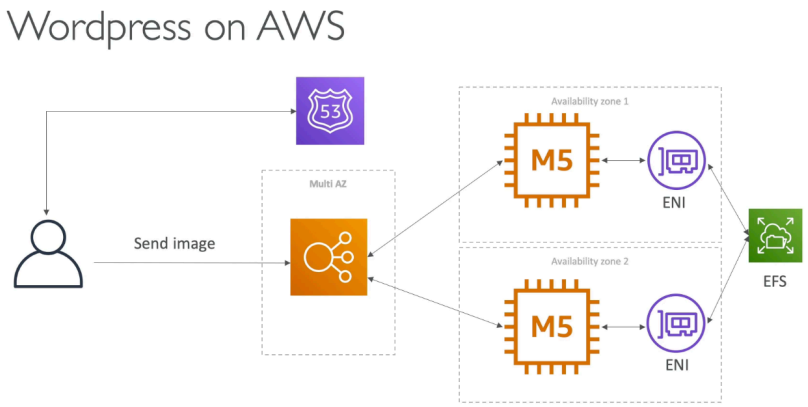
Three Tier Architecture

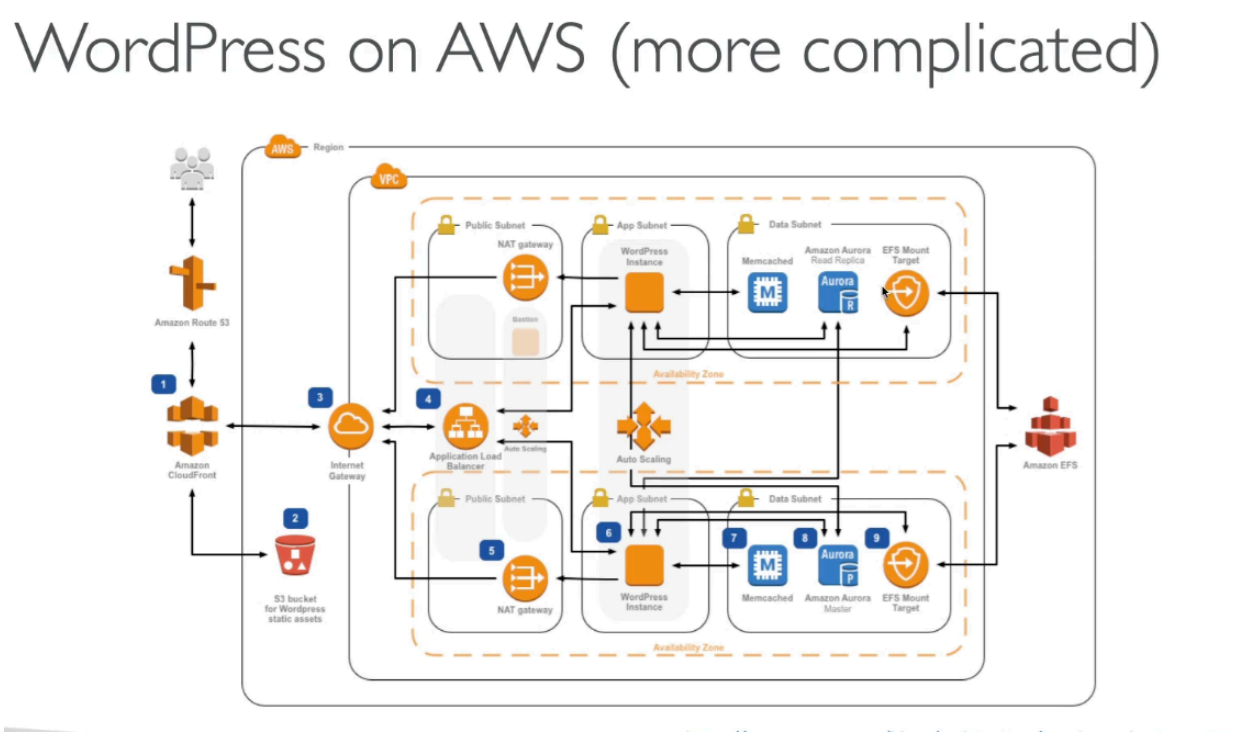


LAMP Stack on EC2

- Linux (OS on EC2 instances), Apache (web server to run on linux - EC2), MySQL (DB on RDS), PHP (application logic on EC2) → LAMP
    - Can add redis/memcached (elasticache) to include a caching technology
    - To store local app data & software: EBS drive (root)

Wordpress on AWS



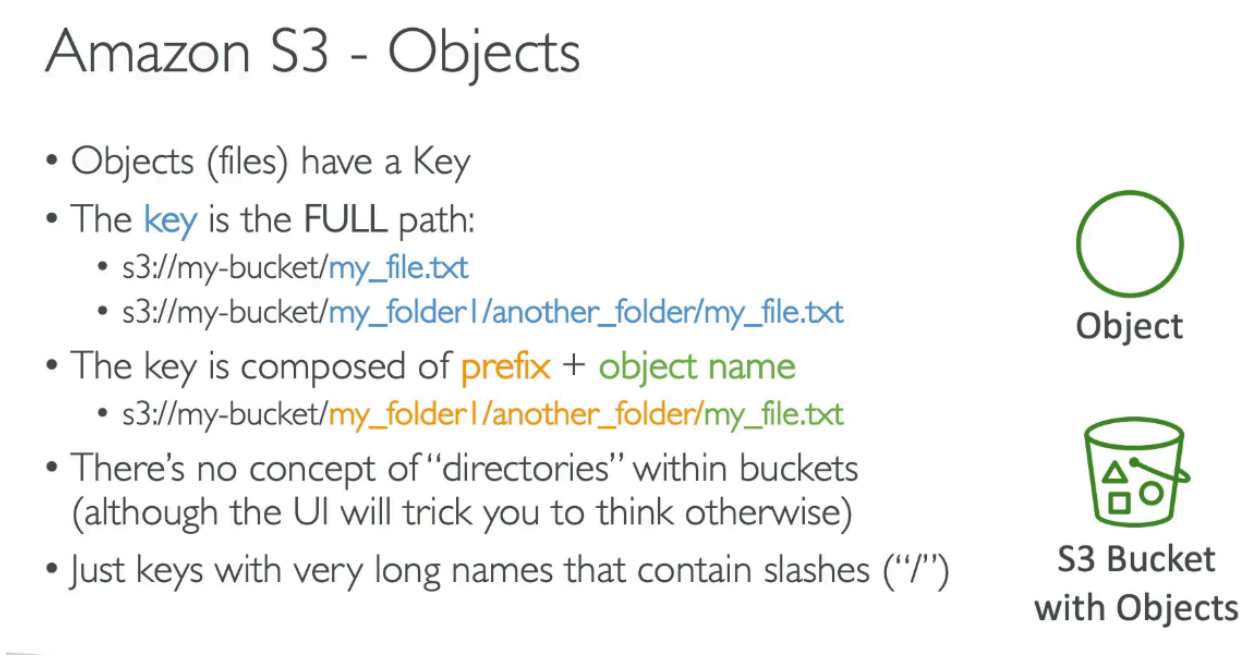


Section 11: S3 Intro

S3 Overview

- Backup and storage, disaster recovery, archive, hybrid cloud storage, application hosting
- Buckets (directories) allow storage of objects (files) and must be globally unique name across all regions
    - However, buckets are regional level created
- Naming convention: no uppercase, no underscore, 3-63 char long, not IP, not start with xn–, and not end in -s3alias

S3 Objects



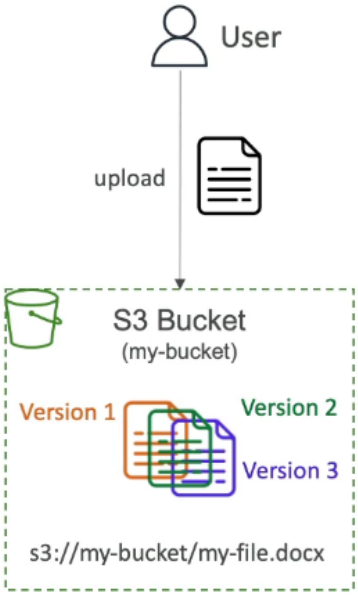
- Objects (files) have a key where key is the full path
    - Key is composed of prefix + object name
- There is no concept of directories within buckets, only keys that are just long names that contain slashes
- Object values are the content of the body:
    - Max size: 5TB, but if uploading more than 5GB, must use multi-part upload
- Metadata: list of text key / pair values - system or user metadata
- Tags: unicode key / value pair - up to 10 - useful for security / lifecycle
- Version ID (if enabled)

S3 Security: Bucket Policy

- User based
    - IAM Policies: which API calls allowed for specific user from IAM
- Resource Based
    - Bucket policies - bucket wide rules that allow cross account access
    - Object Access Control List (ACL) - finer grain (can be disabled)
    - Bucket Access Control List (ACL) - less common (can be disabled)
- Note: IAM principal can access S3 object if
    - User IAM permissions allow OR resource policy allows
    - AND no explicit deny
- Encryption: using encryption keys
- JSON based policies

    - Resource block: buckets and objects
    - Effect: allow/deny
    - Actions: set of API to allow/deny
    - Principal: account or user to apply policy to
- Use S3 bucket for policy to:
    - Grant public access to bucket
    - Force objects to be encrypted at upload
    - Grant access to another account (cross account)

S3 Static Website Hosting

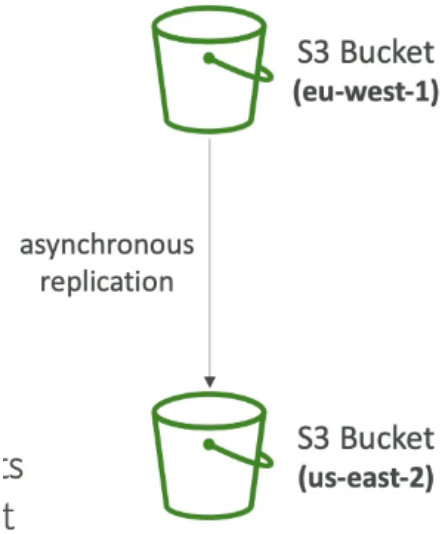


- S3 Can host static websites and accessible on internet
    - 403 forbidden error, make sure bucket policy allows public reads
- Website URL varies based on region, but is from the bucket website endpoint

S3 Versioning

- Enabled at bucket level
- Same key overwrite will change the version to: 1, 2, 3…
    - Protect against unintended deletes (ability to restore version)
    - Easy roll back
- Notes:
    - Any file not versioned prior to enabling versioning will have version “null”
    - Suspending versioning does not delete previous versions
    - Delete marker is on deleted items

S3 Replication (CRR & SRR)



- Versioning must be enabled on source and destination buckets
- CRR = Cross region replication
- SRR = same region replication
- Buckets can be in different AWS accounts
- Copying is asynchronous
- Must give proper IAM permissions to S3
- Use cases:
    - CRR: compliance, low latency access, replication across accounts
    - SRR: log aggregation, live replication between prod and test
- After replication is enabled, only new objects are replicated

- Optionally you can replicate existing objects using S3 Batch Replication
    - Replicates existing objects and objects that failed replication
- For delete operations:
    - Can replicate delete markers from source to target (optional) → delete marker replication
    - Deletions with version ID are not replicated (to avoid malicious deletes)
- No “chaining” of replication
    - If bucket 1 has replication into bucket 2, which has replication into bucket 3
    - Then objects created in bucket 1 are not replicated to bucket 3

S3 Storage Classes

S3 Storage Classes

- Amazon S3 Standard - General Purpose
- Amazon S3 Standard-Infrequent Access (IA)
- Amazon S3 One Zone-Infrequent Access
- Amazon S3 Glacier Instant Retrieval
- Amazon S3 Glacier Flexible Retrieval
- Amazon S3 Glacier Deep Archive
- Amazon S3 Intelligent Tiering

- Can move between classes manually or using S3 Lifecycle configurations

S3 Durability and Availability

- Durability:
    - High durability (99.999999999%, 11 9's) of objects across multiple AZ
    - If you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years
    - Same for all storage classes

- Availability:
    - Measures how readily available a service is
    - Varies depending on storage class
    - Example: S3 standard has 99.99% availability = not available 53 minutes a year

General Purpose

- Frequently accessed data, low latency and high throughput
- Sustain 2 concurrent facility failures

Infrequent Access

- Amazon S3 Standard-Infrequent Access (S3 Standard-IA)
  - 99.9% Availability
  - Use cases: Disaster Recovery, backups

- Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA)
  - High durability (99.999999999%) in a single AZ; data lost when AZ is destroyed
  - 99.5% Availability
  - Use Cases: Storing secondary backup copies of on-premise data, or data you can recreate

- Less frequently accessed data, but requires rapid access when needed
- Lower cost than S3 Standard, but cost on retrieval

Glacier Storage Classes

Amazon S3 Glacier Storage Classes

- Low-cost object storage meant for archiving / backup
- Pricing: price for storage + object retrieval cost

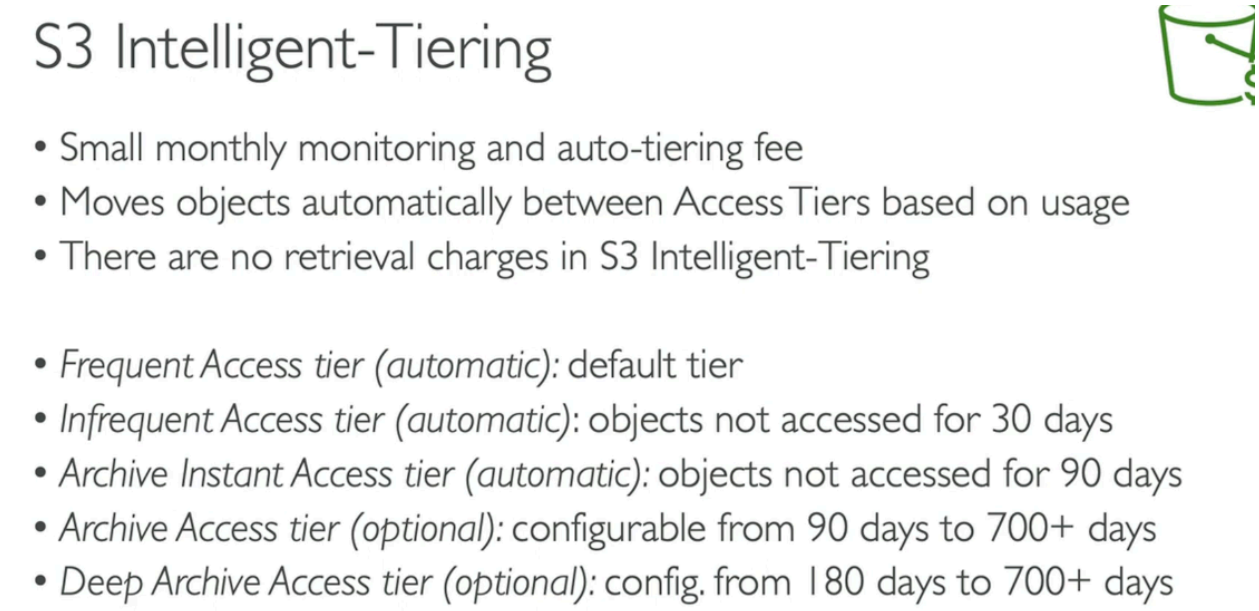
- Amazon S3 Glacier Instant Retrieval
  - Millisecond retrieval, great for data accessed once a quarter
  - Minimum storage duration of 90 days

- Amazon S3 Glacier Flexible Retrieval (formerly Amazon S3 Glacier):
  - Expedited (1 to 5 minutes), Standard (3 to 5 hours), Bulk (5 to 12 hours) – free
  - Minimum storage duration of 90 days

- Amazon S3 Glacier Deep Archive – for long term storage:
  - Standard (12 hours), Bulk (48 hours)
  - Minimum storage duration of 180 days

- Low cost object storage for archiving / backup
- Pricing: price for storage + object retrieval cost

S3 Intelligent Tiering



- Small monthly monitoring and auto-tiering fee
- Moves objects automatically between Access Tiers based on usage
- No retrieval charges

Section 12: AWS CLI, SDK, IAM Roles & Policies

EC2 Instance Metadata (IMDS)

- Allows EC2 instances to “learn about themselves” without using IAM role for that purpose
- You can retrieve IAM role name from metadata, but cannot retrieve IAM policy
- Metadata = info about EC2 instance
- Userdata = launch script of EC2 instance

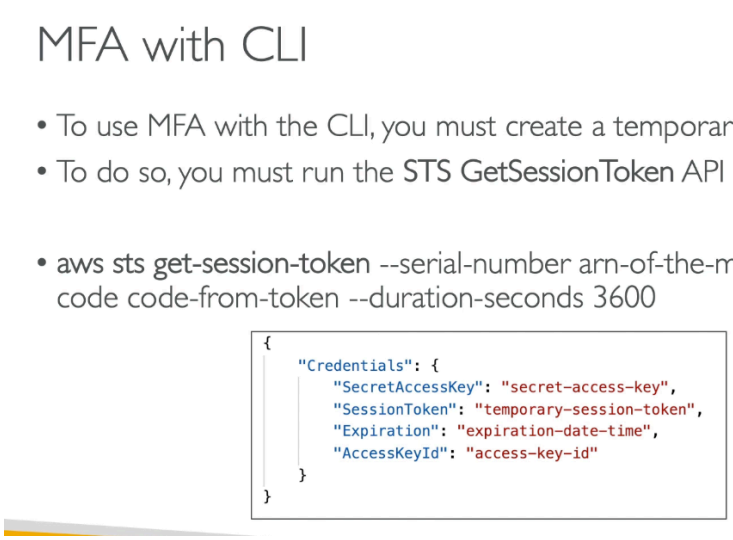
IMDSv1 vs IMDSv1

- IMDSv1 access metadata URL directly
- IMDSv2 is more secure and done in 2 steps:
    1. Get session token (limited validity) using headers and PUT
    2. Use session token in IMDSv2 calls using headers

AWS CLI Profiles

- Multiple profiles on CLI can be set by using aws configure –profile name

MFA with CLI



- Must create a temp session running STS GetSessionToken API call

AWS SDK Overview

- SDK to perform AWS actions via code
- US East 1 chosen by default region if no region is selected

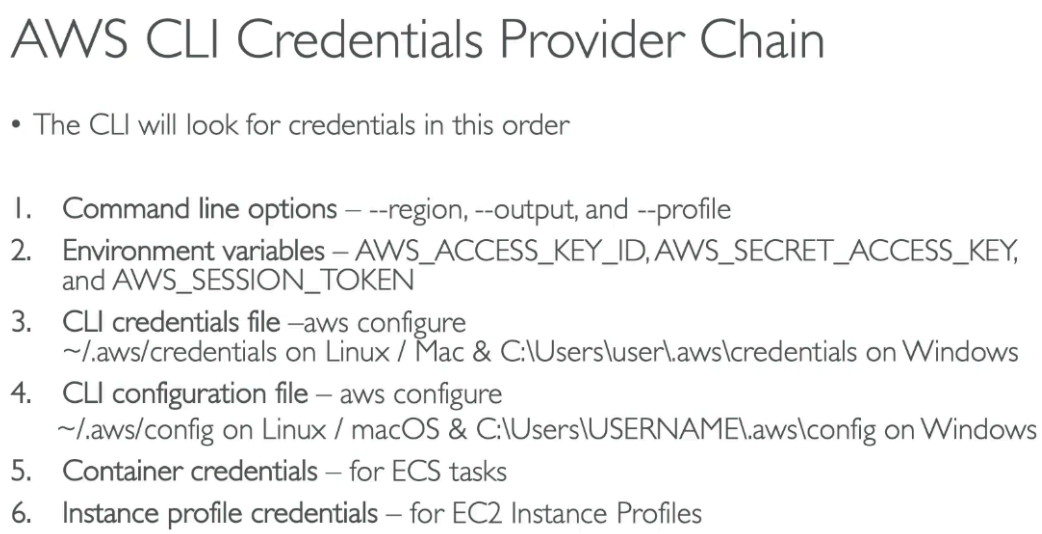
AWS Limits (Quotas)

- API Rate Limits
    - Intermittent Errors: implement exponential backoff
    - For consistent errors: request API throttling limit increase
- Service Quotas (service limits)
    - Service limit increase by opening ticket and service quota increase by using service quota API

Exponential Backoff (any AWS service)

- When an error is reached, a retry is done exponentially (doubling each retry)
- If ThrottlingException intermittently, use exponential backoff
- Retry mechanism already included in SDK API calls
- Must implement yourself if using AWS API as is or in specific cases
    - Must only implement the retries on 5xx server errors and throttling
        - Do not implement on 4xx client errors

AWS CLI Credentials Provider Chain



- CLI looks for credentials in this order:

    1. CLI options
    2. ENV vars
    3. CLI cred file
    4. CLI config file
    5. Container credentials
    6. Instance profile creds

AWS SDK Default Credentials Provider Chain

1. Java system properties
2. ENV variables
3. Default credential profiles file
4. Amazon ECS container credentials - for ECS containers
5. Instance profile credentials - used on E2 instances

Scenario

AWS Credentials Scenario

- An application deployed on an EC2 instance is using environment variables with credentials from an IAM user to call the Amazon S3 API.
- The IAM user has S3FullAccess permissions.
- The application only uses one S3 bucket, so according to best practices:
  - An IAM Role & EC2 Instance Profile was created for the EC2 instance
  - The Role was assigned the minimum permissions to access that one S3 bucket

- The IAM Instance Profile was assigned to the EC2 instance, but it still had access to all S3 buckets. Why?

  the credentials chain is still giving priorities to the environment variables

AWS Credentials Best Practices

- Never store AWS creds in code, best practice to be inherited from credentials chain
- If working with AWS, use IAM roles
- If working outside AWS, use env variables

Signing AWS API Requests

SigV4 Request examples

- HTTP Header option (signature in Authorization header)

```
GET https://iam.amazonaws.com/?Action=ListUsers&Version=2010-05-08 HTTP/1.1
Authorization: AWS4-HMAC-SHA256 Credential=AKIDEXAMPLE/20150830/us-east-1/iam/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=5d672d79c15b13162d9279b0855cfba6789a8edb4c82c400e06b5924a6f2b5d7
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
x-amz-date: 20150830T123600Z
```

- Query String option, ex: S3 pre-signed URLs (signature in X-Amz-Signature)

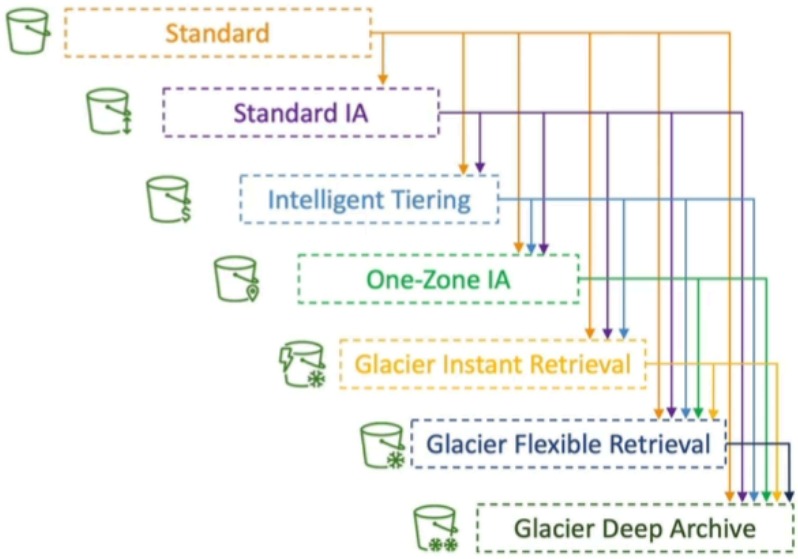
```
GET https://iam.amazonaws.com?Action=ListUsers&Version=2010-05-08&
X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Credential=AKIDEXAMPLE%2F20150830%2Fus-east-1%2Fiam%2Faws4_request&
X-Amz-Date=20150830T123600Z&X-Amz-Expires=60&X-Amz-SignedHeaders=content-type%3Bhost&
X-Amz-Signature=37ac2f4fde00b0ac9bd9eadeb459b1bbee224158d66e7ae5fcadb70b2d181d02 HTTP/1.1
content-type: application/x-www-form-urlencoded; charset=utf-8
host: iam.amazonaws.com
```

- When calling AWS HTTP API, sign the request so AWS can identify you using AWS creds
    - Some requests to S3 don't need to be signed
    - If you use SDK or CLI, HTTP requests are signed for you
- You should sign AWS HTTP request using Signature v4 (SigV4)
    - Sent via:
        - HTTP header in Authorization header
        - Query string in the URL

Section 13: Advanced S3

S3 Lifecycle Rules (with S3 analytics)

- Transition objects between storage classes
- For infrequently accessed objects, move to Standard IA. For archive objects that you don't need fast access to, move to Glacier or Glacier Deep archive
- Moving objects automated with lifecycle rules
- Transition Action: configure object to transition to another storage class
- Expiration actions: configure objects to expire (delete) after some time
    - Can delete old versions of files (for versioning)
- Rules can be specified for certain prefix
- Rules can be created for certain object tags



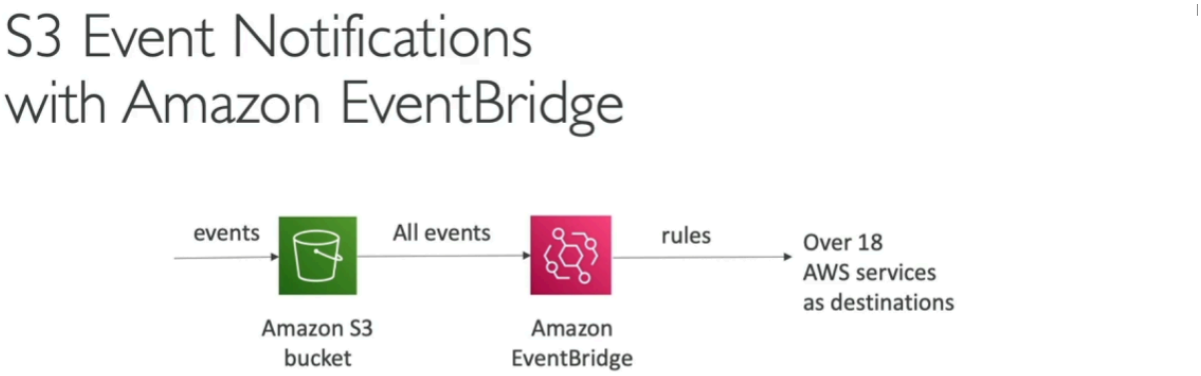
S3 Storage Class Analytics

- Helps decide when to transition objects to the right storage class
- Recommendations for Standard and Standard IA
    - Does not work for one zone IA or Glacier
- Report updated daily
    - 24 to 48 hours to start seeing data analytics

S3 Event Notifications

- React any time an action occurs in S3 bucket
  - S3:ObjectCreated, S3:ObjectRemoved, S3:ObjectRestore, S3:Replication
  - Object name filtering possible
- Must set IAM permissions
  - Other services must have a resource policy set to retrieve from bucket
- SNS, SQS, and Lambda function are Event Notification targets

S3 Event Notifications with Amazon EventBridge



- Enhanced version with advanced filtering options, multiple destinations, and EventBridge Capabilities such as Archive, Replay events, Reliable Delivery

S3 Performance

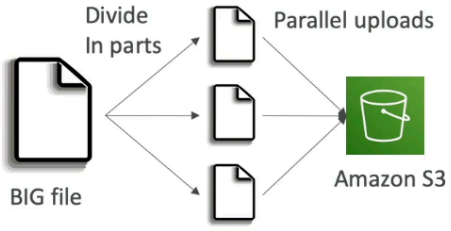
S3 – Baseline Performance

- Amazon S3 automatically scales to high request rates, latency 100-200 ms
- Your application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in a bucket.
- There are no limits to the number of prefixes in a bucket.
- Example (object path => prefix):
  - bucket/folder1/sub1/file => /folder1/sub1/
  - bucket/folder1/sub2/file => /folder1/sub2/
  - bucket/1/file => /1/
  - bucket/2/file => /2/
- If you spread reads across all four prefixes evenly, you can achieve 22,000 requests per second for GET and HEAD

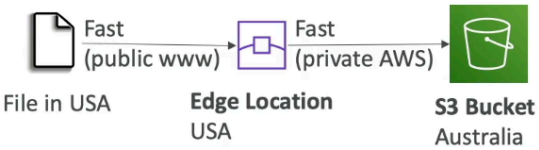
- Automatically scales to high request rates with low latency
  - 3500 PUT/COPY/POST/DELETE or 5500 GET/HEAD requests per second per prefix in a bucket
  - No limit to number of prefixes in a bucket

S3 Performance

- Multi-Part upload:
  - recommended for files > 100MB, must use for files > 5GB
  - Can help parallelize uploads (speed up transfers)



- S3 Transfer Acceleration
  - Increase transfer speed by transferring file to an AWS edge location which will forward the data to the S3 bucket in the target region
  - Compatible with multi-part upload



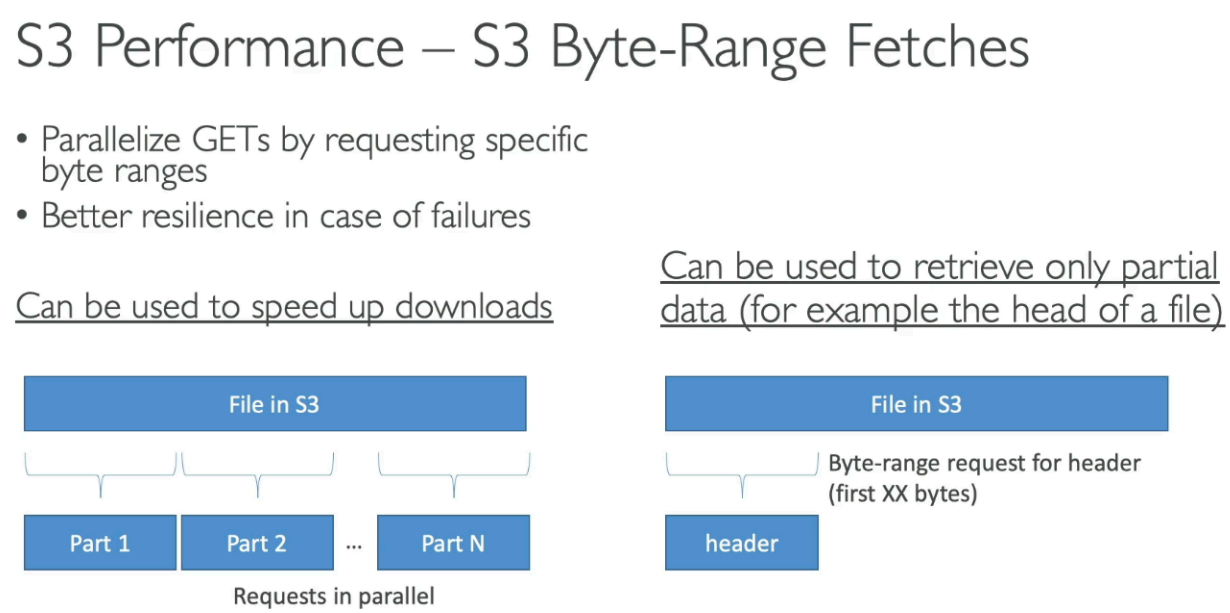
Multi-Part Upload

- Recommended for files > 100MB, must use for > 5GB
- Can help parallelize uploads (speed up transfers)

S3 Transfer Acceleration

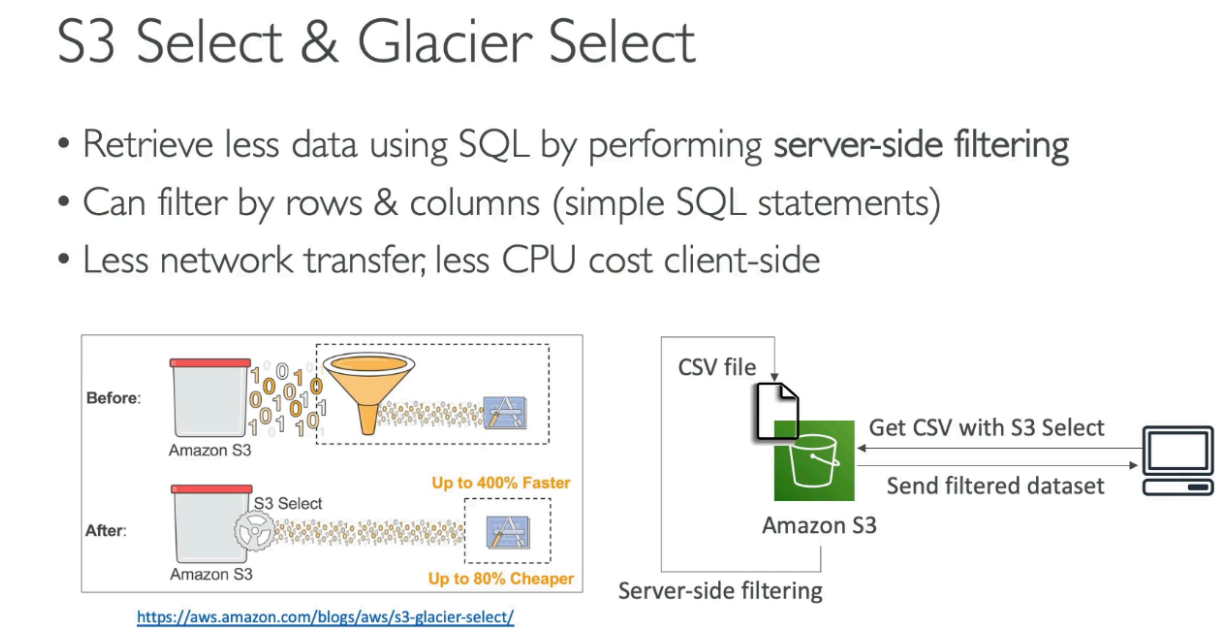
- Increase transfer speed by transferring file to AWS Edge location which will forward the data to S3 bucket in the target region
- Compatible with Multi-part upload

S3 Byte Range Fetches



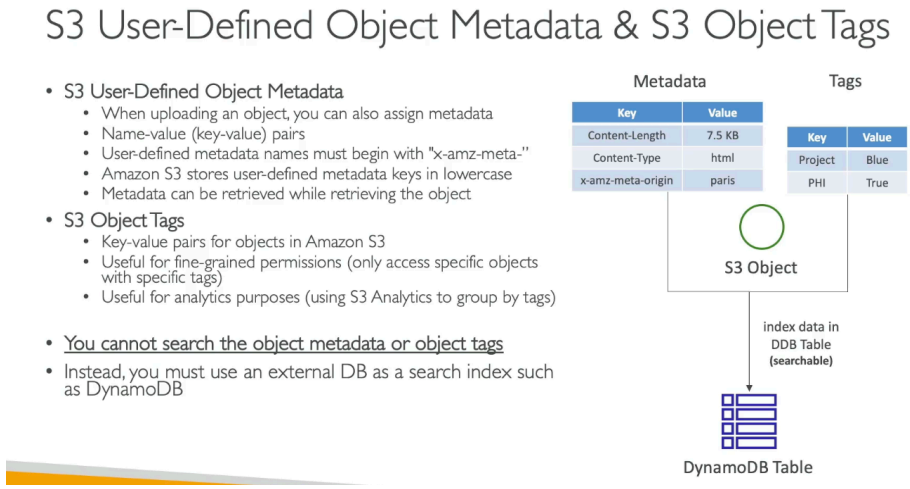
- Parallelize GETs by requesting specific byte ranges
- Better resilience in case of failures
- Can be used to speed up downloads
- Can be used to retrieve only partial data (ex: head of a file)

S3 Select & Glacier Select



- Retrieve less data using SQL by performing server-side filtering
    - Can filter by rows/columns
- Less network transfer, less CPI cost client-side

S3 User-Defined Object Metadata & S3 Object Tags



- Cannot search object metadata or object tags
    - Instead, you must use external DB as search index

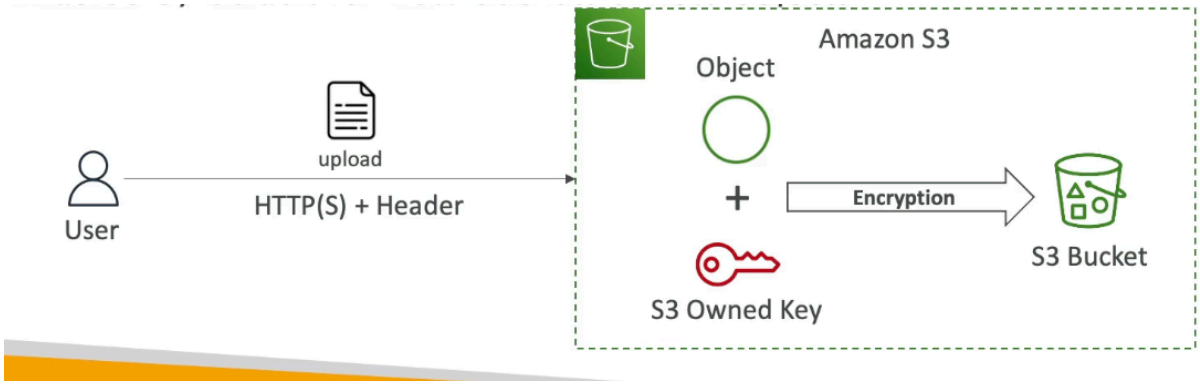
- S3 User-Defined Object Metadata
    - When uploading an object, can assign metadata via name-value pairs
    - User-defined metadata names must begin with “x-amz-meta-”
    - S3 stores user-defined metadata in lowercase
    - Metadata can be retrieved while retrieving the object
- S3 Object Tags
    - Key value pairs for objects in S3
    - Useful for fine-grained permissions or analytics (only access specific objects with tag)

Section 14: Amazon S3 Security

S3 Encryption

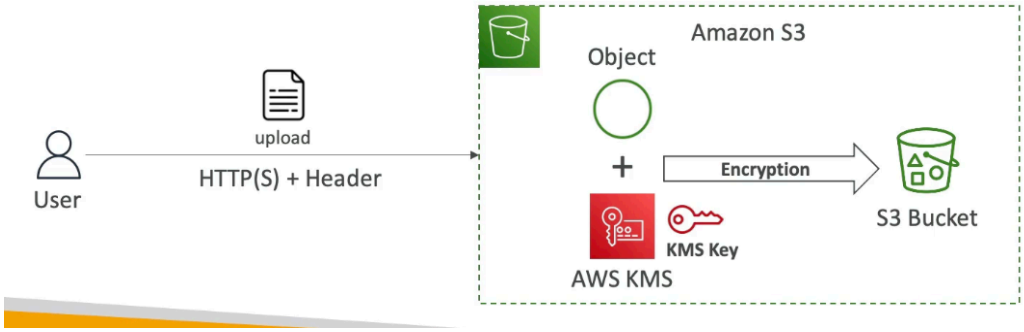
- Server Side Encryption (SSE)
    - Amazon S3 managed keys (SSE-S3) - enabled by default
        - Encrypts S3 objects using keys managed by AWS
    - SSE-KMS with KMS keys
        - Leverage AWS KMS to manage keys
    - SSE with Customer Provided Keys (SSE-C)
        - Manage own encryption keys
- Client Side Encryption

SSE-S3 Encryption



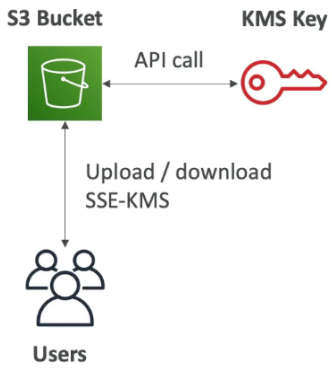
- Encryption is handled server side and keys are AWS managed and owned
    - Encryption type is AES-256
- Must set header “x-amz-server-side-encryption”: “AES256”
- Enabled by default for new buckets/objects

SSE-KMS Encryption



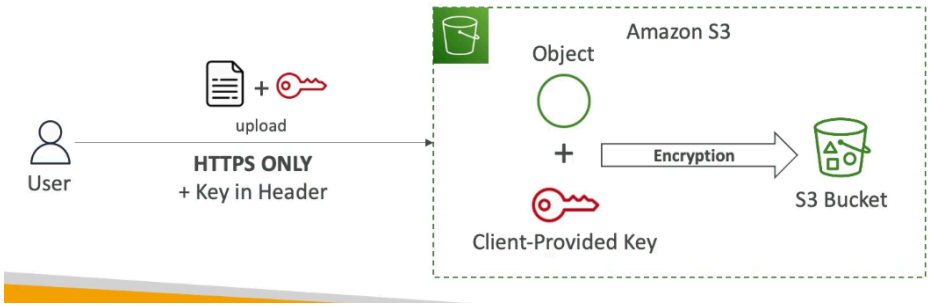
- Encryption using keys managed by AWS KMS
    - User control + audit key usage via CloudTrail
- Header: “x-amz-server-side-encryption”: “aws:kms”

KMS Limitation



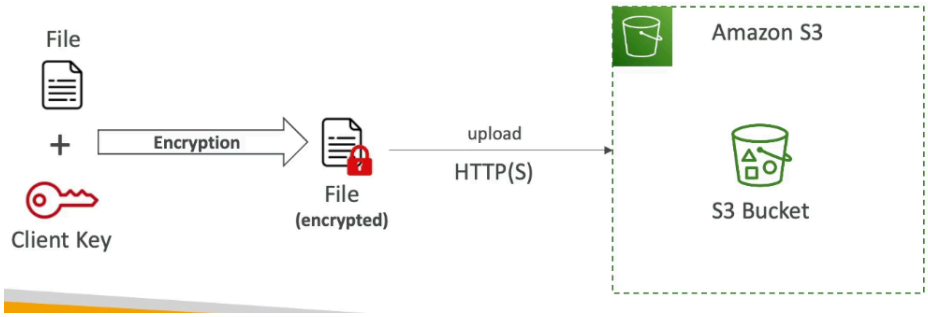
- If you use KMS key, may be impacted with KMS limits
- When you upload, it calls GenerateDataKey KMS API and download calls Decrypt KMS API
    - Count towards KMS quota per second, but can request a quota increase using Service Quotas Console

SSE-C



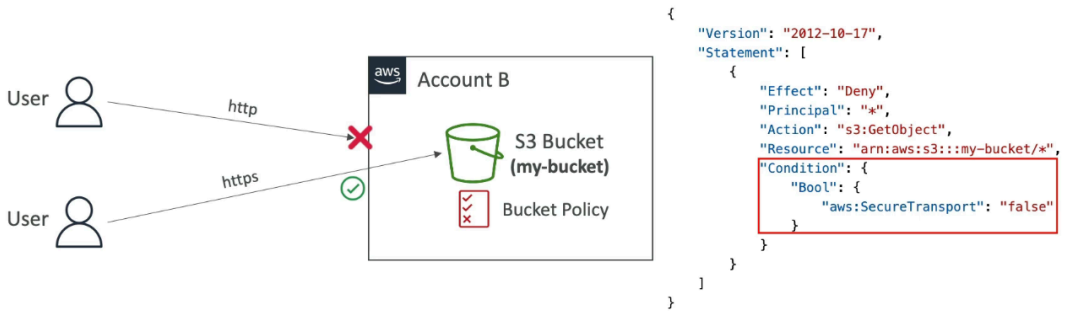
- Server side encryption fully managed outside of AWS by customer
    - S3 does not store encryption key
- HTTPS must be used
- Encryption key must be provided in HTTP headers for every HTTP request made

Client Side Encryption



- Client libraries to implement where clients must encrypt data themselves before sending to S3 and decryption of data when retrieving from S3
- Customer fully manages the keys and encryption cycle

Encryption in Transit (SSL / TLS)

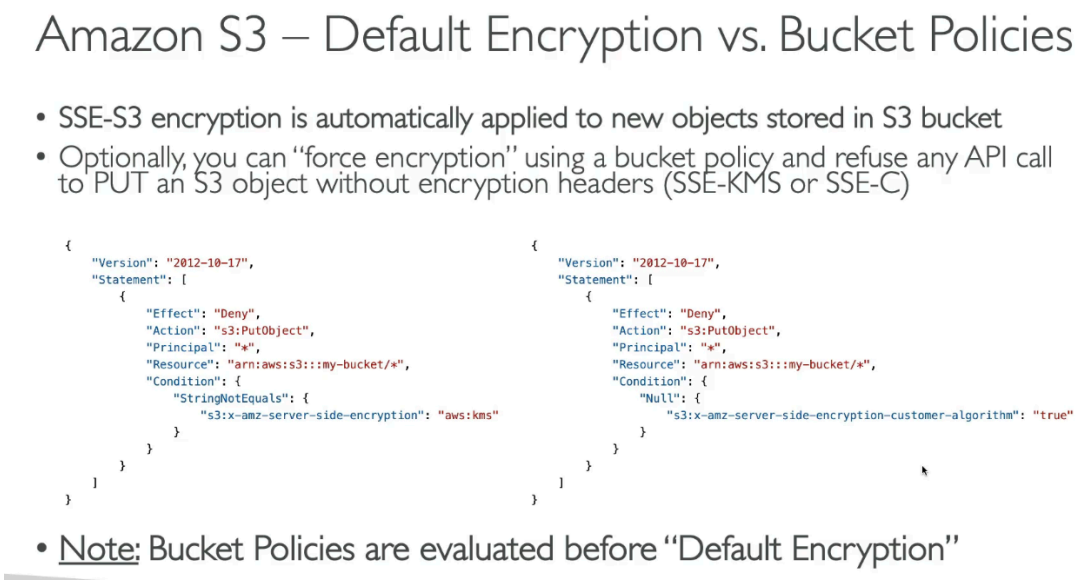


- Encryption in flight also called SSL / TLS
- S3 exposes 2 endpoints:
    - HTTP (non encrypted)
    - HTTPS (encryption in flight)
- HTTPS recommended and mandatory for SSE-C
    - Most clients use HTTPS endpoints by default

Force Encryption in Transit

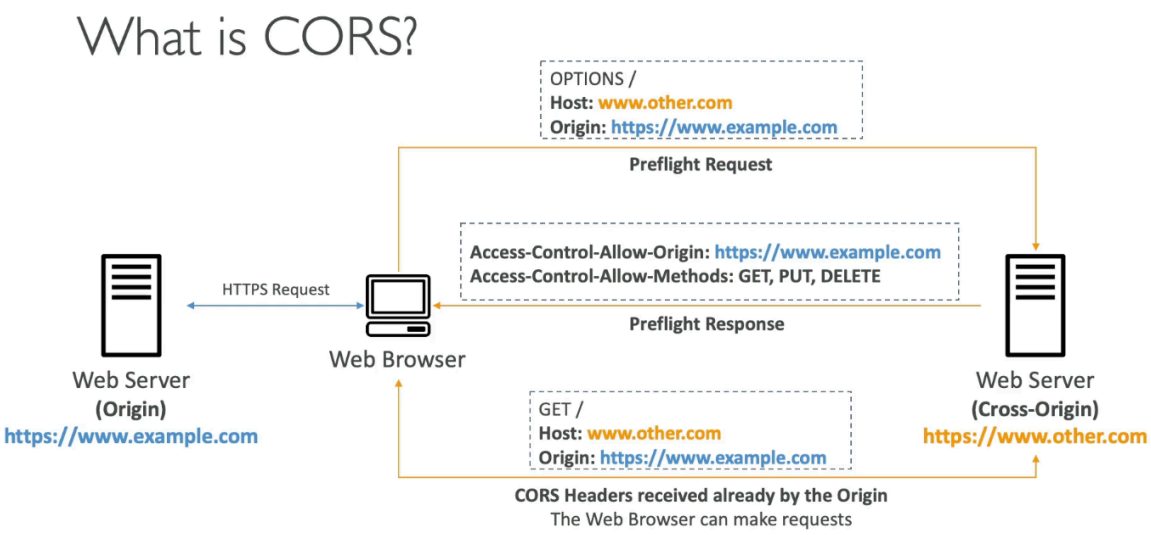
- aws:SecureTransport

S3 Default Encryption vs Bucket Policies



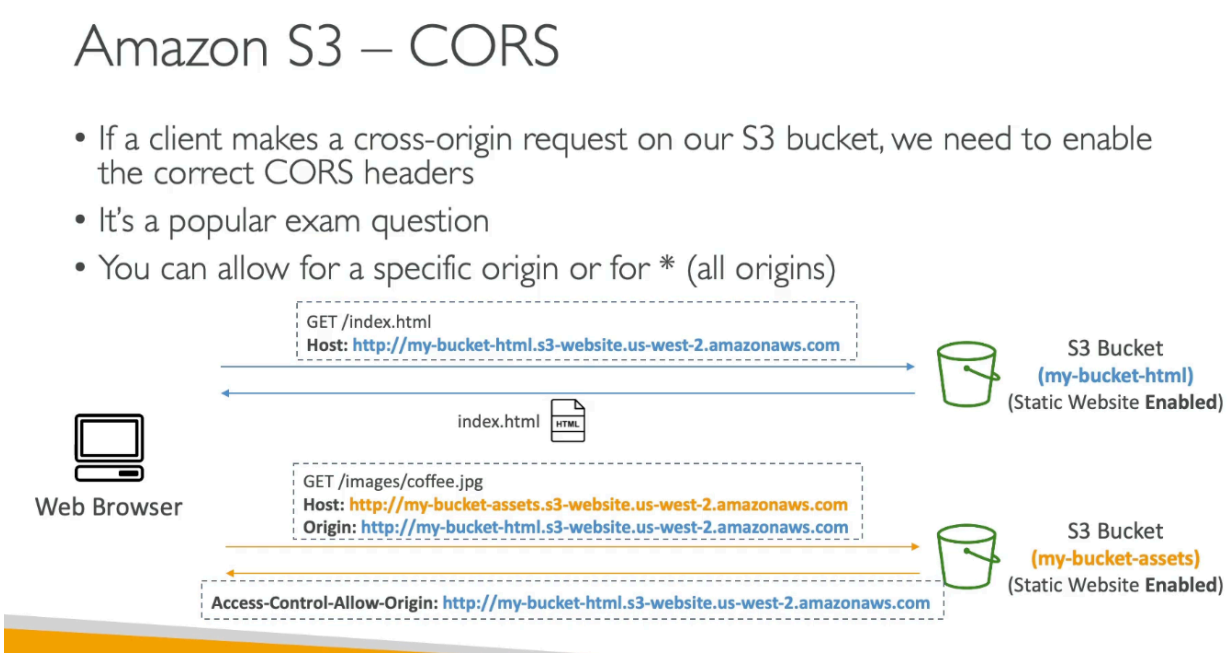
- SSE-S3 set by default, but a "force encryption" can be done using bucket policy to refuse any API call without encryption headers
- Bucket policies evaluated before default encryption

S3 CORS



- Origin = scheme (protocol) + host (domain) + port
  - https://example.com (implied port is 443 for HTTPS, 80 for HTTP)
- Web browser based mechanism to allow requests to other origins while visiting the main origin

    - Same origin: http://example.com/app & http://example.com/app2
    - Different origin: http://www.example.com & http://other.example.com
- Requests won't be fulfilled unless the other origin allows for the requests, using CORS headers



- If a client makes a CORS request on S3 bucket, we need to enable the correct CORS headers
    - Can allow specific origin or * for all origins

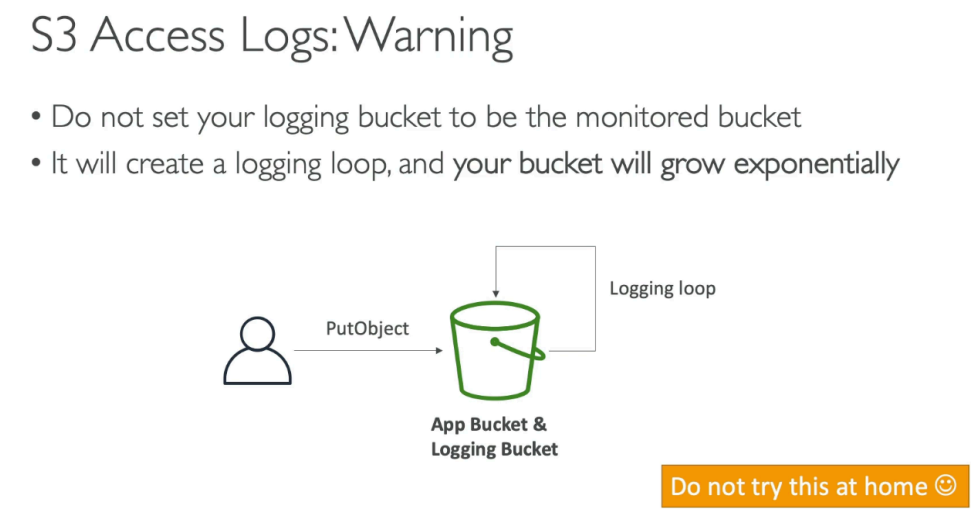
S3 MFA Delete

- MFA will be required to:
    - Permanently delete an object version
    - Suspend versioning on bucket
- MFA won't be required to:
    - Enable versioning
    - List deleted versions
- To use MFA delete, versioning must be enabled on the bucket
- Only bucket owner (root account) can enable / disable MFA delete
    - Only done via CLI, not UI

S3 Access Logs

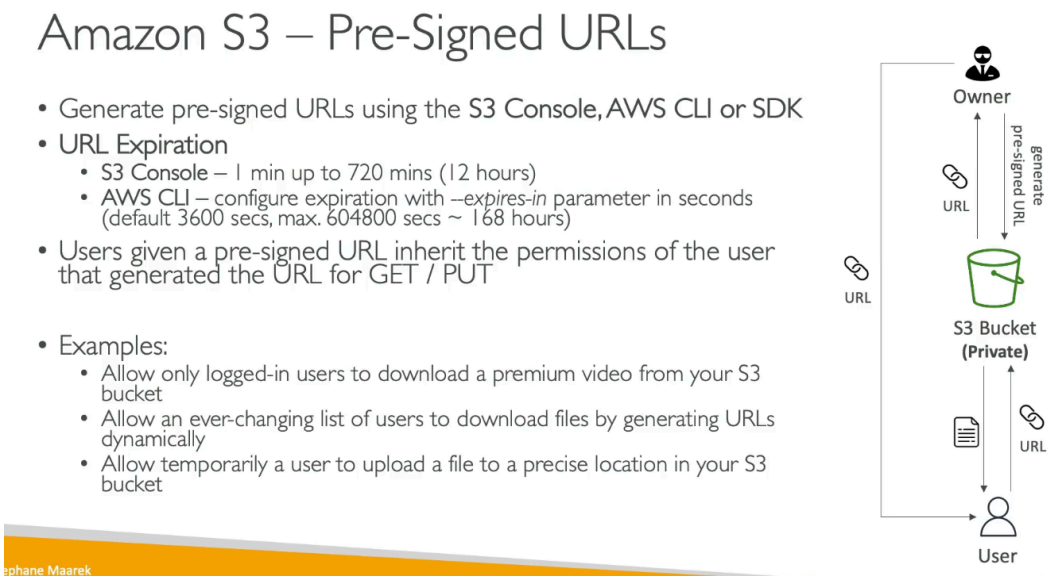
- For audit purposes, log all access to S3 buckets
- Any request made to S3 from any account will be logged to another S3 bucket in the same AWS region

Warning



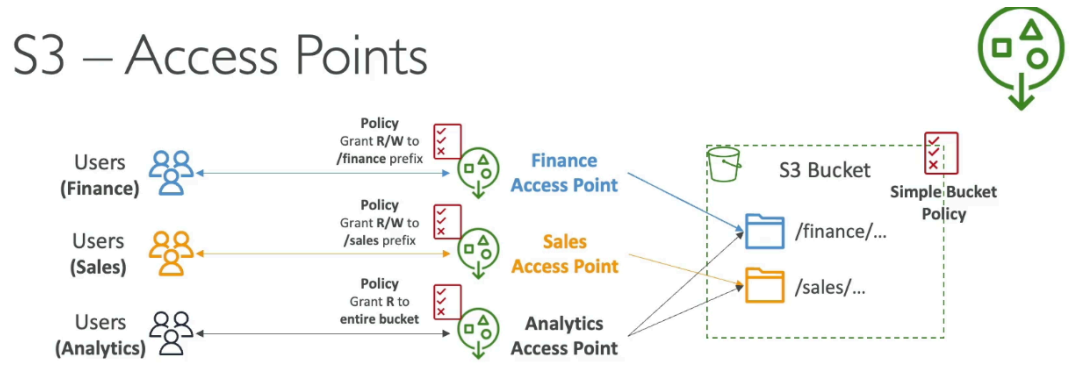
- Don't set logging bucket to be monitored bucket, creates a logging loop and bucket grows exponentially

S3 Pre-signed URLs



- Users given a pre-signed URL inherit the permissions of the user that generated the URL for GET / PUT
    - Temporary access for specific files for upload or download
- Generate via Console, CLI, SDK
    - Console: 1 min to 12 hours
    - CLI: configure expiration

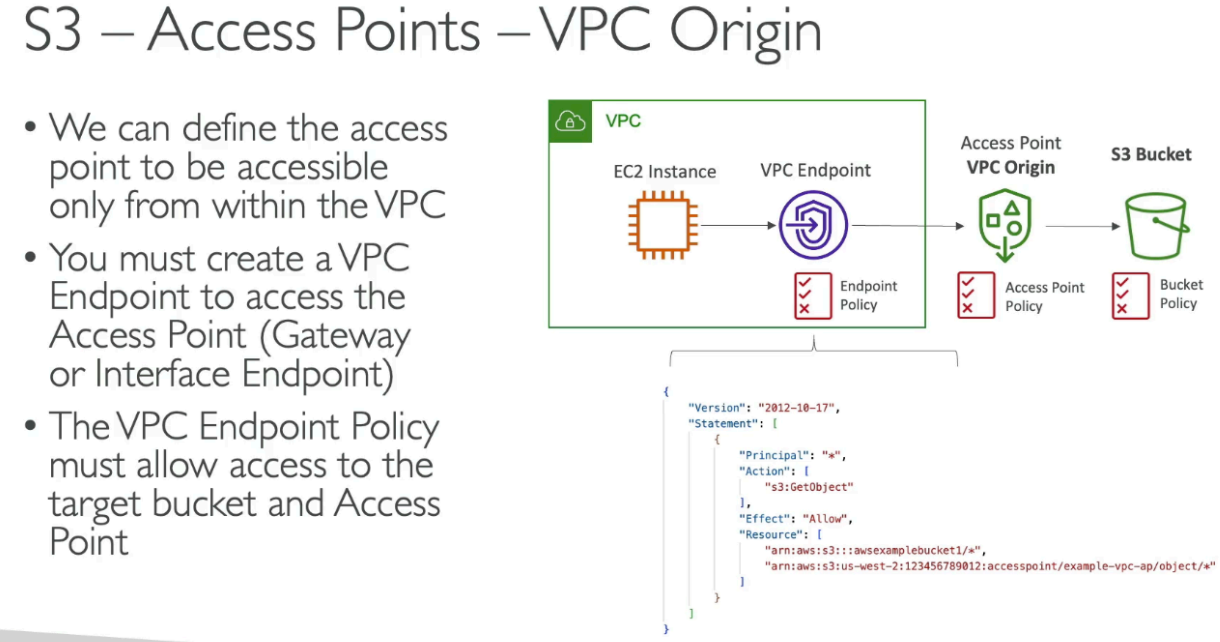
S3 Access Points



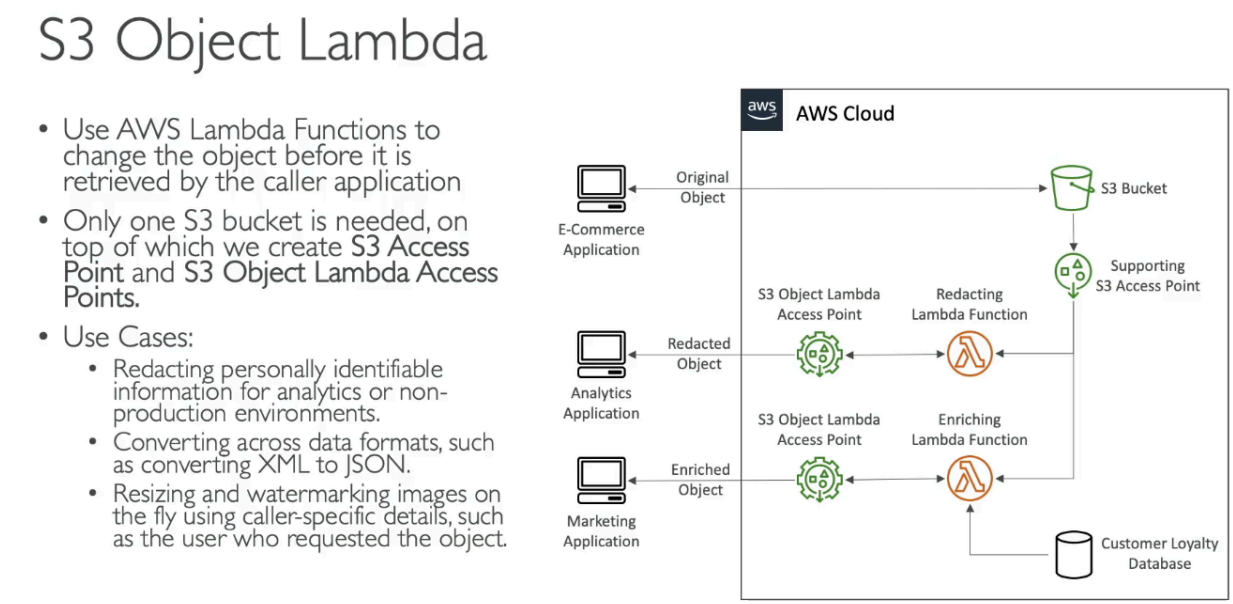
- Access Points simplify security management for S3 Buckets
- Each Access Point has:
  - its own DNS name (Internet Origin or VPC Origin)
  - an access point policy (similar to bucket policy) – manage security at scale

- Separate policies to allow access to specific prefix
- Simple bucket policy
- Each access point has:
  - Own DNS name (internet origin or VPC origin)
  - Access point policy (similar to bucket policy) - manage security at scale

Access Points - VPC Origin



S3 Object Lambda



- Use Lambda functions to change the object before it is retrieved by the caller application
- Only 1 bucket is needed, on top of S3 Access point and S3 Object Lambda Access Point
- Use cases: redact PI data, converting across data formats, etc…

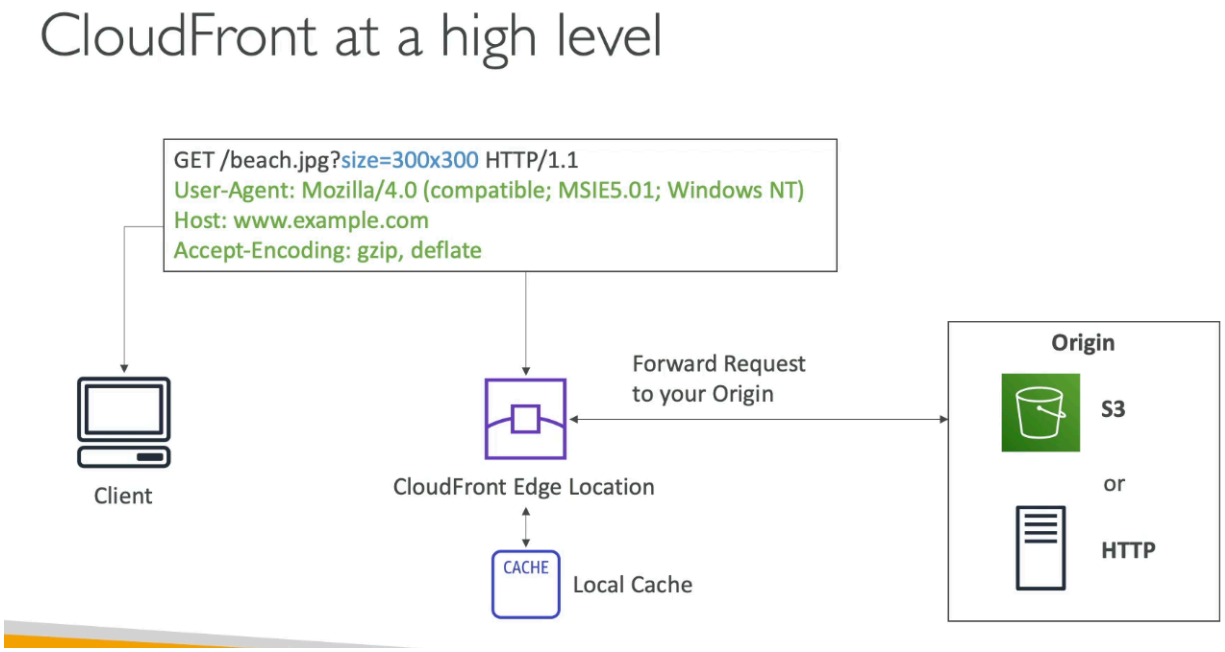
Section 15: CloudFront

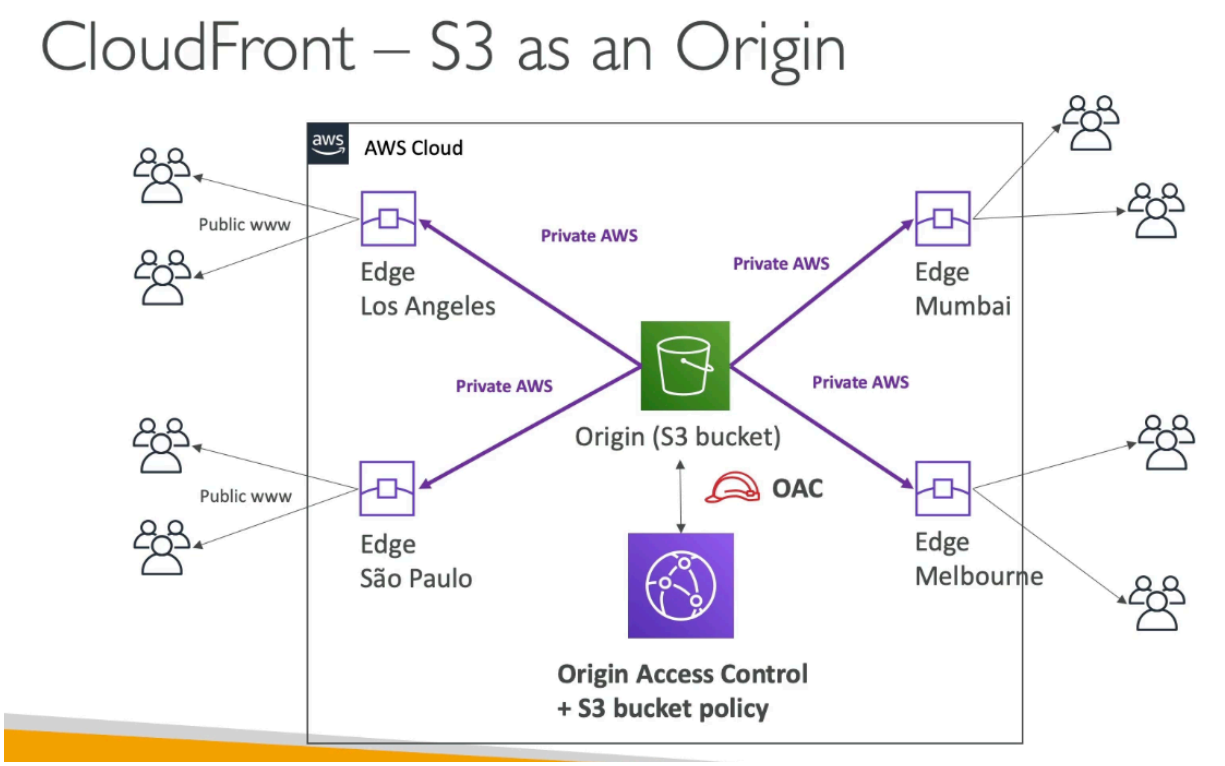
CloudFront Overview

- Content Delivery Network (CDN)
- Improves read performance, content is cached at the edge to improve user experience
    - 216 edge locations (point of presence)
- DDoS protection, integration with Shield and WAF

Origins

- S3 Bucket
    - For distributing files and caching at the edge
    - Enhanced security with CloudFront Origin Access Control (OAC)
    - OAC replaces Origin Access Identity (OAI)
    - CloudFront and be used as an ingress (upload files to S3)
- Custom Origin (HTTP)
    - ALB, EC2, S3 website, any HTTP backend



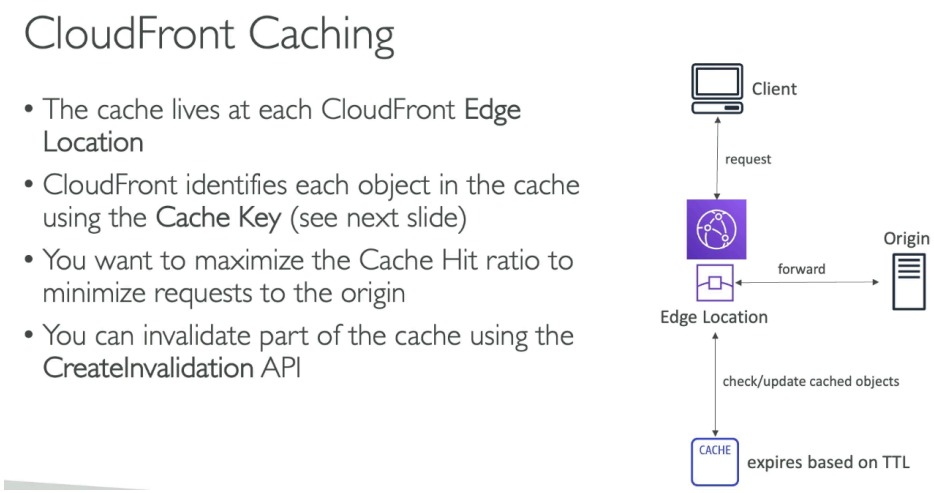


CloudFront vs S3 Cross Region Replication

- CloudFront:
  - Global Edge network
  - Files are cached for TTL
  - Great for static content that must be available everywhere

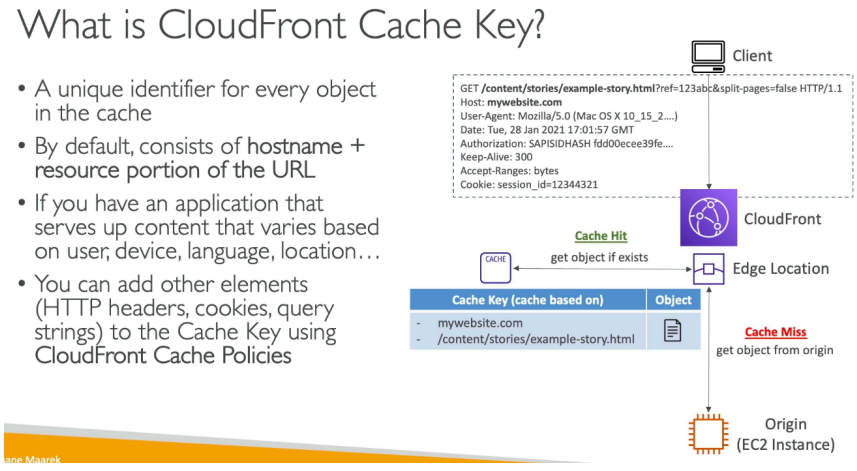
- S3 Cross Region Replication
    - Must be setup for each region you want replication
    - Files updated near real-time
    - Read only
    - Great for dynamic content that needs to be available at low-latency in a few regions

CloudFront - Caching & Caching Policies



- Cache lives at edge locations
- CloudFront identifies each object in the cache using Cache Key
- Maximize cache hit ratio to minimize requests to origin
- Can invalidate part of the cache using CreateInvalidation API

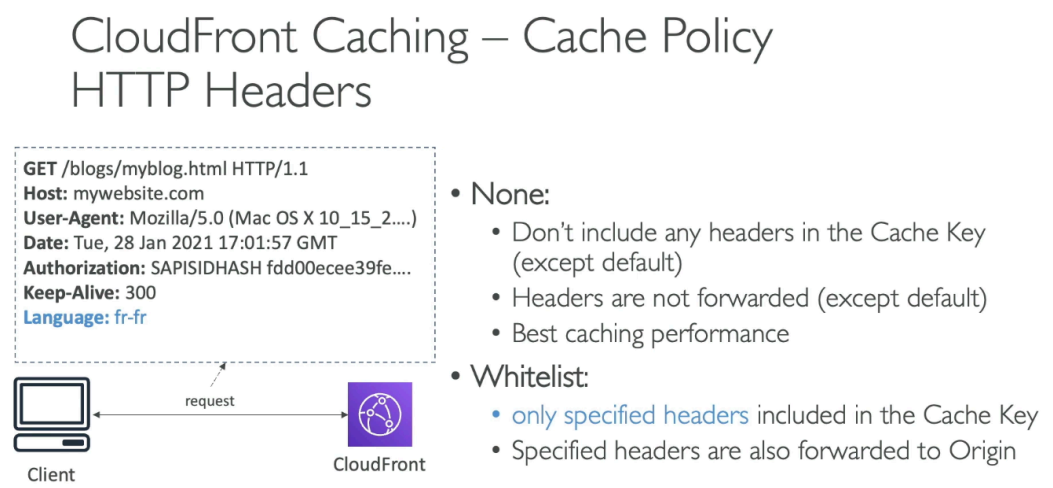
What is CloudFront Cache Key?

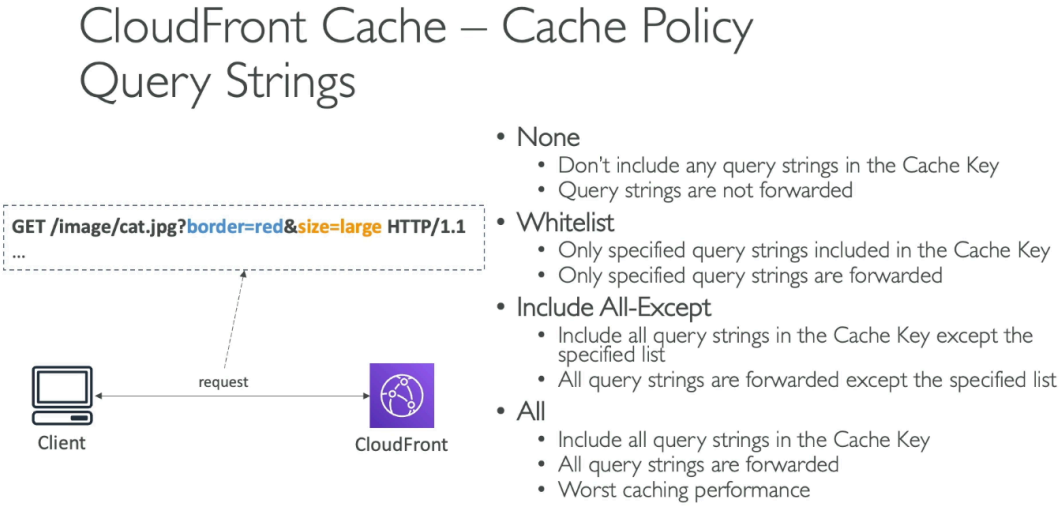


- Unique identifier for every object in the cache

- By default, consists of hostname + resource portion of URL
    - Other elements HTTP headers, cookies, etc… to the cache key using CloudFront Cache Policy

Cache Policy

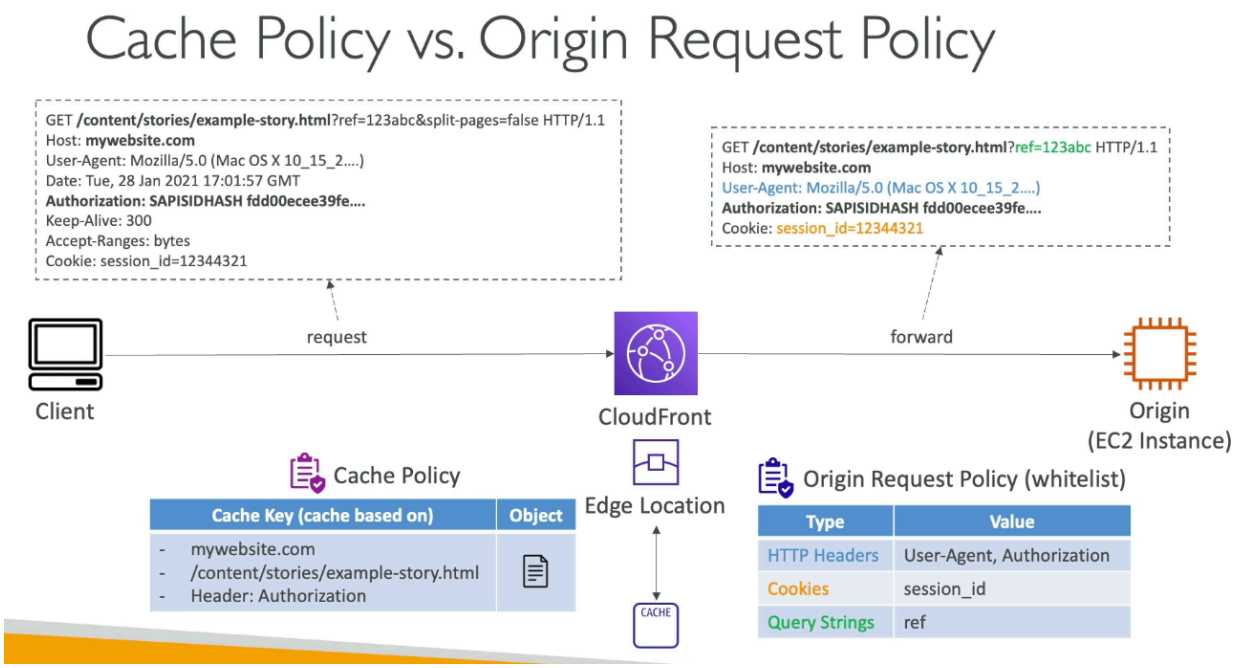




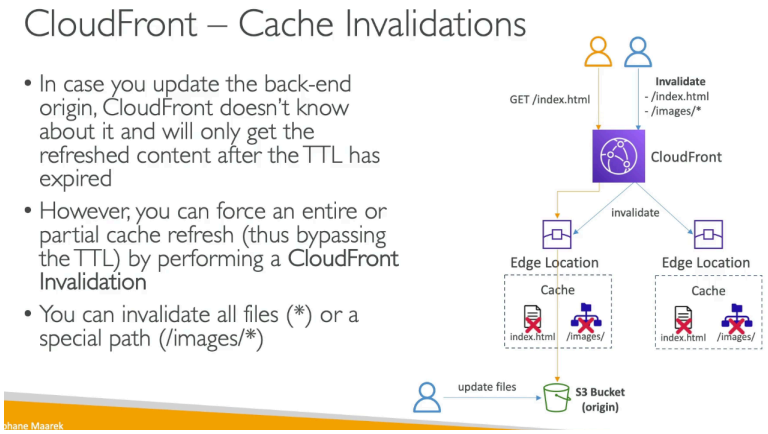
- Cache based on:
    - HTTP Headers: none - whitelist
    - Cookies: none, whitelist, include all except, all
    - Query String: none, whitelist, include all except, all
- Control the TTL, can be set by origin using the cache control header, expires header…
- Create own policy or predefined policies
- All HTTP headers, cookies, and query strings that you include in cache key are automatically included in origin requests

Origin Request Policy

- Specify values want to include in origin request without including in Cache Key (no duplicated cached content)
    - HTTP Headers: none, whitelist, all viewer headers options
    - Cookies: none, whitelist, all
    - Query strings: none, whitelist, all
- Ability to add CloudFront HTTP headers and custom headers to origin request that are not included in viewer request
- Can create own or use predefined policy

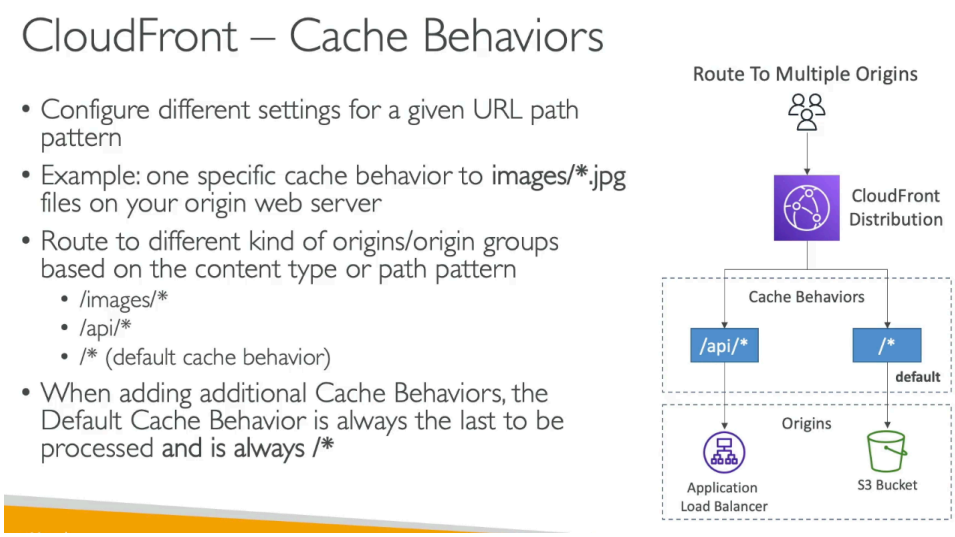


Cache Invalidations

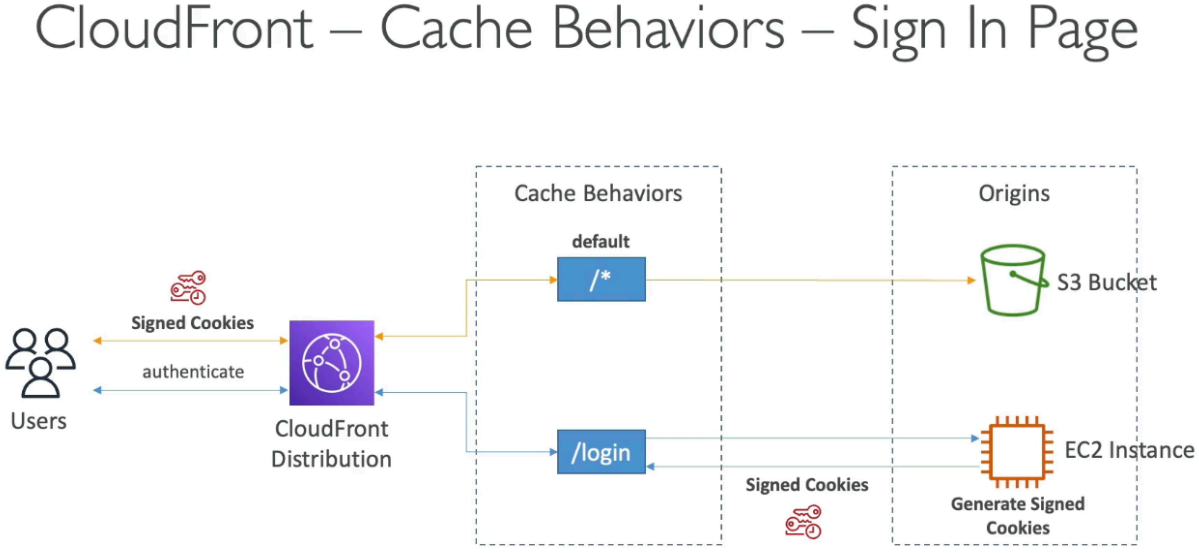


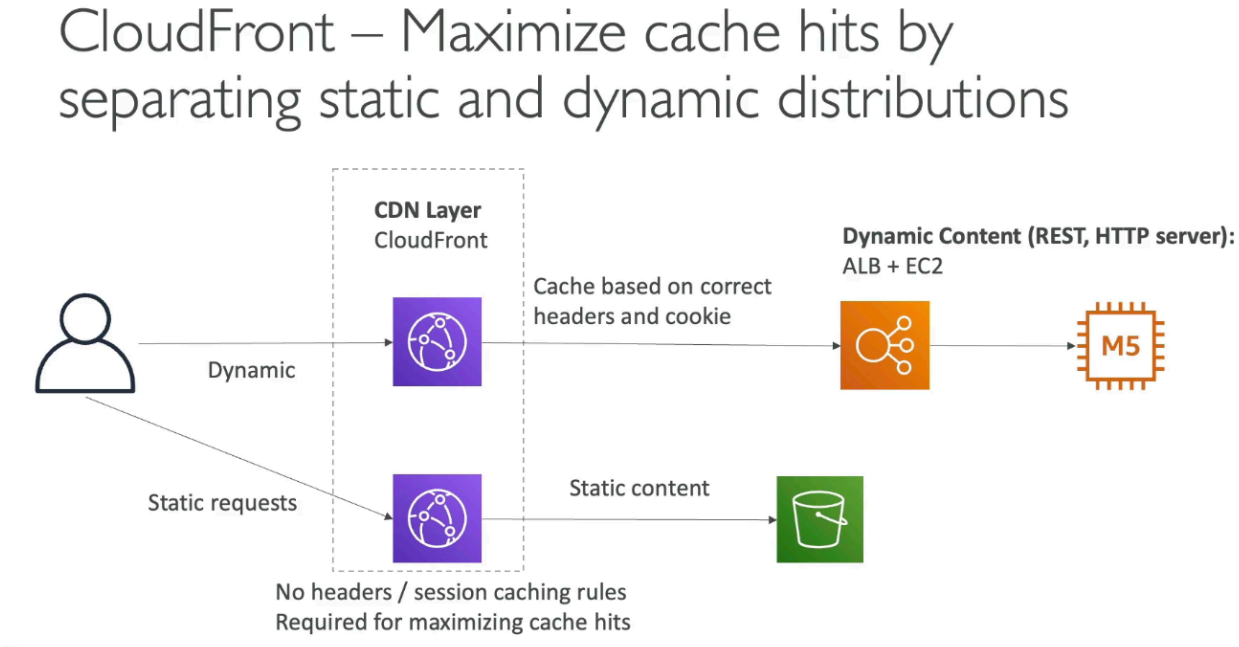
- In case you update the backend origin, CloudFront doesn't know and will only get refreshed content after TTL expired. Force entire or partial cache refresh (bypasses TTL) via CloudFront Invalidation

Cache Behaviors

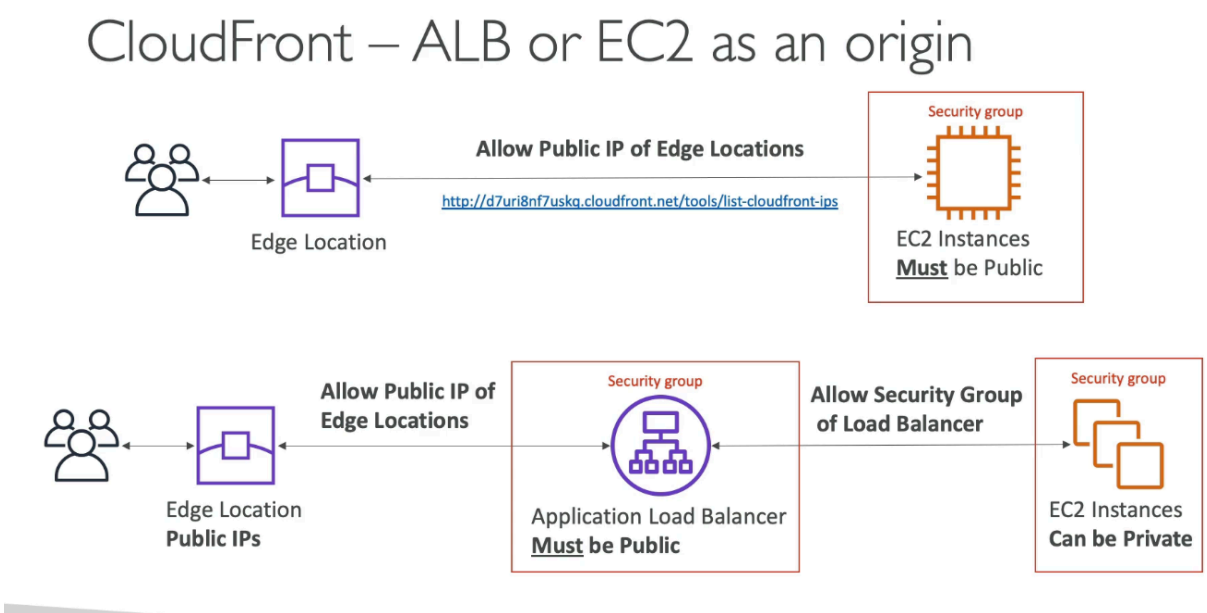


- Configure different settings for given URL path pattern
- Route to different kind of origins/origin groups based on the content type or path pattern
- When adding additional cache behavior, the default cache behavior is always the last processed and is always /*





ALB as an Origin

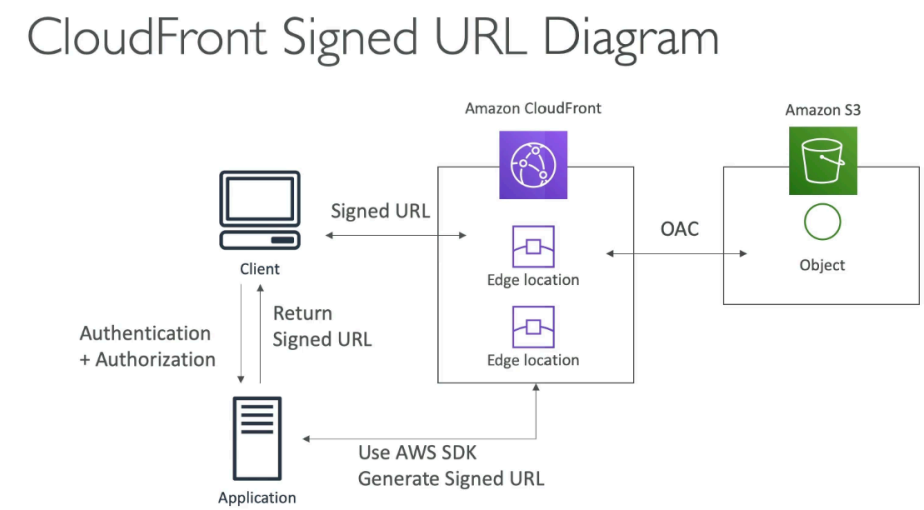


Geo Restriction

- Restrict who can access distribution

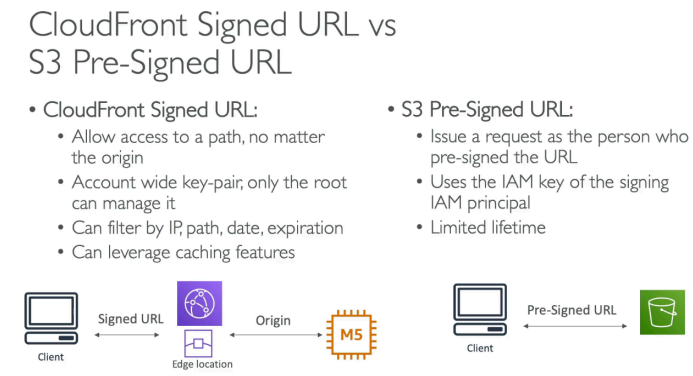
    - Allowlist: allow users to access content only if they're on a list of approved countries
    - Blocklist: prevent users from accessing content if they're on a banned country list
- Country determined via 3rd party Geo-IP database

Signed URL / Signed Cookies



- Must attach policy with:
    - URL expiration
    - IP ranges to access data
    - Trusted signers (which AWS accounts can create signed URLs)
- How long should the URL be valid for?
    - Shared content: short
    - Private content: years
- Signed URL = access to individual files (1 signed URL per file)
- Signed Cookies = access to multiple files (one signed cookie for many files)

Signed URL vs S3 Pre-signed URL



- CloudFront Signed URL
    - Allow access to path, no matter the origin
    - Account wide key pair, not only the root can manage
    - Can filter by IP, path, date, expiration
    - Can leverage caching features
- S3 Pre-Signed URL
    - Issue a request as person who pre-signed URL
    - Uses IAM key of the signing IAM principal
    - Limited Lifetime

Signed URL Process

- 2 types of signers
    - Trusted key group
        - Can leverage APIs to create and rotate keys (and IAM for API security)
    - AWS account that contains a CloudFront Key Pair
        - Need to manage keys using root account and console (not recommended)
- In CloudFront distribution, create 1+ trusted key groups
- Generate your own public / private key
    - Private key used by applications to sign URLs
    - Public key (uploaded) is used by CloudFront to verify URLs

CloudFront Advanced Concepts

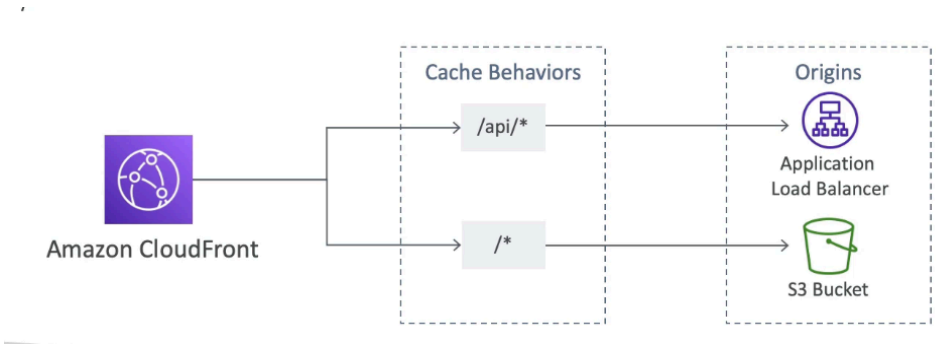
Pricing

- Cost of data out per edge location varies based on location of edge location

Price Classes

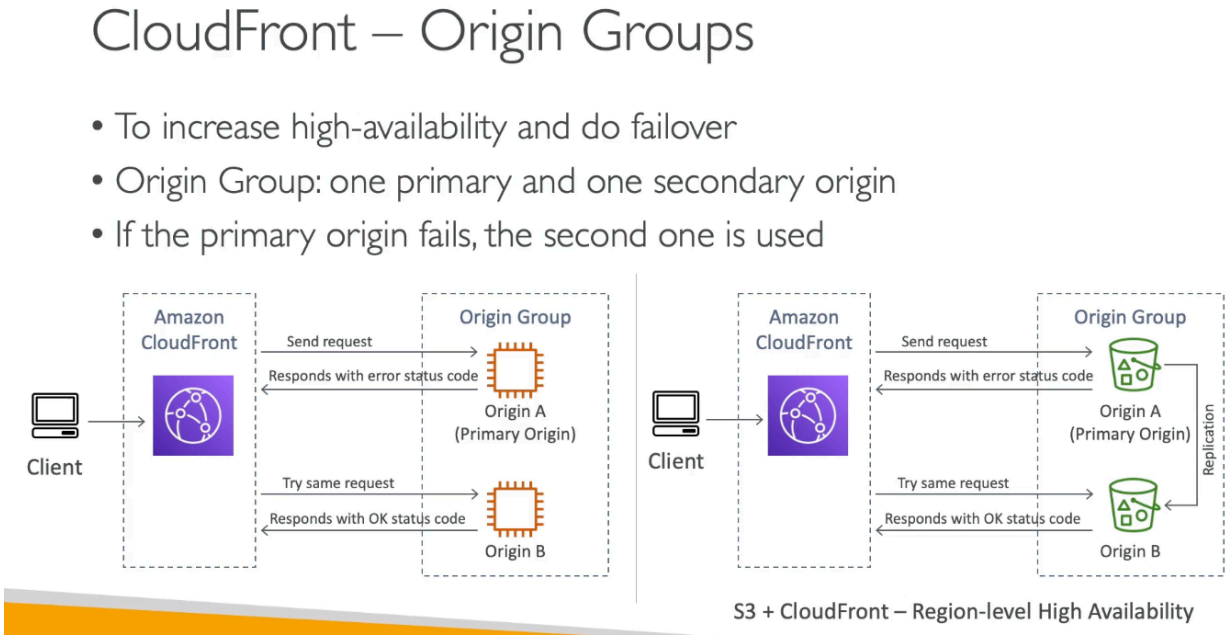
- Can reduce the number of edge locations for cost reduction
- 3 price classes:
    1. Price Class All: all regions, best performance
    2. Price Class 200: most regions, excludes most expensive regions
    3. Price Class 300: only the least expensive regions

Multiple Origin



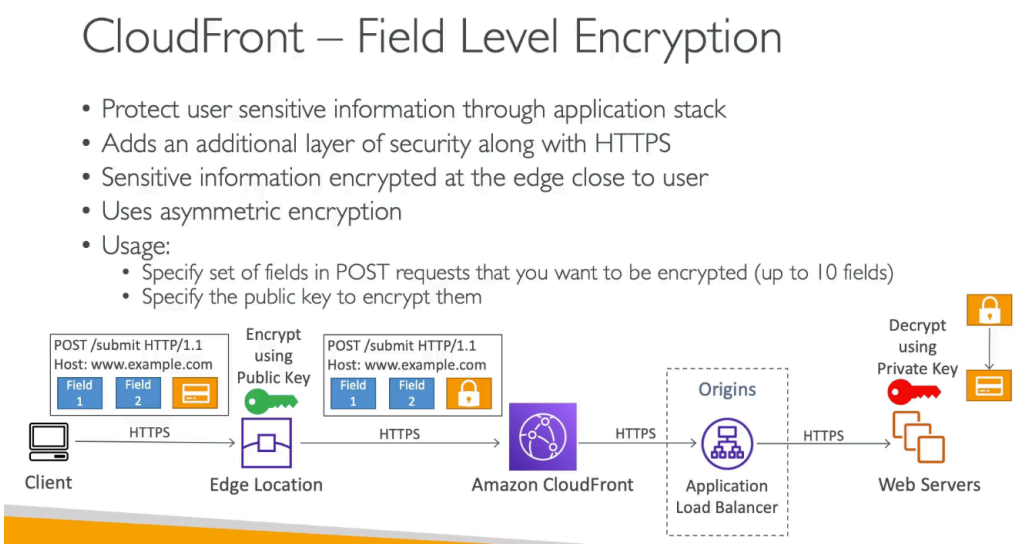
- To route to different kind of origins based on content type based on path pattern

Origin Groups



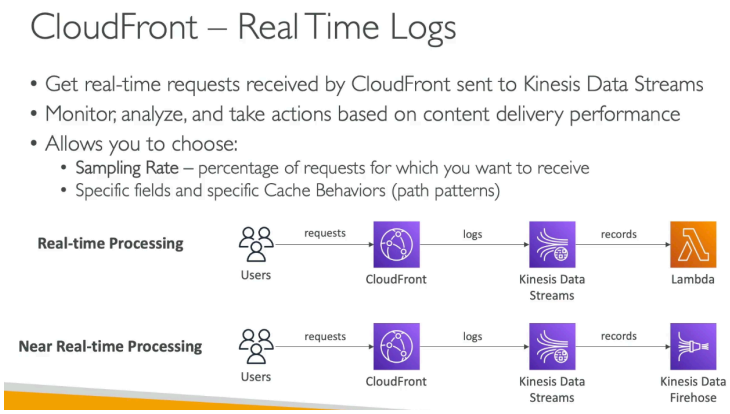
- Increase high availability and do failover
- Origin group: one primary and one secondary
- If the primary origin fails, second is used

Field Level Encryption



- Protect user sensitive info through application stack
- Additional layer along with HTTPS
- Sensitive information encrypted at the edge close to user
- Uses asymmetric encryption
- Usage:
    - Specify set of fields in POST you want encrypted (up to 10 fields)
    - Specify the public key to encrypt

Real Time Logs



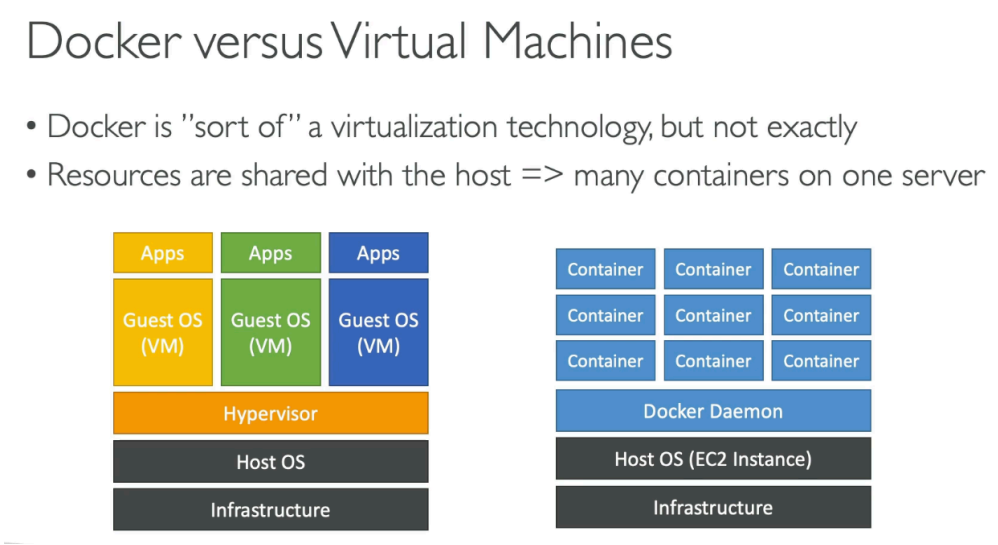
- Real time requests received by CloudFront sent to Kinesis Data stream
    - Monitor, analyze based on content delivery performance
- Choose:
    - Sampling rate: % of requests you want to receive
    - Specific fields and specific cache behaviors (path patterns)

Section 16: ECS, ECR & Fargate - Docker in AWS

Docker Introduction

- Software development to deploy apps
- Apps are packaged in containers to run on any OS (predictable behavior)
- Docker images are stored in Docker repos @ Docker Hub or Amazon ECR

Docker vs VM



- Docker is “sort of” a virtualization tech, but not exactly
- Resources shared with the host → many containers on 1 server

Docker Container Management on AWS

- ECS, EKS, Fargate, ECR

Amazon ECS - EC2 Launch Type

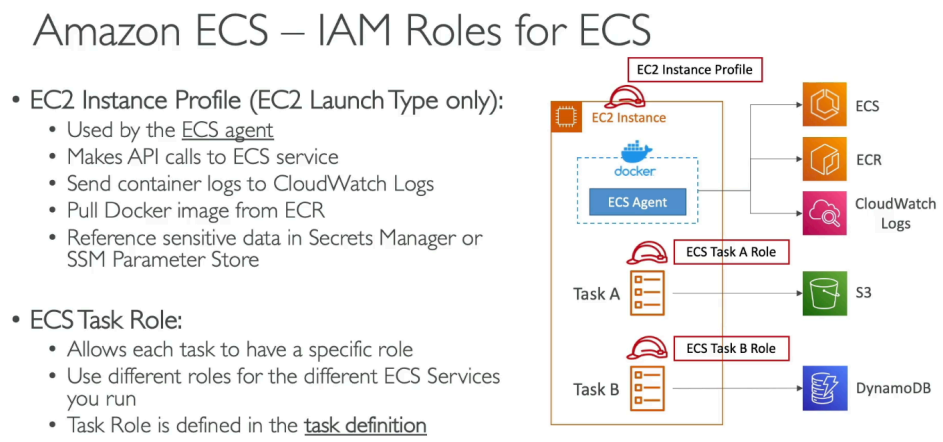
- Launch Docker containers on AWS = launch ECS tasks on ECS clusters
- EC2 launch type: must provision & maintain the infrastructure (EC2 instances)
    - Each EC2 instance must run the ECS Agent to register in the ECS Cluster
    - AWS takes care of starting / stopping containers

Fargate Launch Type

- Launch Docker containers on AWS
- Do not provision infrastructure (serverless)

- Create task definitions, AWS just runs ECS tasks based on CPU / RAM needed - scaling to increase number of tasks

IAM Roles for ECS



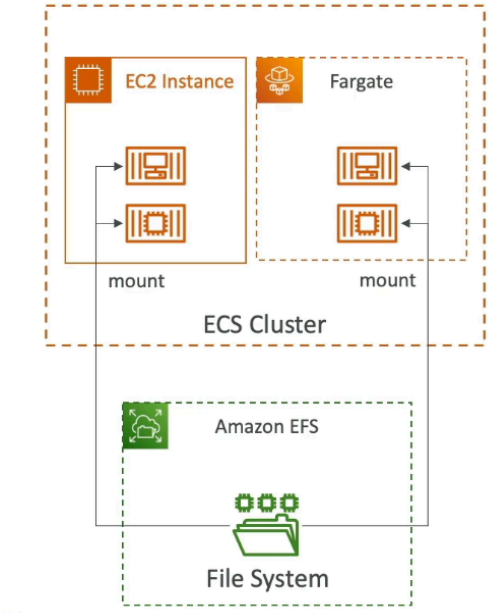
- EC2 instance profile (EC2 launch type only)
    - Used by ECS agent and makes API calls to ECS service
        - Can send container logs to CloudWatch, pull Docker image from ECR, reference data in Secrets Manager
- ECS Task Role
    - Allows each task to have a specific role
    - Use different roles for different ECS services you run
    - Task role defined in task definition

Load Balancer Integrations

- ALB supported and works for most cases
- NLB recommended for high throughput / high performance or with AWS Private Link
- Classic LB not recommended

ECS Data Volumes (EFS)

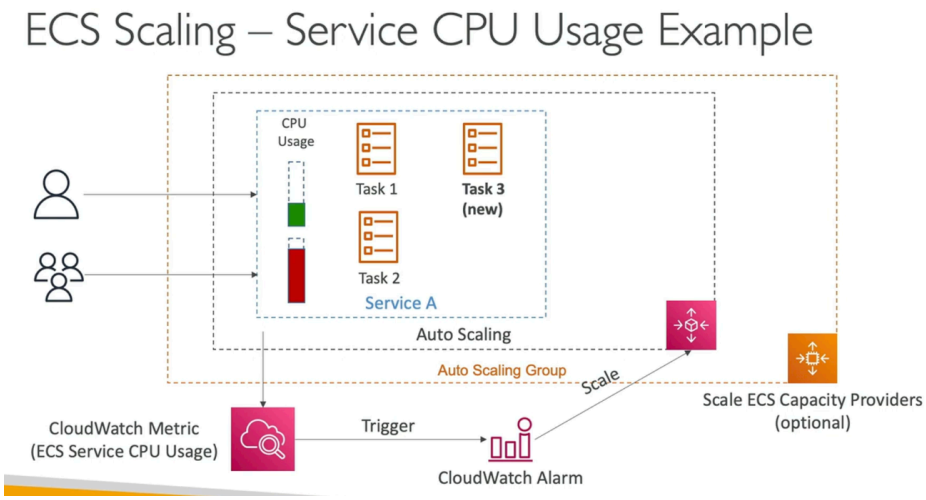
- Mount EFS file system onto ECS tasks
- Works for both EC2 and Fargate
- Tasks running in any AZ will share same data in EFS
- Fargate + EFS = serverless
- Use Case: persistent multi-AZ shared storage for containers
- S3 cannot be mounted as a file system



ECS Service Auto Scaling

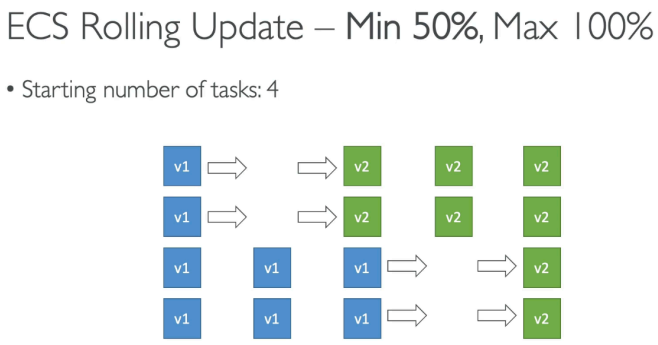
- Automatically increase / decrease ECS tasks via Application Auto Scaling
    - Average CPU, memory utilization, or ALB request count
- Target Tracking: scale based on specific CloudWatch metric
- Step scaling: scale based on specific CloudWatch alarm
- Scheduled Scaling: scale based on time/date
- ECS Service Auto Scaling (task level) != EC2 Auto Scaling (EC2 instance level)
- Fargate Auto scaling easier to set up

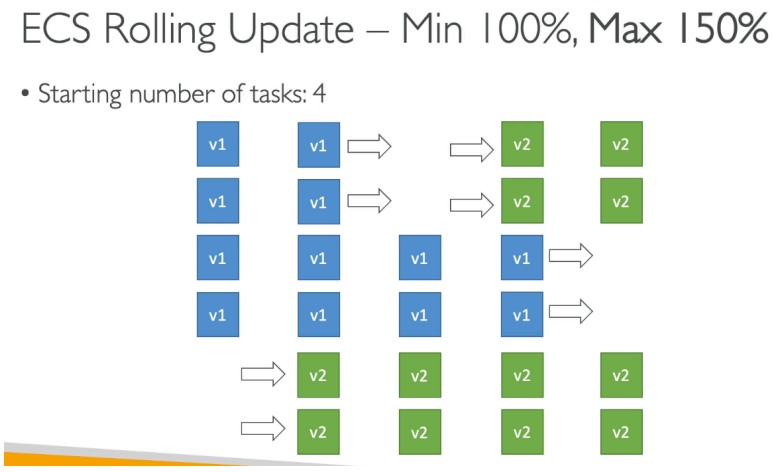
Auto Scaling EC2 Instances for EC2 Launch Type



- Auto Scaling Group Scaling
    - Scale ASG based on utilization
- ECS Cluster Capacity Provider
    - Used to automatically provision and scale infrastructure for ECS tasks
    - Paired with ASG to add EC2 instances when missing capacity

ECS Rolling Updates

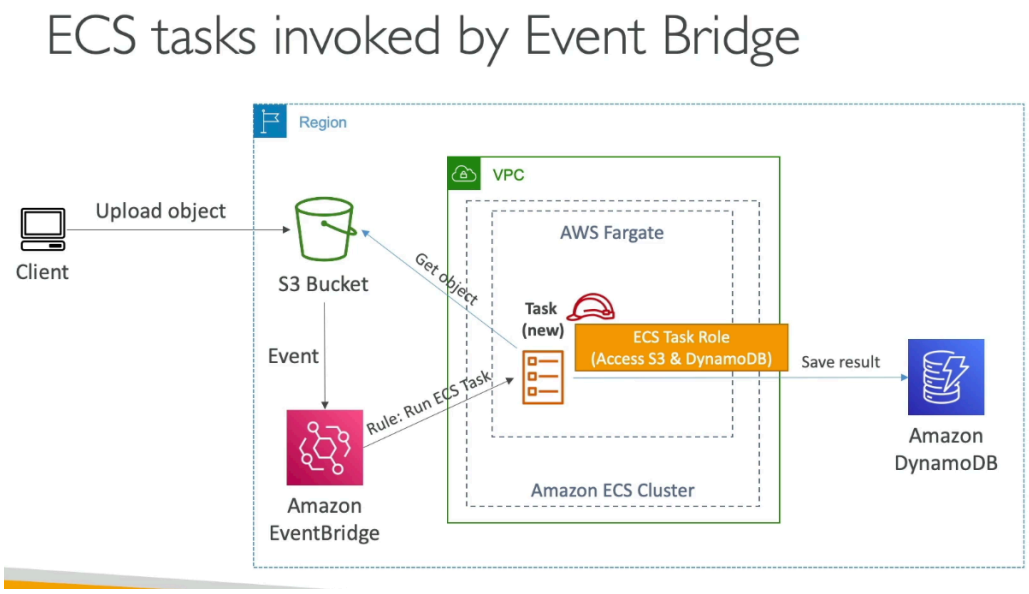




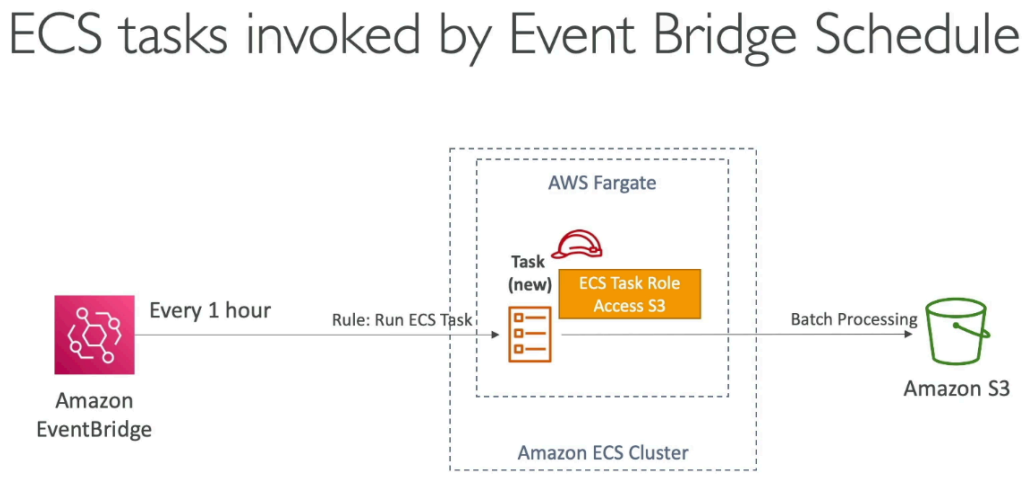
- When updating from v1 to v2, control how many tasks can be started and stopped and in which order

ECS Solutions Architecture

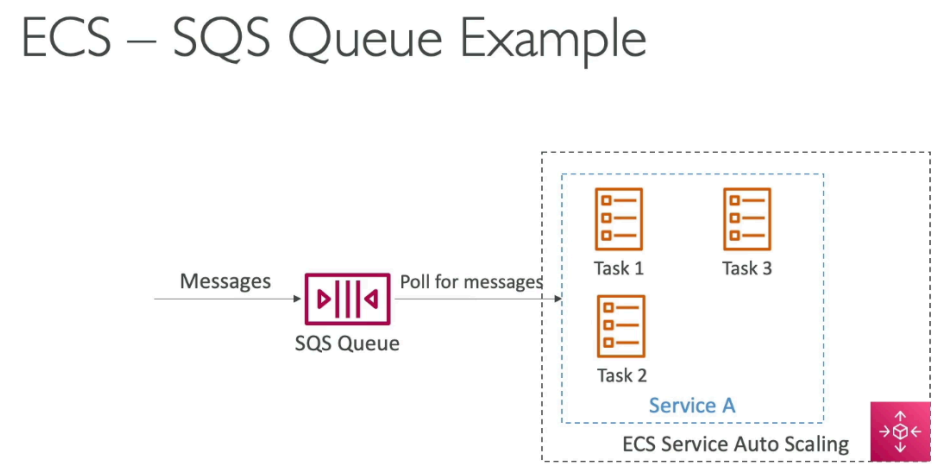
ECS Tasks invoked by Event Bridge



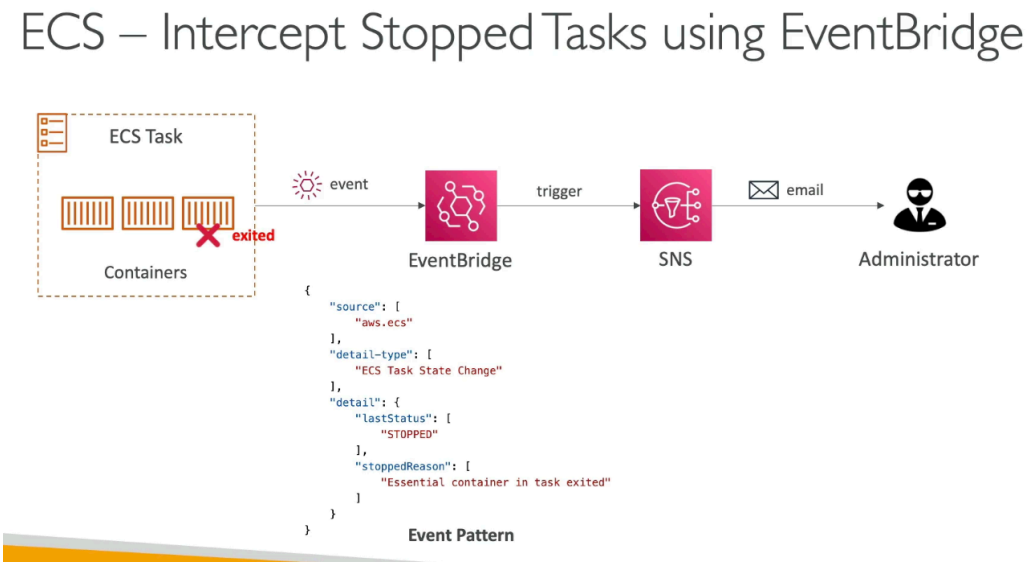
ECS Tasks Invoked by Event Bridge Schedule



ECS- SQS Queue Example

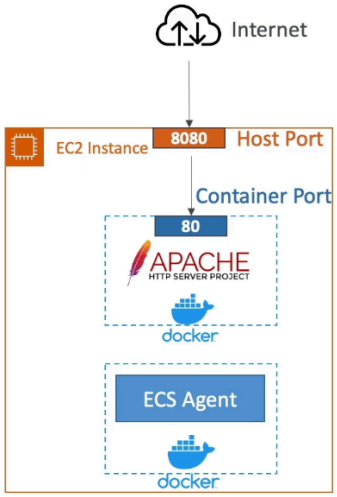


ECS – Intercept Stopped Tasks using EventBridge

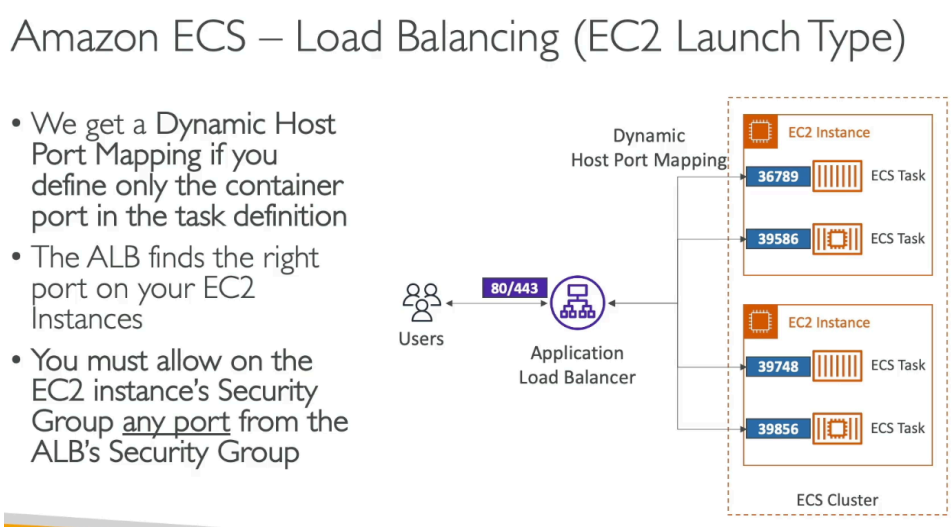


ECS – Task Definitions

- Metadata in JSON to tell ECS how to run a Docker container
  - Image name, port binding for container and host, memory, CPU, env vars, Network info, IAM Role, logging configuration
- Can define up to 10 containers in task definition

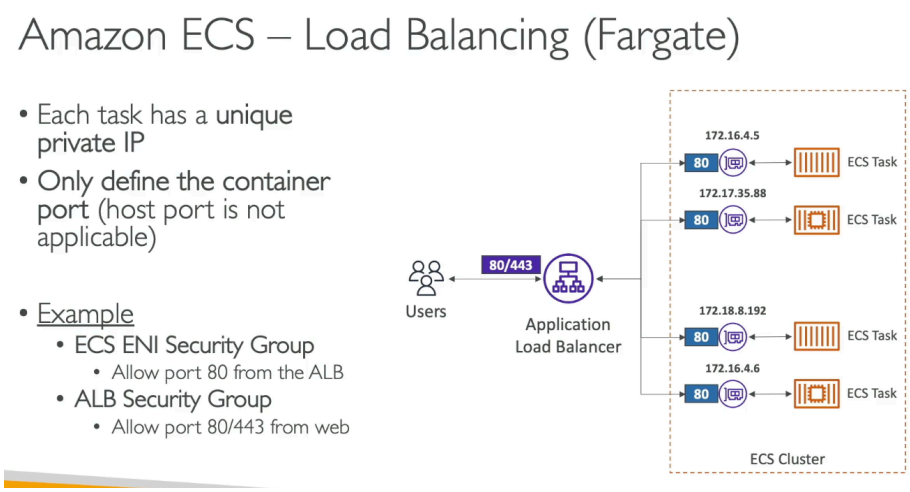


ECS – Load Balancing (EC2 launch type)



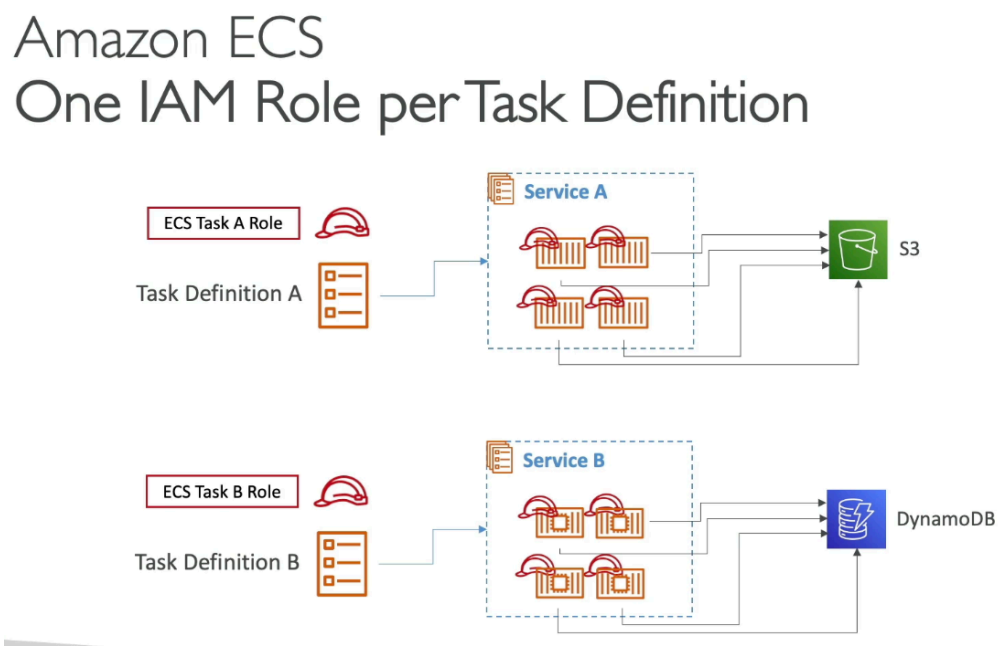
- We get a dynamic host port mapping if you define only the container port in the task definition
    - ALB finds the right port on EC2 instances via dynamic host port mapping
- Must allow on EC2 instance's security group any port from ALB's security group

Load Balancing (Fargate)



- Each task has a unique private IP
- Only define container port (host port not applicable)
- Example:
    - ECS ENI Security Group: allow port 80 from ALB
    - ALB SG: Allow port 80/443 from web

One IAM Role per Task Definition



- Where do you define an IAM role for an ECS task? On task definition. Services will inherit from task

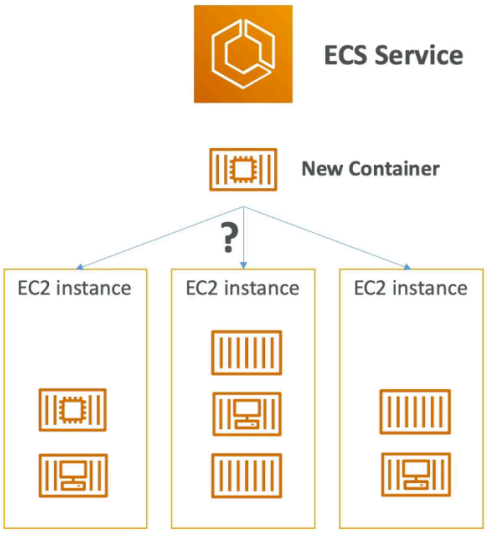
Environment Variables

- Hardcoded, SSM Parameter store, Secrets Manager
- Environment files (bulk) via S3

Data Volumes (Bind Mounts)

- Share data between containers in same task definition (works for both EC2 and Fargate tasks)
- EC2 tasks – using EC2 instance storage
    - Data are tired to EC2 instance lifecycle
- Fargate tasks – using ephemeral storage
    - Data tied to container using them
    - 20 GB to 200 GB (default is 20)
- Use cases:
    - Share ephemeral data between multiple containers
    - “Sidecar” container pattern where “sidecar” container used to send metrics / logs to other destinations

Task Placements



- When a task of type EC2 is launched, ECS must determine where to place it, with the constraints ot CPU, memory, and available port
- Similarly when a service scales in, ECS needs to determine which task to terminate
- Define a task placement strategy and task placement constraints
- Only for ECS with EC2

Task Placement Process

- Based on best effort
- When ECS places tasks:
    1. Identify instances that satisfy CPU, memory, and port requirements in the task definition
    2. Identify instance that satisfy task placement constraints
    3. Identify instance that satisfy task placement strategies
    4. Select instance for task placement

- Binpack
    - Place tasks based on least available amount of CPU or memory
    - Minimizes the # of instances in use (cost saving)
    - Fills 1 instance completely before using another EC2 instance
- Random
    - Places task randomly
- Spread
    - Place the task evenly based on a specified value (ex: instance ID, AZ)
- Can mix placement strategies

ECS Task Placement Constraints

- distinctInstance: place each task on a different container instance
- memberOf: places task on instances that satisfy an expression
    - Uses cluster query language (advanced)

ECR

- Store and manage Docker images (public or private) on AWS
- Fully integrated with ECS, backed by S3
    - IAM Role to EC2 instance to pull images
- Supports image vulnerability scanning, versioning, image tags, image lifecycle

ECR – Using AWS CLI

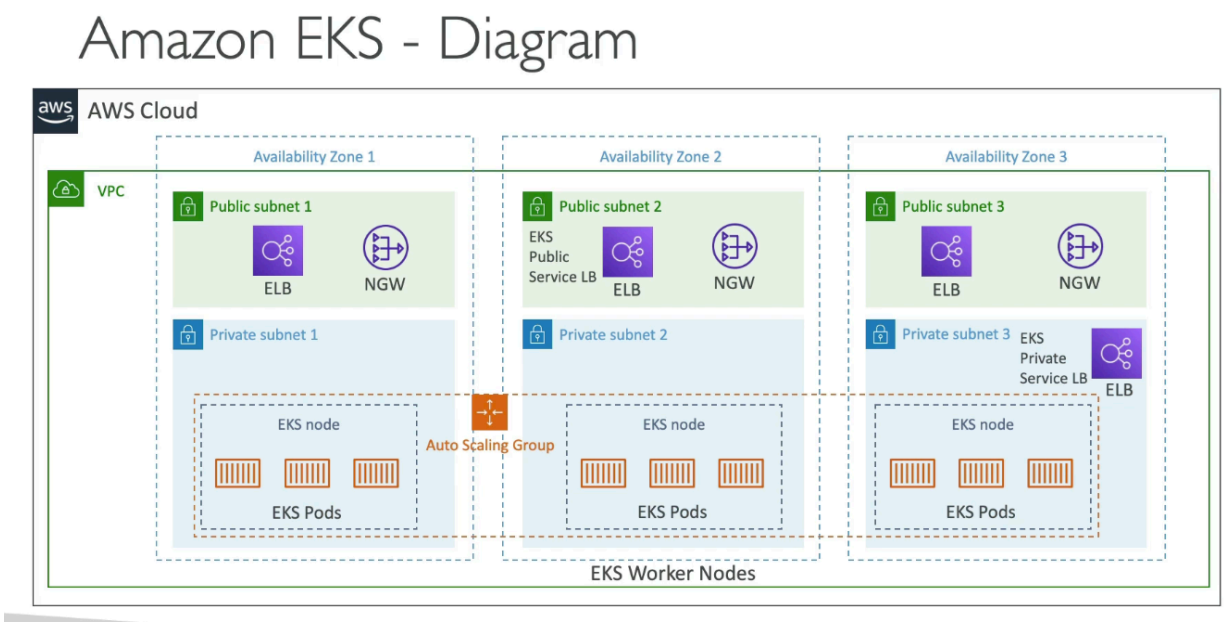
- Login command
    - Login command to ECR that passes it to docker command so Docker can pull images from ECR
- In case EC2 instance can't pull a docker image, check IAM permissions

AWS Copilot

- CLI tool to build, release, and operate prod ready containerized apps
- Runs apps on AppRunner, ECS, Fargate
- Helps focus on building app rather than infrastructure
- Provisions all required infrastructure for containerized apps
- Automated deployments via CodePipeline
- Deploy to multiple environments
- Troubleshooting, logs, health status

Amazon EKS

- Launches managed Kubernetes clusters on AWS
    - K8 is for automatic deployment, scaling and management of containerized (Docker) application
    - Alternative to ECS
    - Cloud-agnostic: can use in any cloud
- EKS supports EC2 if you want to deploy worker nodes or Fargate for serverless containers



EKS Node Types

- Managed Node Groups
  - Creates and manages Nodes (EC2 instances) for you
  - Part of ASG managed by EKS
  - Supports on demand or spot instances
- Self Managed Nodes
  - Nodes created by you and registered to EKS cluster and managed by ASG
  - Can use prebuilt AMI – EKS Optimized AMI
  - Supports on demand or spot instances
- AWS Fargate
  - No maintenance, no managed nodes

EKS – Data Volumes

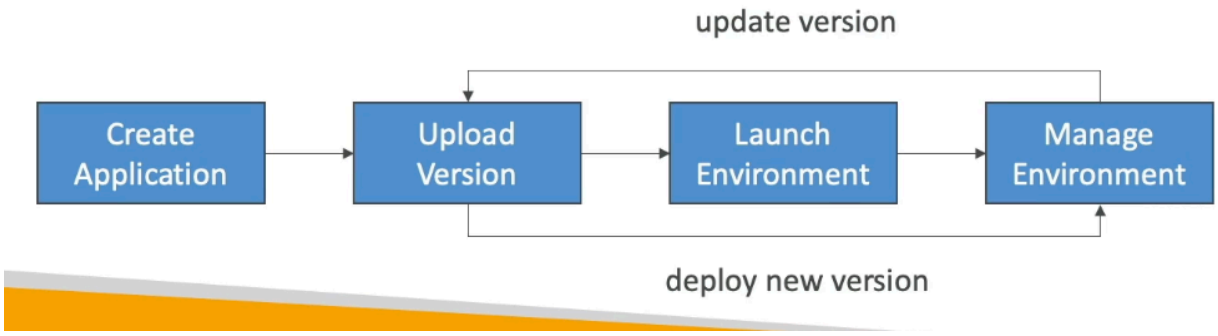
- Need to specify StorageClass manifest on EKS cluster
- Leverages a Container Storage Interface (CSI) compliant driver
- Support for:
  - EBS, EFS (works with Fargate), FSx for Lustre, FSx for NetApp ONTAP

Section 17: AWS Elastic Beanstalk

Elastic Beanstalk High Level Overview

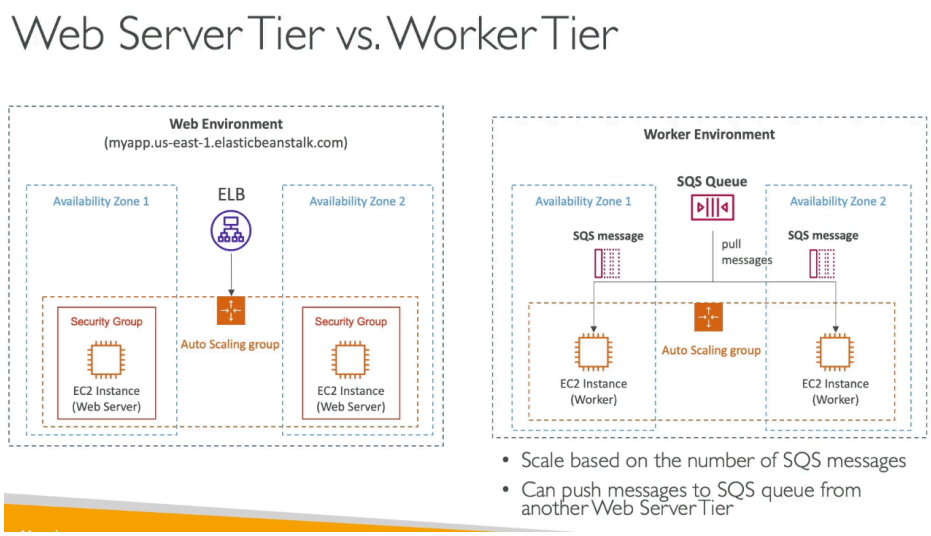
- Developer centric view of deploying apps on AWS
- Managed service
    - Automatically handles capacity provisioning, LB, scaling, application health
    - Application code is developer responsibility
- Full control over configuration
- Free, but paid for underlying services

Beanstalk Components

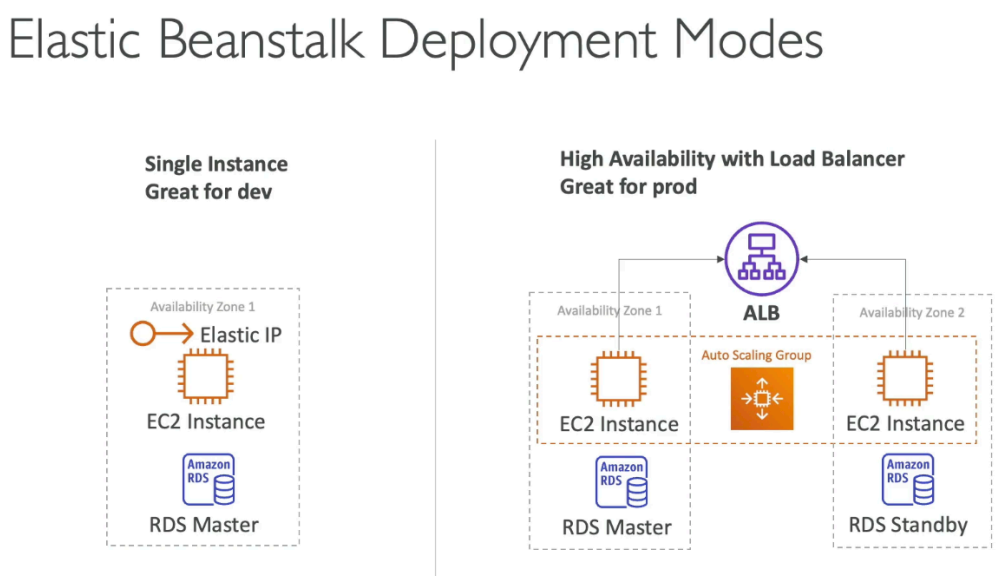


- Application: collection of Elastic Beanstalk components (environments, versions, configurations)
- Application version, environment (tiers)
- Supports many platforms

Web Server Tier vs Worker Tier



Deployment Modes

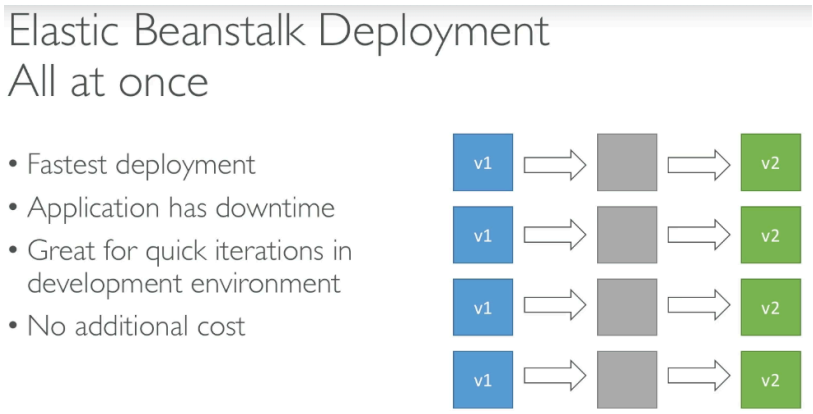


- Single Instance (for dev) or high availability with LB (prod)

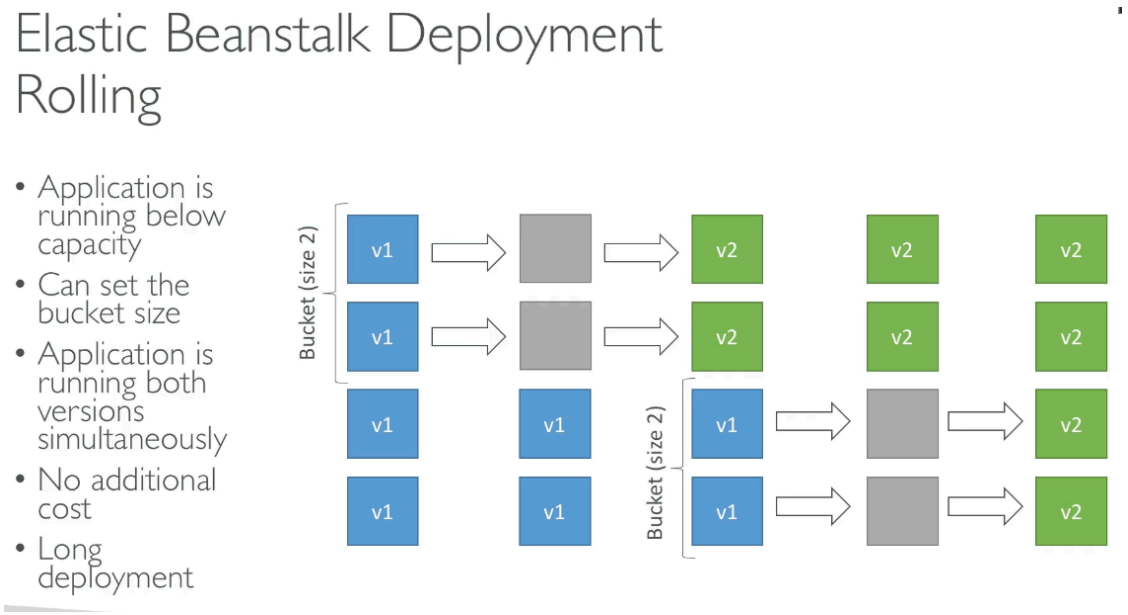
Beanstalk Deployment Modes

- All at once (deploy in one go): fastest and quick iterations, but has some downtime
- Rolling: update a few instances at a time and then move to the next bucket
- Rolling with additional batches: rolling, but has new instances to move the batch (so old app is still available)
- Immutable: spins up new instances in ASG, deploys version to these instances then swaps all instances when healthy
- Blue green: new environment and switch
- Traffic splitting: canary testing – send small % of traffic to a new deployment

All at Once

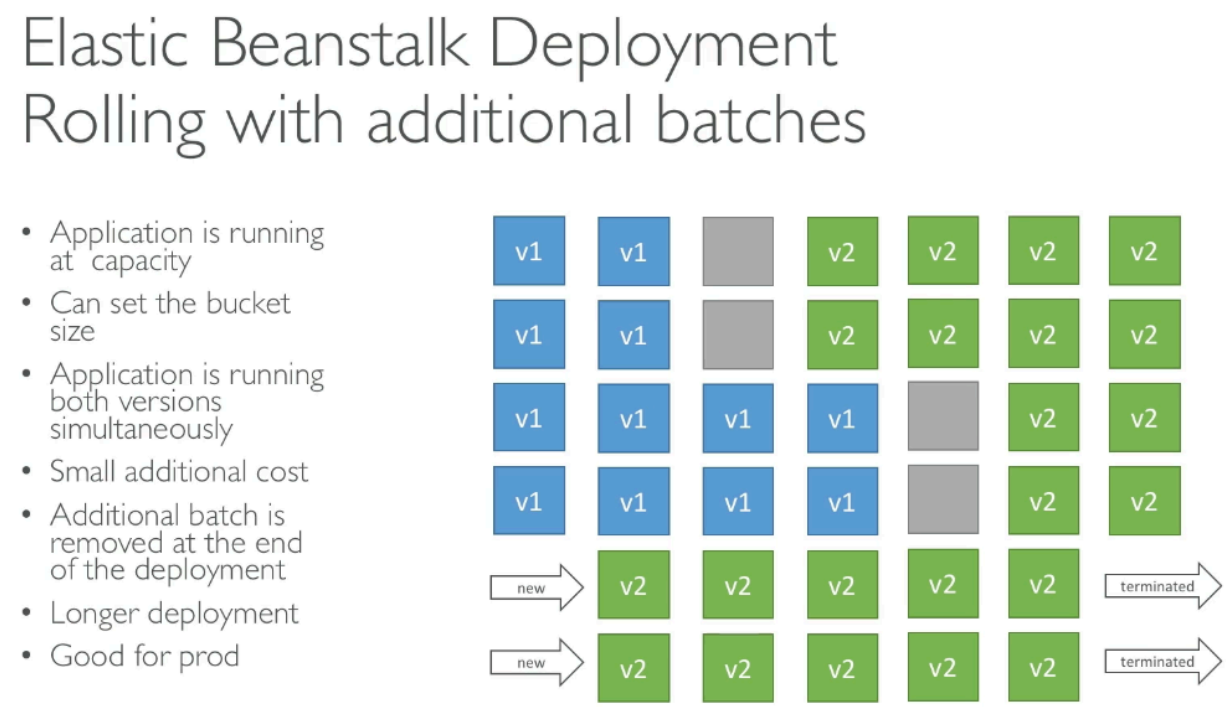


Rolling



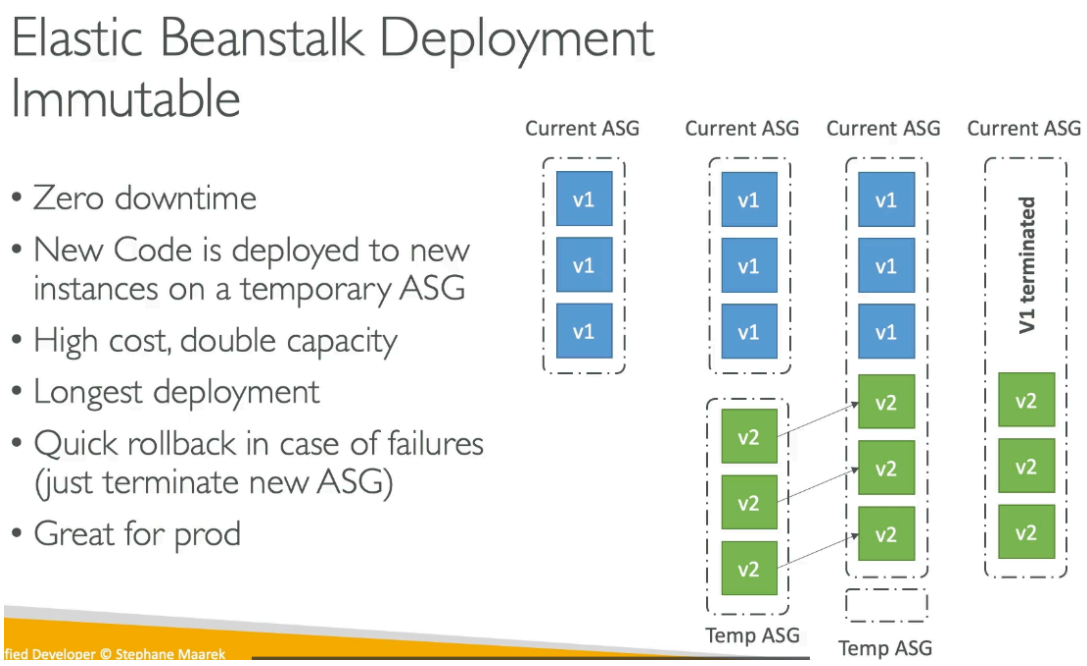
- Application is running below capacity (can set bucket size)
- Application is running both versions simultaneously with no additional cost
- Long deployment

Rolling with Additional Batches



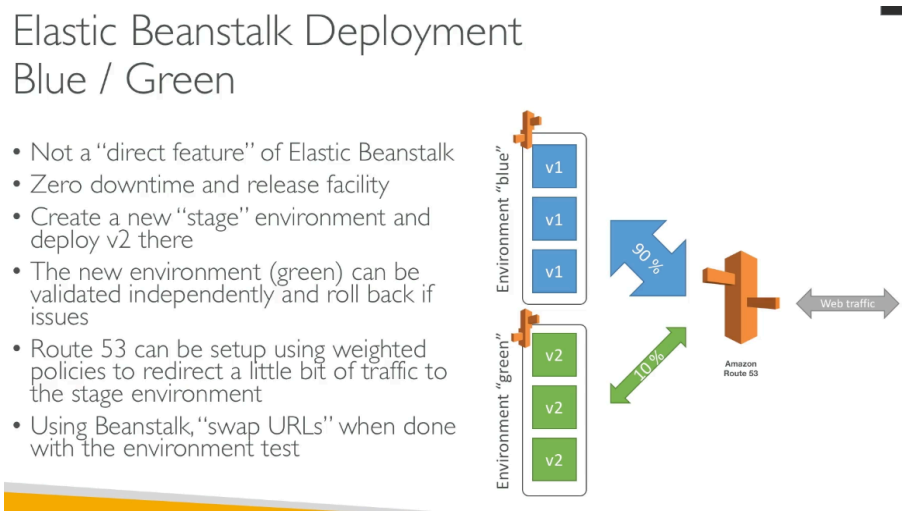
- Application is running at capacity, but small additional cost for additional batch until it is removed at the end of deployment
- Longer deployment, good for prod

Immutable



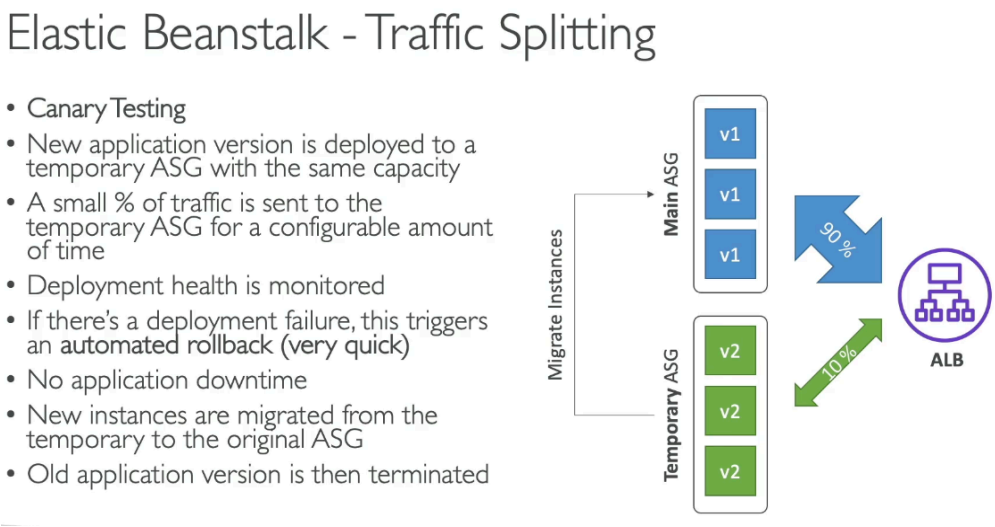
- 0 downtime, with new code deployed to new instances on a temporary ASG
- High cost, double capacity, longest deployment
- Quick rollback in case of failures (terminate new ASG), great for prod

Blue Green



- Not a direct feature, but has 0 downtime and release facility
- Create a new stage environment and deploy v2 there
- New env (green) can be validated and roll back if issues
- Route 53 can be setup using weighted policies to redirect a little bit of traffic to the stage environment
- Using Beanstalk, “swap URLs” when done with environment test

Traffic Splitting



- Canary Testing
- New application version deployed to a temporary ASG with the same capacity and a small % of traffic is sent to temporary ASG
- Deployment health is monitored, where failure triggers an automated rollback (fast)
- No application downtime
- New instances are migrated from temporary to original ASG
- Old application version is then terminated

Elastic Beanstalk Deployment Summary from AWS Doc

- https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.deploy-existing-version.html

Deployment methods

Method	Impact of failed deployment	Deploy time	Zero downtime	No DNS change	Rollback process	Code deployed to
All at once	Downtime	◷	x	✓	Manual redeploy	Existing instances
Rolling	Single batch out of service; any successful batches before failure running new application version	◷ ◷ †	✓	✓	Manual redeploy	Existing instances
Rolling with an additional batch	Minimal if first batch fails; otherwise, similar to **Rolling**	◷ ◷ ◷ †	✓	✓	Manual redeploy	New and existing instances
Immutable	Minimal	◷ ◷ ◷ ◷	✓	✓	Terminate new instances	New instances
Traffic splitting	Percentage of client traffic routed to new version temporarily impacted	◷ ◷ ◷ ◷ ††	✓	✓	Reroute traffic and terminate new instances	New instances
Blue/green	Minimal	◷ ◷ ◷ ◷	✓	x	Swap URL	New instances

Elastic Beanstalk CLI

- ELB cli
    - eb create, status, health, etc…

Deployment Process

- Describe dependencies, package as zip, upload zip and deploy with console or CLI
- Beanstalk will deploy zip on each EC2 instance and start application

Beanstalk Lifecycle Policy Overview

- At most 1000 app versions, will max out and cannot deploy
- Lifecycle policy based on time, space
- Option to not delete source bundle in S3 to prevent data loss

Extensions

- Must be set in .ebextensions/ directory in root in YAML / JSON format and must end in .config
    - Able to modify some default settings using option_settings

- Ability to add resources, but resources managed by .ebextensions get deleted if the environment goes away

Beanstalk & CloudFormation

- Relies on CF to provision other AWS services

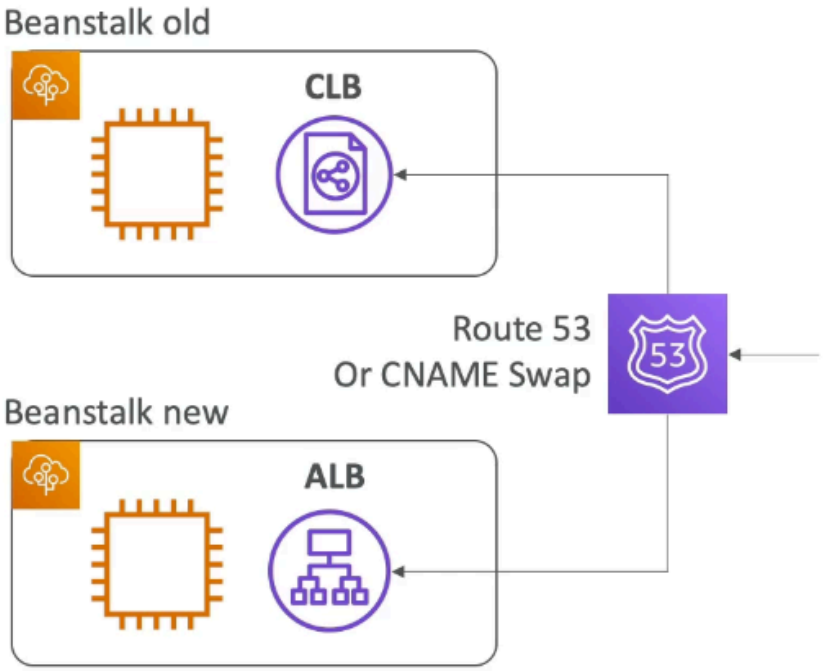
Beanstalk Cloning

- Clone an environment with the exact same configuration, useful for a test deployment version
- All resources and configuration preserved (LB, DB, env vars)
    - Settings can be changed after cloning

Beanstalk Migrations

Load Balancer

- After creating a beanstalk environment, you cannot change the ELB type (only configuration)
- To migrate:
    1. Create a new env with same configuration except LB
    2. Deploy app to new env
    3. Perform CNAME swap or Route 53 for DNS update

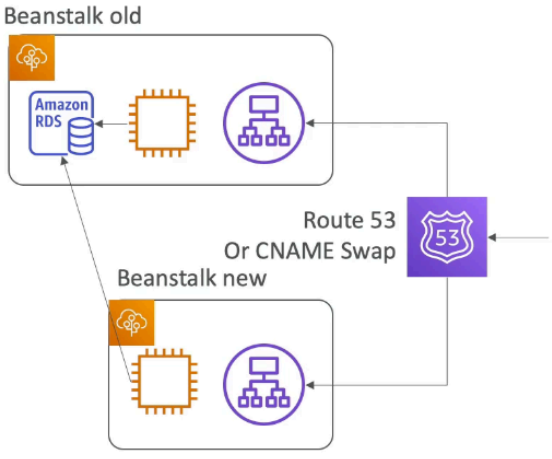


RDS

- Can be provisioned with Beanstalk
- Not great for prod as DB lifecycle is tied with Beanstalk
- Best way for prod is to separately create RDS DB and provide EB application with connection string

Decouple RDS

1. Create snapshot
2. Go to RDS console and protect DB from deletion



3. Create new Beanstalk env without RDS, point application to existing RDS
4. Perform CNAME swap (blue/green) or Route 53 update
5. Terminate only env (RDS won't be deleted)
6. Delete CF stack

Section 18: CloudFormation

- Declarative way to outline AWS infrastructure via code, created in the right order, declaratively
- IaaC
    - No resources manually created, version control with code
- Cost: all resources within the stack is tagged with in identifier to see cost
    - Can estimate costs
    - Can create and destroy
- Productivity:
    - Destroy and recreate infrastructure on the fly
- Separation of concern: create many stacks for many apps and many layers
- Don't reinvent the wheel with existing templates and documentation

How CF works

- Templates uploaded to S3 and then referenced by CF, where a stack is created and resources made
- New version is needed to update the stack
- Stacks are identified by name and deleting the stack removes all corresponding resources
- Manual way of Application Composer or code editor, console
- Automated way: edit templates as YAML and using CLI to deploy templates

CF Building Blocks

- Template Components
    - AWSTemplateFormatVersion: identifies the template capabilities
    - Description
    - Resources (mandatory): AWS resources declared
    - Parameters: dynamic inputs
    - Mappings: static variables
    - Outputs: references to what is created
    - Conditionals: list of conditions to perform resource creation

- Template Headers
    - References
    - Functions

CF Resources

- Core of CF (mandatory), and represent different AWS components that will be created and configured
    - Resources are declared and can reference each other
- Resource type identifiers in the form:

```
service-provider::service-name::data-type-name
```

- Not every AWS service is supported and the workaround is CF Custom Resources

CF Parameters

- Way to provide inputs to CF templates
    - Important for reuse as some inputs cannot be determined ahead of time
- When to use?
    - Is CF resource configuration likely to change in the future? Can it be determined ahead of time?
- Parameters can be controlled by a number of settings:
    - Type: string, num, list, etc…
    - Description
    - Default value, etc…

How to reference a parameter?

- !Ref function to reference elements within the template

Pseudo Parameters

- Used at any time and enabled by default

Reference Value	Example Returned Value
AWS::AccountId	123456789012
AWS::Region	us-east-1
AWS::StackId	arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-11e3-aff7-50e2416294e0
AWS::StackName	MyStack
AWS::NotificationARNs	[arn:aws:sns:us-east-1:123456789012:MyTopic]
AWS::NoValue	Doesn't return a value

CF Mappings

- Fixed variables to differentiate between environments, regions…
- Hardcoded

Accessing Mapping Values (FindInMap)

- FindInMap to return a named value from a specific key
    - !FindInMap [MapName, TopLevelKey, SecondLevelKey]
- Works great for AMIs because they are region specific

Mappings vs Parameters

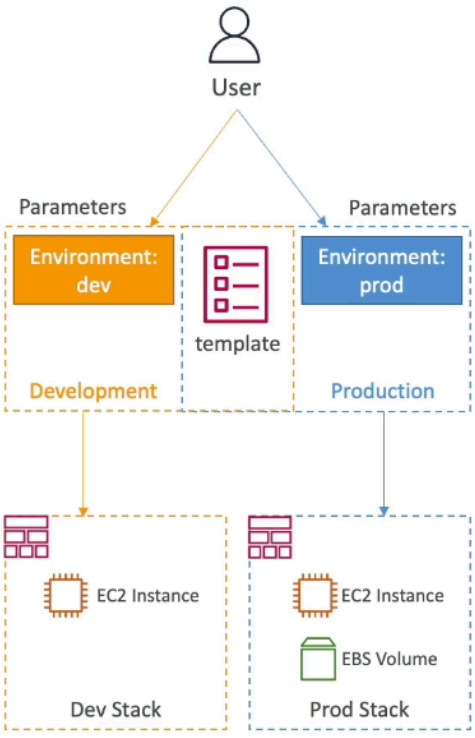
- Mappings when you know in advance all the values that can be taken and they can be deduced from pseudo parameters
    - Allow safer control over template
- Parameters when the values are user specific

CF – Outputs

- Declares optional output values to import into other stacks (after export)
    - View outputs in console or CLI
    - Best way to perform collaboration across stacks
        - Fn::ImportValue function to reference exported values from another stack
- Cannot delete the first stack that has its value exported until all referenced stacks are deleted

CF – Conditions

- Control the creation of resources or outputs based on a condition
- Each condition can reference another condition, parameter value or mapping
- The logical ID you can choose, it's how you name the condition
- Intrinsic function can be AND, EQ, NOT, IF, OR



CF – Intrinsic Functions

CloudFormation – Intrinsic Functions

Blue = must know

- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Fn::ForEach
- Fn::ToJsonString
- Fn::Base64
- Fn::Cidr
- Fn::GetAZs
- Fn::Select
- Fn::Split
- Fn::Transform
- Fn::Length
- Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc…)

Fn::Ref or !Ref

- Leveraged to reference:
  - Parameters: returns the value of the parameter
  - Resources: returns physical ID of underlying resource

Fn::GetAtt

- Attributes are attached to any resources you created

Fn::FindInMap

Intrinsic Functions – Fn::FindInMap

- We use Fn::FindInMap to return a named value from a specific key
- !FindInMap [MapName, TopLevelKey, SecondLevelKey]

```
Mappings:
  RegionMap:
    us-east-1:
      HVM64: ami-0ff8a91507f77f867
      HVMG2: ami-0a584ac55a7631c0c
    us-west-1:
      HVM64: ami-0bdb828fd58c52235
      HVMG2: ami-066ee5fd4a9ef77f1

Resources:
  MyEC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", HVM64]
      InstanceType: t2.micro
```

- Return a named value from a specific key

Fn::ImportValue
- Import values that are exported in other stacks

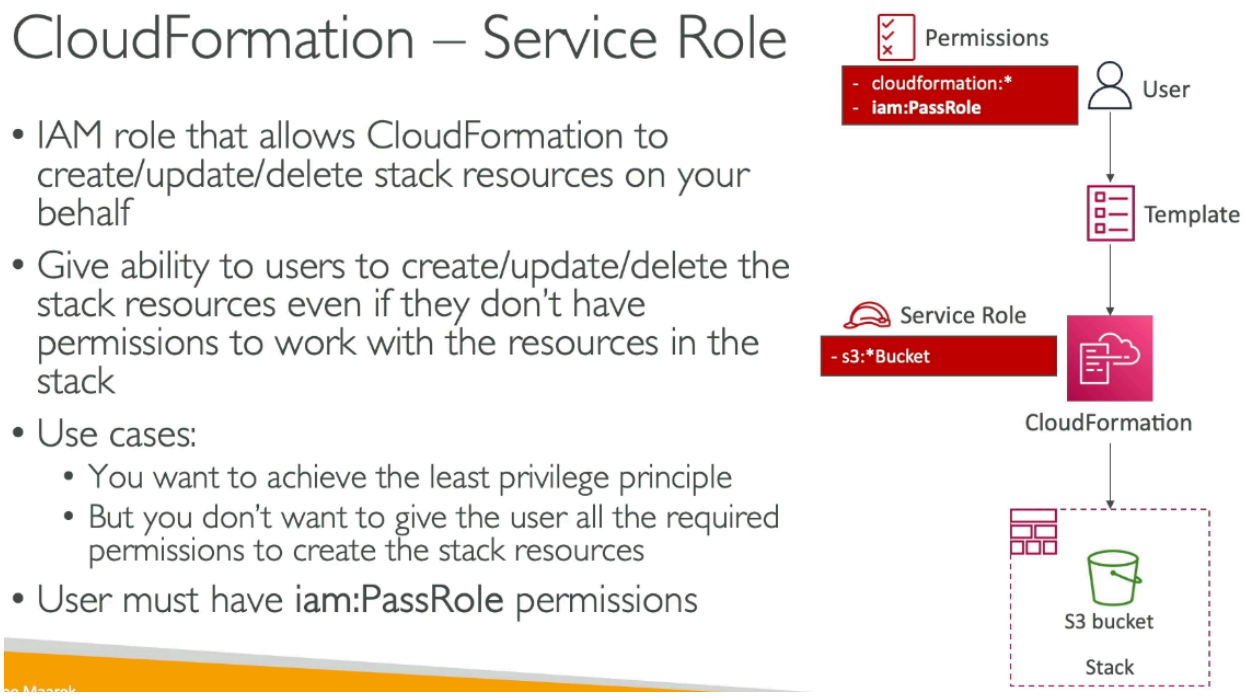
Fn::Base64
- Convert String to base 64
    - Pass encoded data to EC2 instance UserData property: !Base64 “value”

Conditions Functions
- Conditionally create resources using logical ID and intrinsic function

CF – Rollbacks

- Stack Creation Fails:
    1. Default: everything rolls back (deleted). View logs to see what happened
        a. Option to disable rollback and troubleshoot
- Stack Update Fails:
    - Stack auto rolls back to previously known working state
        - See logs of error messages
- Rollback Failure: fix resources manually then issue ContinueUpdateRollback API from console or CLI

CF – Service Roll



- IAM role that allows CF to create / update / delete stack resources on behalf
    - Gives users ability to update resources even without permissions to work with the resources in the stack
        - User must have iam:PassRole Permissions

CF – Capabilities

- CAPABILITY_NAMED_IAM and CAPABILITY_IAM
    - Necessary to enable when CF template is creating or updating IAM resources
    - Specify CAPABILITY_NAMED_IAM if the resources are named
- CAPABILITY_AUTO_EXPAND
    - Needed when CF template includes macros or nested stacks to perform dynamic transformations
        - Acknowledge template may change before deploying
- InsiffucientCapabilitiesException
    - Thrown if capabilities haven't been acknowledge when deploying a template

CF – Deletion Policy

- DeletionPolicy:
    - Control what happens when template is deleted or when resource is removed from template, extra safety to backup resources
    - Default DeletionPolicy = delete
        - If S3 bucket not empty, it fails, must manually delete files within S3 bucket or define a custom resource to delete everything in S3 bucket
- DeletionPolicy Retain
    - Specify which resources to preserve in deletion, works with any resources
- DeletionPolicy Snapshot
    - Create final snapshot before resource deletion

CF – Stack Policies

- During stack update, all update actions are allowed on all resources by default
- Stack policy (JSON) defined the update actions allowed on specific resources during Stack updates
- Protection from unintentional updates
- In a stack policy, all resources are protected by default
    - Specify an explicit allow for resources to be updated

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "Update:*",
            "Principal": "*",
            "Resource": "LogicalResourceId/ProductionDatabase"
        }
    ]
}
```

Allow updates on all resources **except** the ProductionDatabase

CF – Termination Protection

- TerminationProtection to prevent accidental deletes; safety against accidental deletes

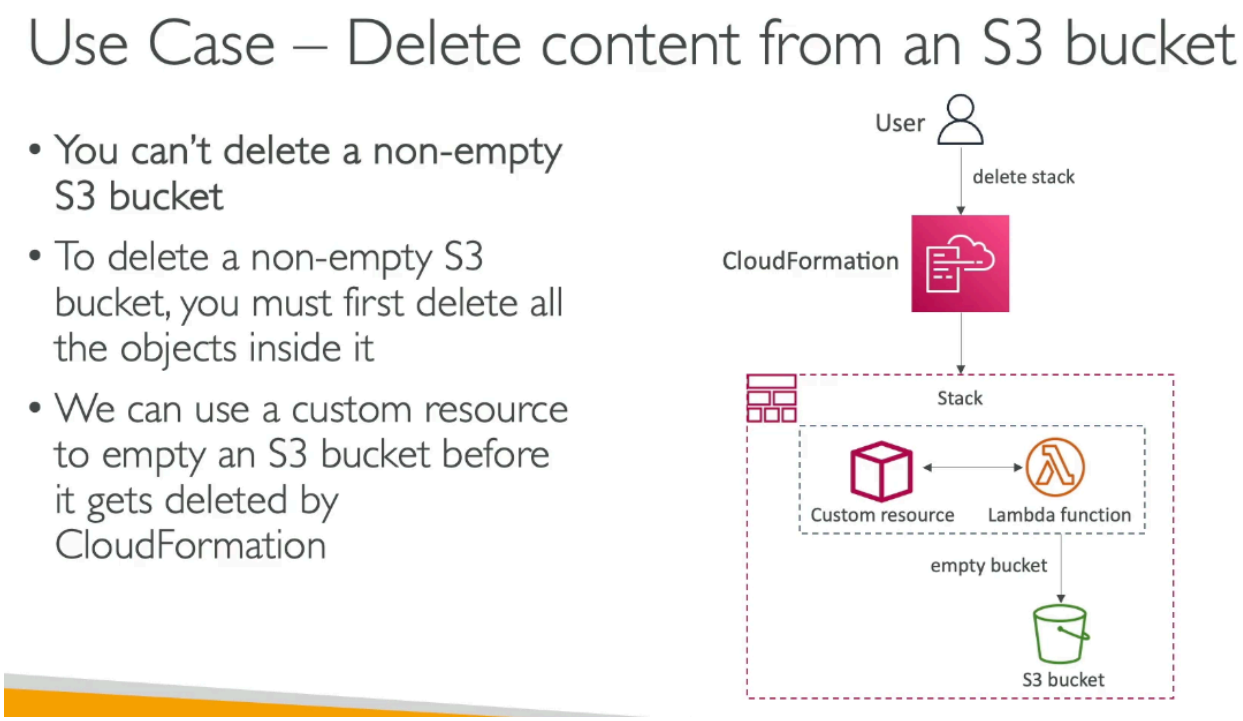
CF – Custom Resources

- Used to define resources not yet supported by CF, define custom logic for resources outside of CF (on prem, 3rd party), custom scripts via Lambda functions (run a lambda function to empty S3 bucket before being deleted)
- Defined in template using:
  - AWS::CloudFormation::CustomResource or Custom::MyCustomResourceTypeName
- Backed by lambda function (most common) or SNS topic

How to define a custom resource?

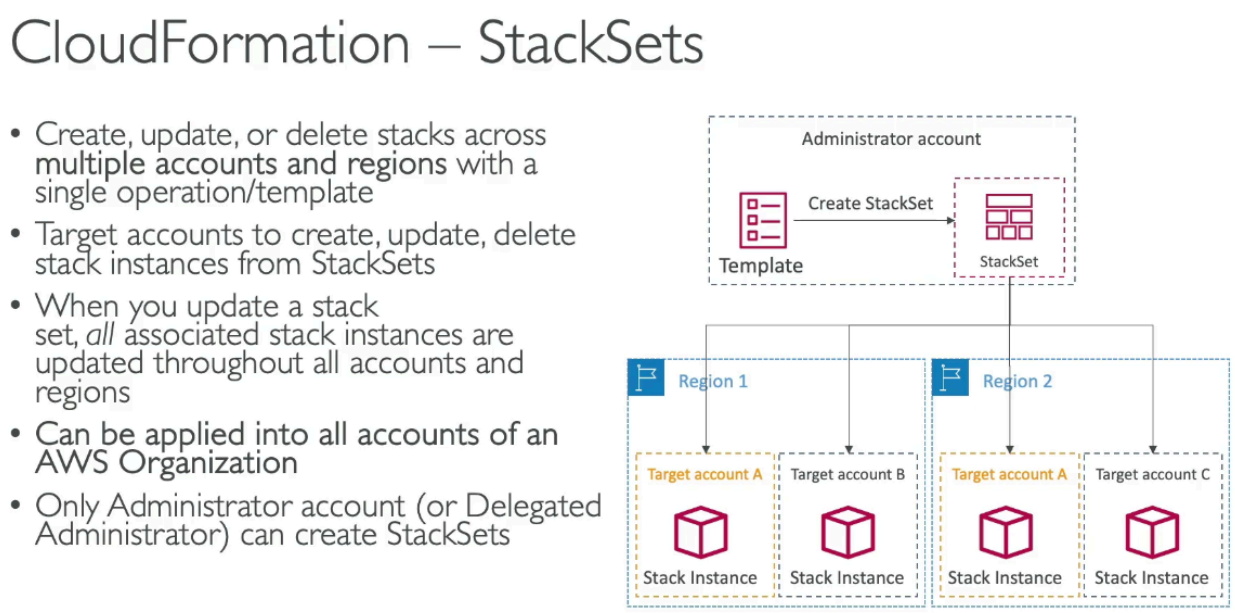
- ServiceToken specifies where CF sends request to, such as Lambda ARN or SNS ARN (required and must be in same region)
- Input parameters optional

Use Cases



- Cannot delete a non-empty S3 bucket, must delete all objects inside it first

CF – StackSets



- Create, update, or delete stacks across multiple accounts and regions within a single template
- When you update stack set, all associated stack instances are updated across all accounts and regions
- Can be applied into all accounts of an AWS Organization, only admin account can create stack sets

Section 19: AWS Integration & Messaging: SQS, SNS & Kinesis

Intro to Messaging

1. Synchronous communication (application to application)
    a. Synchronous between apps if there are sudden spikes in traffic
    b. Decoupling helps to scale independently
2. Asynchronous communication / event based (application → queue → application)

SQS

- A queue has producers that send messages and a consumer that polls messages
- Decouples between producers and consumers

Standard Queue

- Oldest offered, fully managed service for decoupling apps
- Attributes:

        - Unlimited throughput, unlimited number of messages in queue
        - Default retention of messages 4 days, max 14 days
        - Low latency (<10ms on publish and receive)
        - Limitation of 256 KB per message sent
    - Can have duplicate messages (at least once delivery)
    - Can have out of order messages (best effort ordering)

Producers

- Via SendMessageAPI in SDK
- Message is persisted in SQS until a consumer deletes
    - Retention of 4 default days, max 14

Consumers

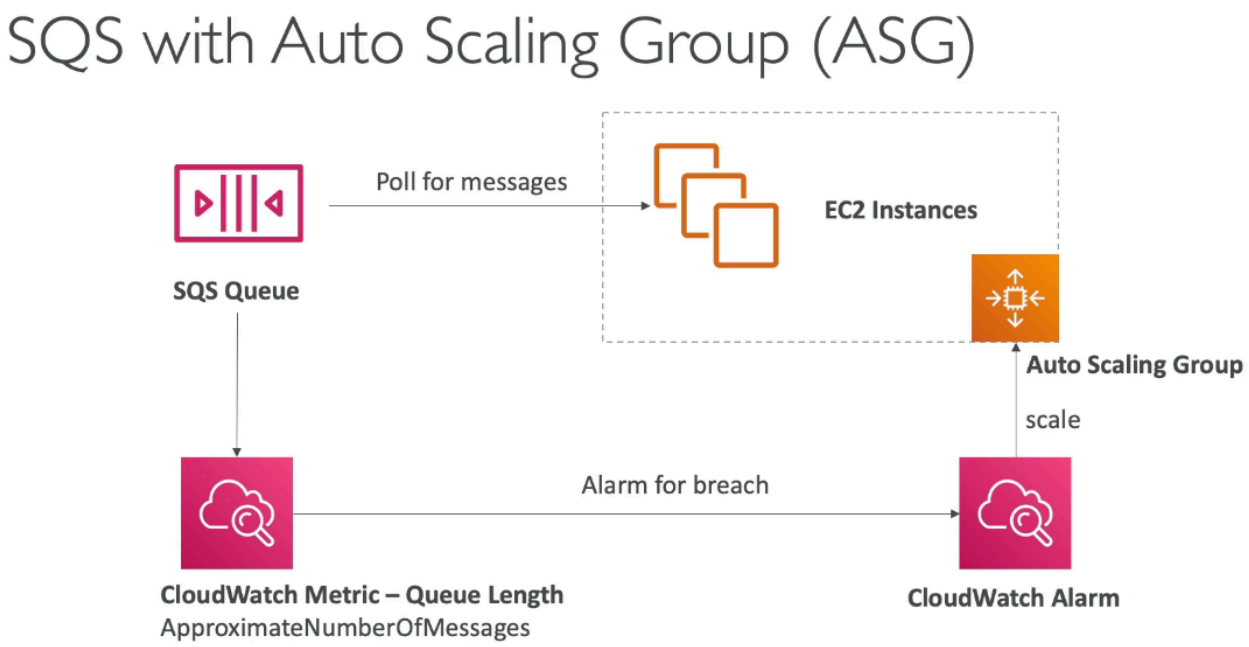
- Consumers on EC2, Lambda, etc…
- Poll SQS for messages (up to 10 at a time) and process messages
- Delete messages via DeleteMessageAPI

Multiple EC2 Instance Consumers

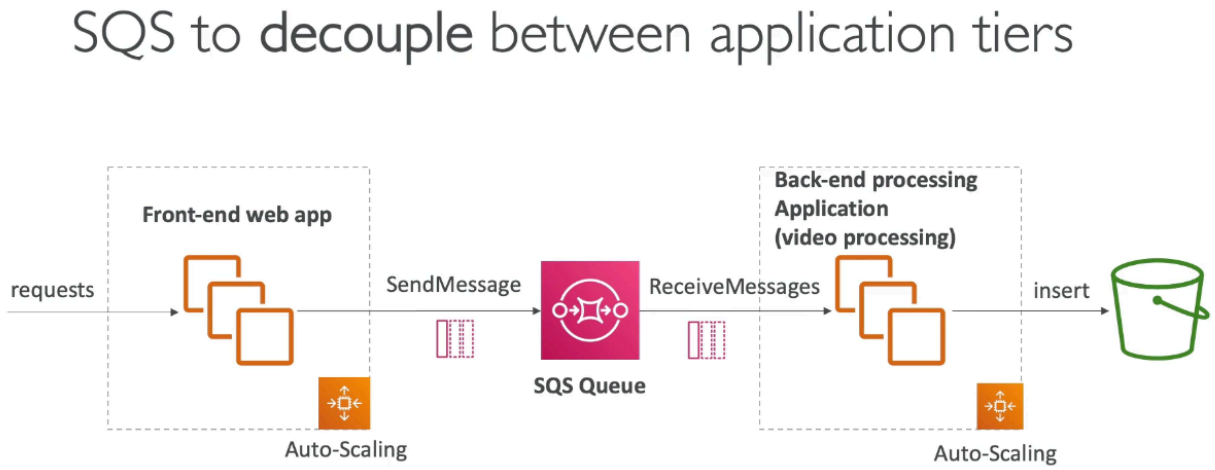
- Consumers receive and process messages in parallel
- At least once delivery
- Best effort message ordering
- Consumers delete messages after processing them
- Scale consumers horizontally to improve processing

SQS w/ ASG

- Queue length CloudWatch metric alarm to increase capacity of ASG



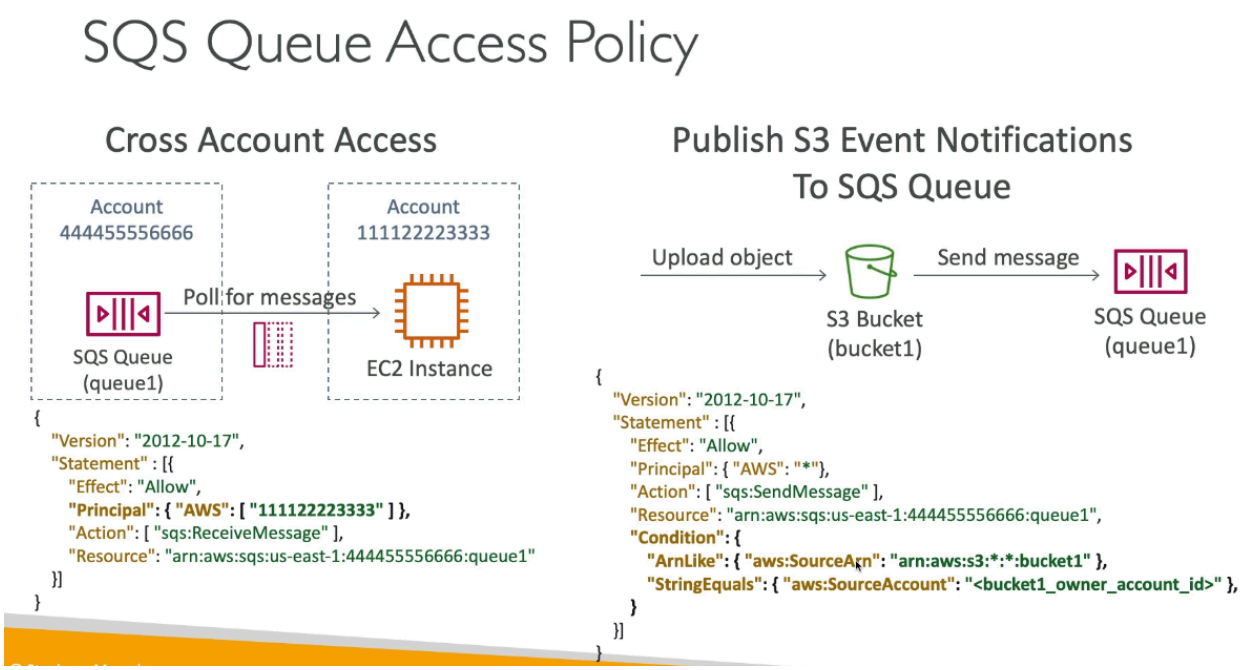
SQS to decouple application tiers



SQS Security

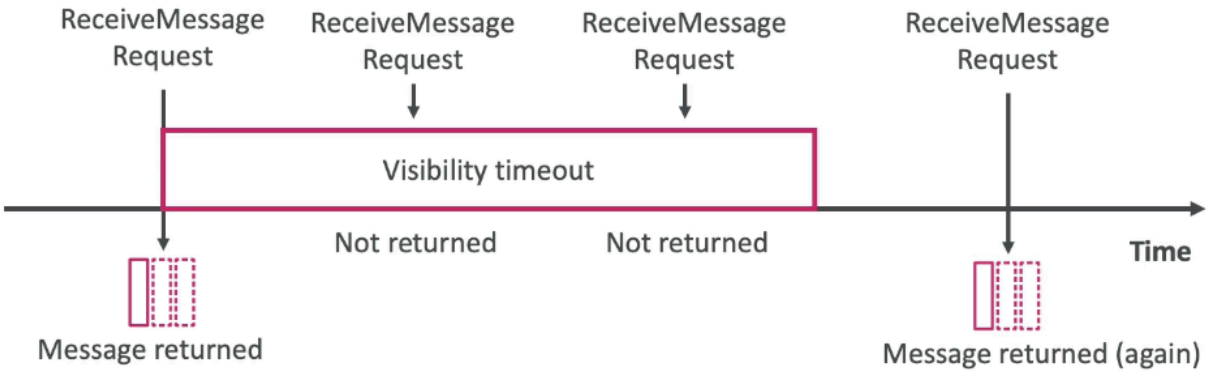
- Encryption
    - In flight via HTTPS API
    - At rest via KMS keys
    - Client side
- Access Controls: IAM policies to regulate access to SQS API
- SQS Access Policies (similar to S3 bucket policies)
    - Cross account access to SQS
    - Allow other services to write to SQS

SQS Queue Access Policy



- Cross Account Access
- Publish S3 Event Notifications to SQS Queue

Message Visibility Timeout



- After a message is polled by a consumer, it becomes invisible to other consumers
  - Default timeout of 30 seconds, which means the message must be processed within 30 seconds.
  - After the timeout ends, the message is “visible” and back on queue in SQS
- If a message is not processed within timeout, it will be processed twice
- A consumer could call ChangeMessageVisibility API to get more time if it needs longer to process

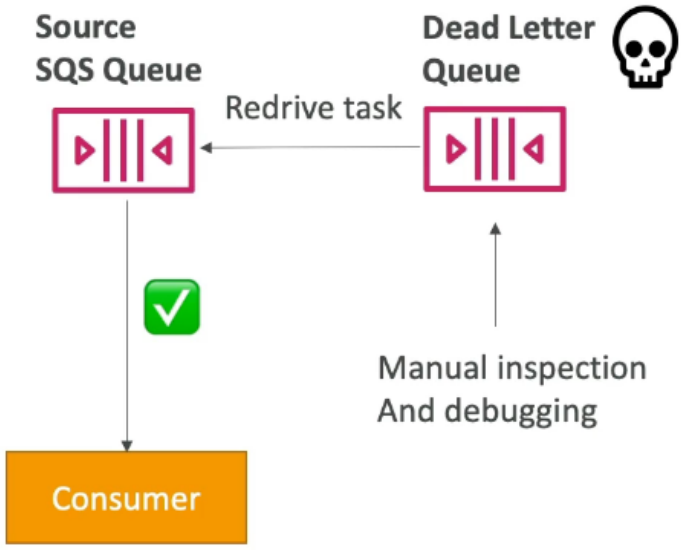
    - If timeout is high (hours) and consumers crashes, reprocessing takes time
    - If timeout is too low (seconds), we may get duplicate processing

Dead Letter Queues (DLQ)

- If a consumer fails to process a message within timeout, it reappears in the queue
    - A threshold of how many times a message can go back in queue (prevent loop)
    - MaximumReceives threshold exceeded, message goes in DLQ
- Debugging purposes
- DLQ of a FIFO queue must also be FIFO, DLQ of standard queue must also be standard queue
- Make sure to process messages before expire (good to set longer retention of DLQ)

DLQ – Redrive to Source

- Feature to help consume messages in DLQ to understand what is wrong
- When fixed, redrive the message from DLQ back to source queue in batches without custom code



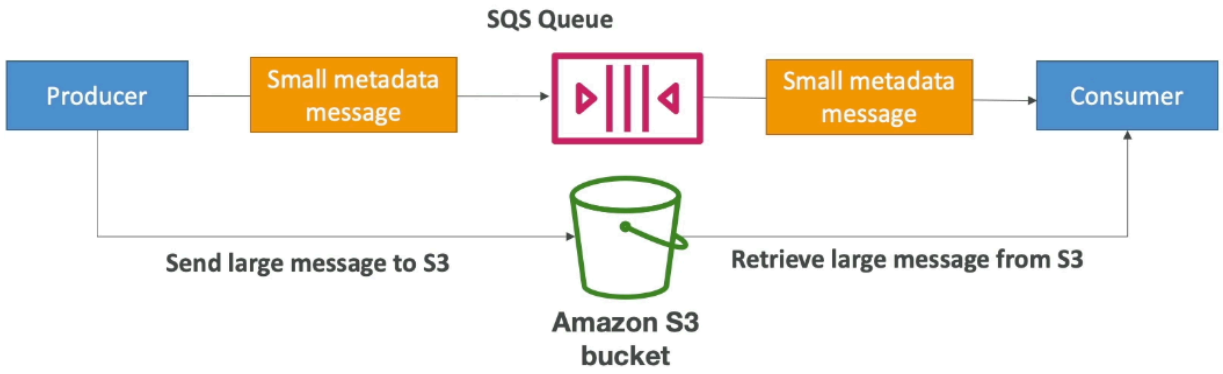
Delay Queue

- Delay message (consumers don't get immediately) up to 15 min (default 0, instantly)
    - Set default at queue level
    - Can override default on send using DelaySeconds parameter

Long Polling

- When consumer requests message, it can "wait" for messages to arrive if queue empty, called long polling
- Done to decrease # API calls made to SQS while increasing efficiency and decreasing latency of application
    - Wait of 1 to 20 sec
    - Preferable to short polling (short polling is 0 seconds)
- Enabled at queue level or at API queue level using ReceiveMessageWaitTimeSeconds

SQS Extended Client



- Message limit size 256 KB, but SQS Extended Client uses S3 bucket as a repo for large data. S3 receives message first, but queue gets small metadata message of a pointer to larger message
    - Consumer reads small metadata message pointer from S3

Must know API calls

SQS – Must know API

- CreateQueue (MessageRetentionPeriod), DeleteQueue
- PurgeQueue: delete all the messages in queue
- SendMessage (DelaySeconds), ReceiveMessage, DeleteMessage
- MaxNumberOfMessages: default 1, max 10 (for ReceiveMessage API)
- ReceiveMessageWaitTimeSeconds: Long Polling
- ChangeMessageVisibility: change the message timeout

- Batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility helps decrease your costs

FIFO Queues

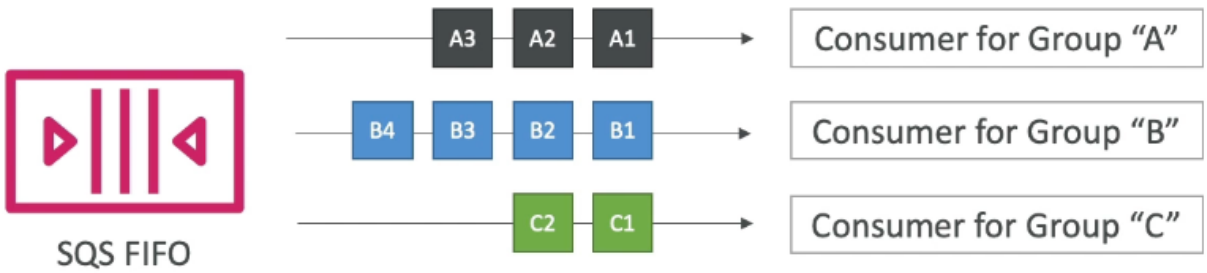
- First in first out (ordering of messages in the queue)
- Limited throughput: 300 msg/s without batching, 3000 msg/s with
- Exactly once send capability (by removing duplicates)
- Messages processed in order by the consumer
- Must have .fifo at the end

Advanced FIFO

Deduplication

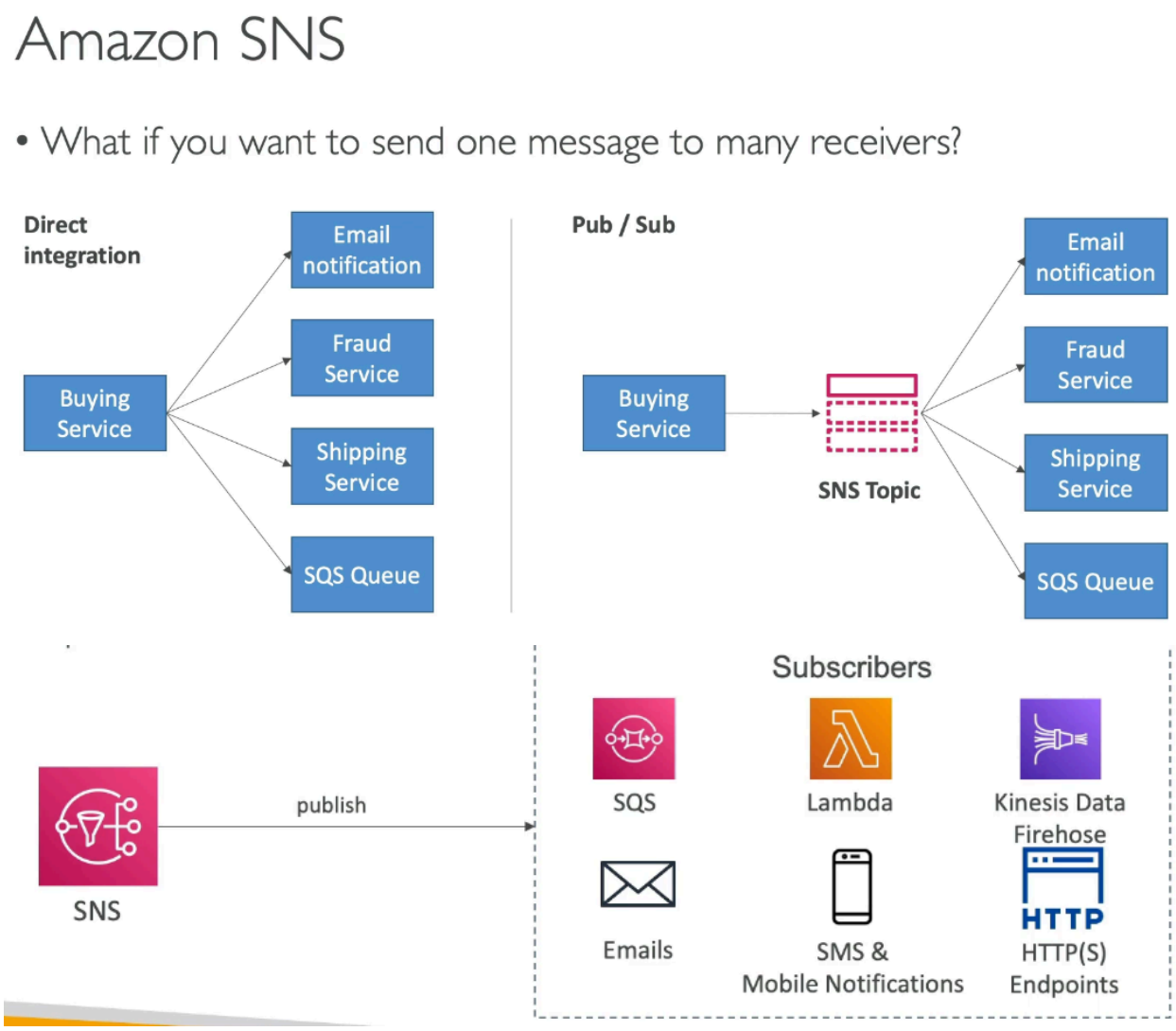
- Interval is 5 minutes where if a duplicated message is sent within the interval, it will be refused
- 2 methods:
    1. Content based deduplication: will do hash on message body
    2. Explicitly provide a message duplication ID

Message Grouping



- If specified the same value of MessageGroupID in FIFO queue, you can only have 1 consumer and all messages are in order
- To get ordering at subset level of messages, specify different values for MessageGroupID
    - Messages that share common group ID will be in order within the group
    - Each group ID has different consumer (parallel processing)
    - Ordering across groups not guaranteed

SNS



- Pub / sub of a middle man between publisher and subscribers
- “Event producer” only sends message to 1 SNS topic and “event receiver” (subscriber) will listen to SNS topic notifications
    - Each sub to the topic will get the messages and many subs per topic allowed
- AWS services can send data directly to SNS for notifications

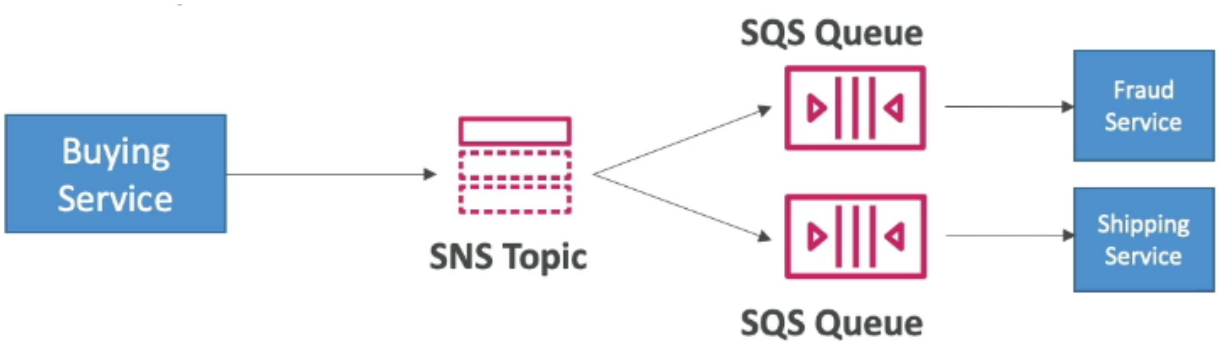
How to publish

- Topic Publish (SDK)
    - Create topic, subscription, publish
- Direct Publish (mobile apps SDK)
    - Create a platform application, platform endpoint publish

SNS Security

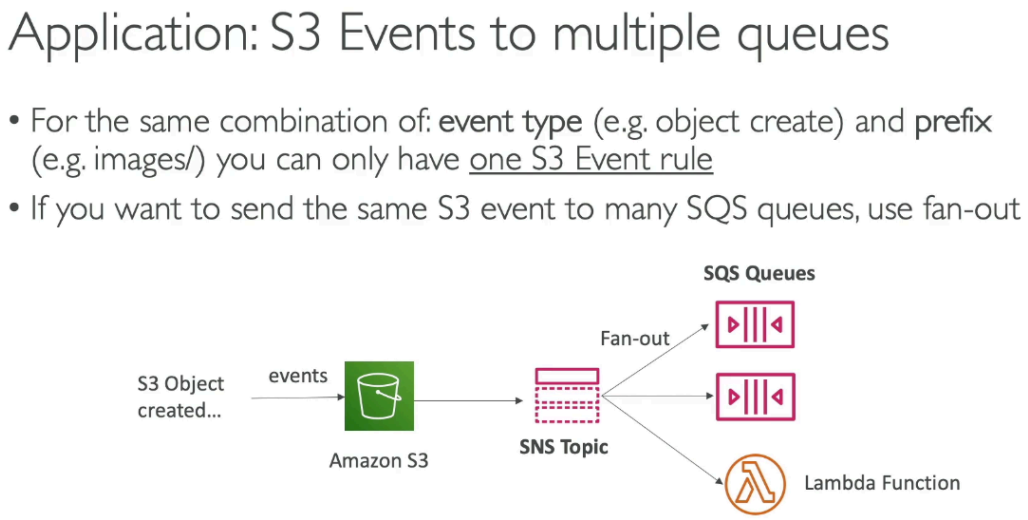
- Encryption:
  - In flight via HTTPS
  - At rest via KMS
  - Client side
- Access Controls: IAM policies
- SNS Access Policies (similar to S3 bucket policies)
  - Cross account, allow other services to write to SNS topic

SNS + SQS Fan Out Pattern



- Push once in SNS, receive in all SQS queues that are subscribers
- Fully decoupled, no data loss
  - SQS allows for: data persistence, delayed processing and retries
- Ability to add more SQS subscribers over time
- Must have SQS queue access policy allow for SNS to write
  - Cross region delivery for SQS and SNS different regions

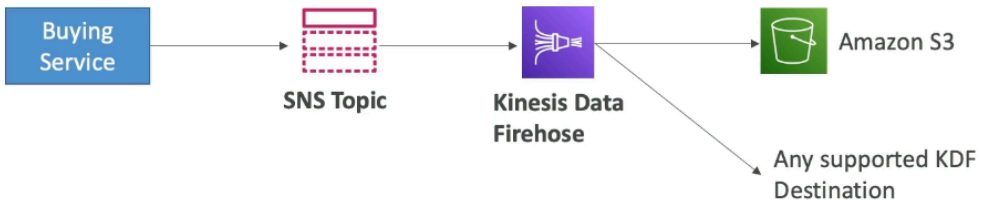
S3 Events to multiple queues



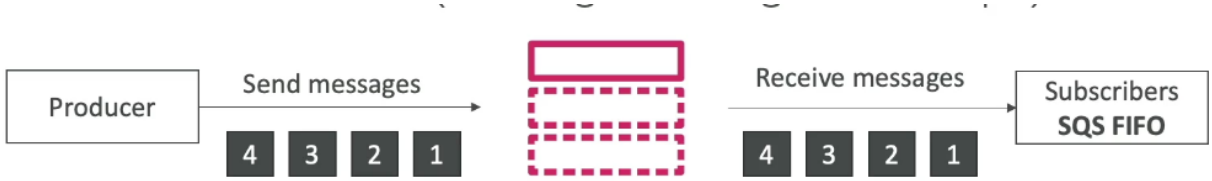
SNS to S3 via Kinesis

Application: SNS to Amazon S3 through Kinesis Data Firehose

- SNS can send to Kinesis and therefore we can have the following solutions architecture:

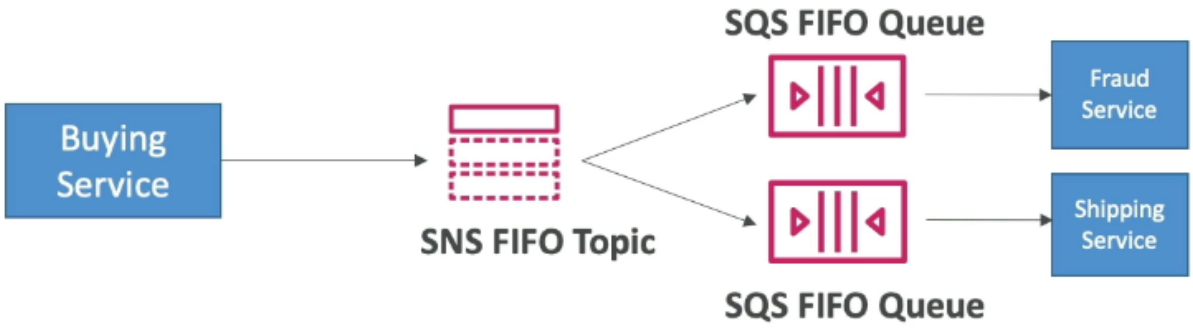


SNS – FIFO Topic



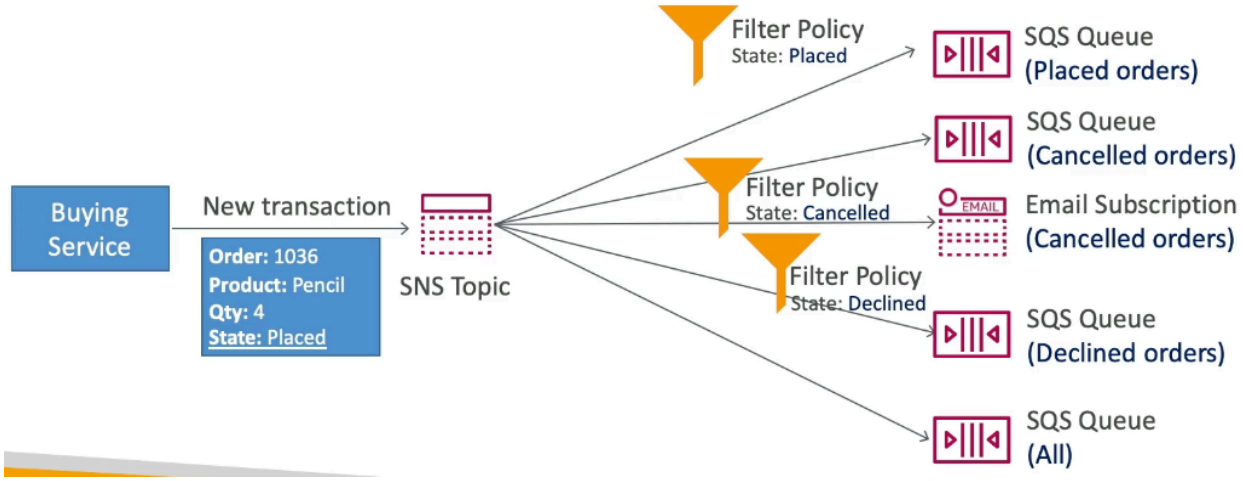
- Similar as SQS FIFO
  - Ordering by message group ID
  - Deduplication via ID or Content based
- Can ONLY have SQS Standard and FIFO queues as subscribers

SNS FIFO + SQS FIFO: Fan Out



- Fan out + ordering + deduplication

Message Filtering

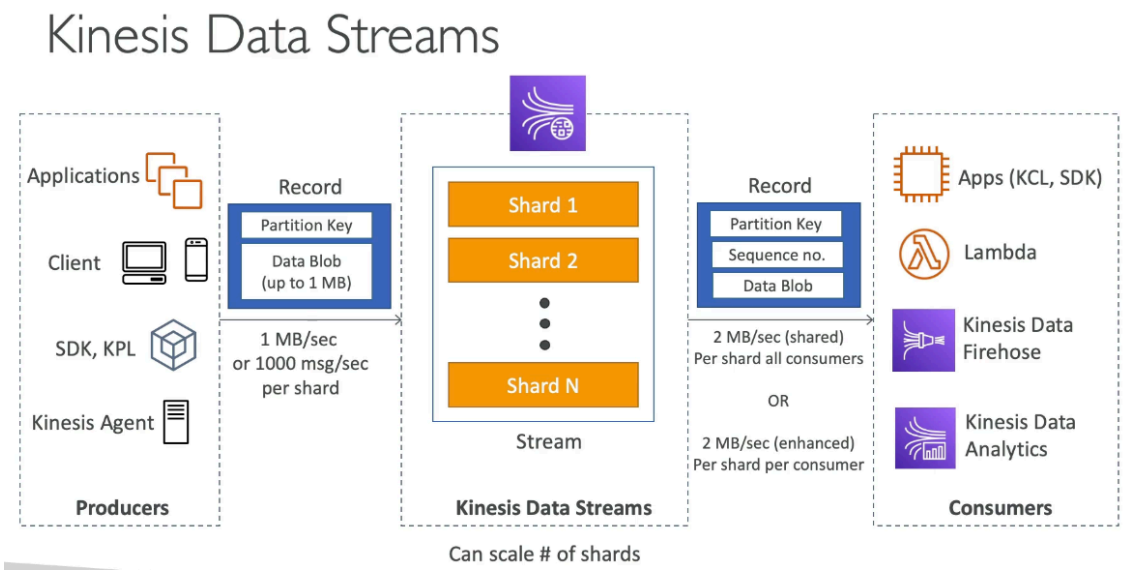


- JSON policy to filter messages sent to SNS topic's subscriptions

Kinesis

- Makes it easy to collect, process, and analyze streaming data in real time
    - Application logs, metrics…
- Kinesis Data Streams: capture, process and store data streams
- Kinesis Data Firehose: load data streams into AWS data stores
- Kinesis Data Analytics: analyze data streams with SQL or Apache Flink
- Kinesis Video Streams: capture, process and store video streams

Kinesis Data Streams

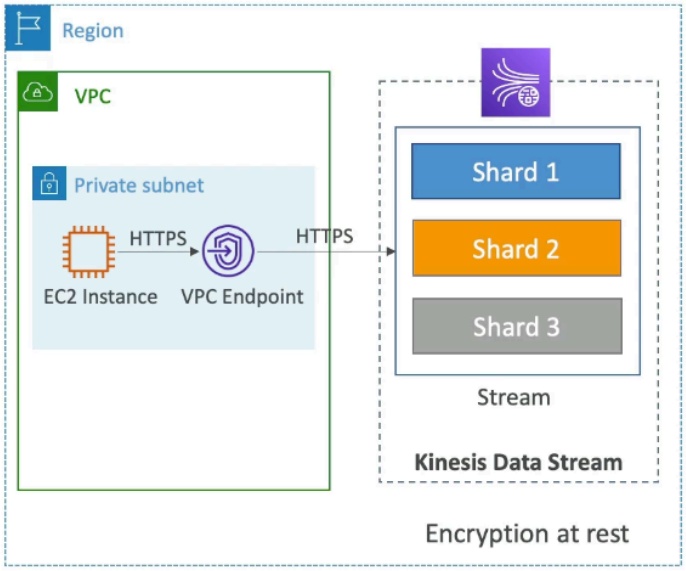


- Made of shards, provisioned ahead of time. Data is split across all shards. Producers send records (made of partition key + data blob).

- Retention between 1 and 365 days with ability to reprocess data
- Once data in Kinesis, cannot be deleted (immutability) and data that shared the same partition key goes in the same shard (ordering) → key based ordering

Capacity Modes

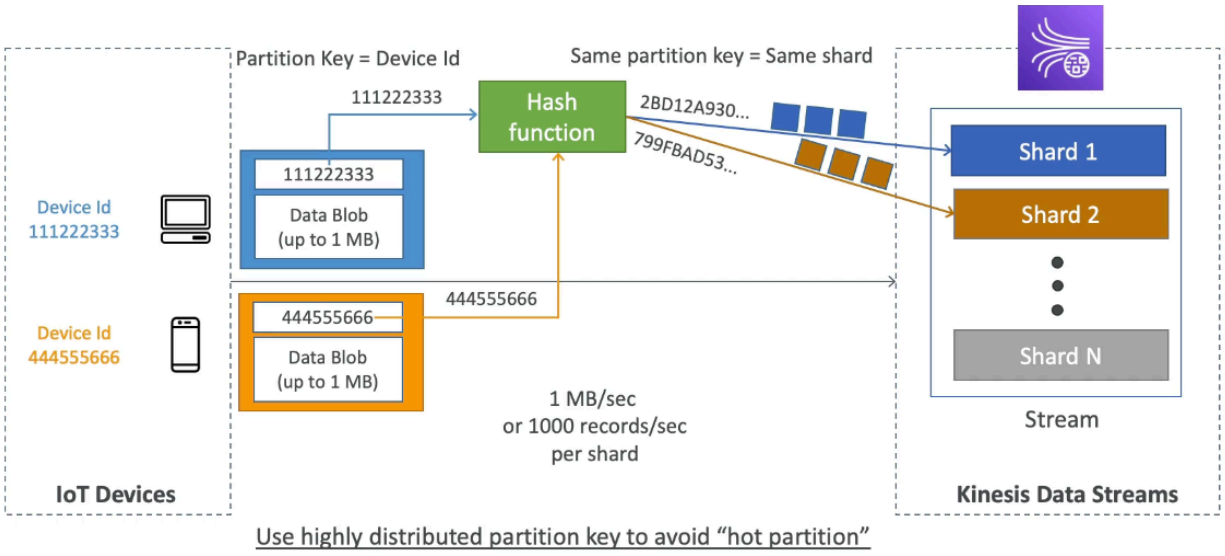
- Provisioned Mode:
  - Choose # shards provisioned, each shard gets 1 MB/s in and 2 MB/s out
  - Pay per shard per hour
- On demand:
  - No provisioning or capacity management
  - Default capacity provisioned (4 MB/s)
  - Scales automatically based on throughput peak in last 30 days
  - Pay per stream per hour & data in/out per GB



Security

- Control access via IAM policies, in flight, at rest, client side encryption
- VPC endpoints available for kinesis to access within VPC
- Can be monitored via CloudTrail

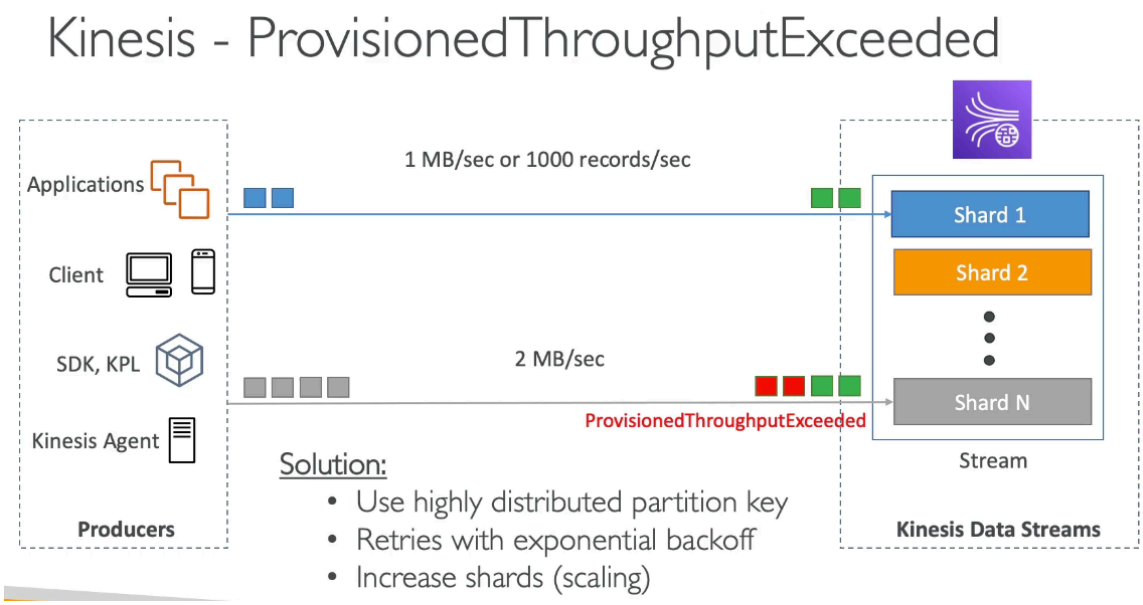
Kinesis Producers



- Puts data records in data streams
  - Data records have:

    - Sequence # → unique per partition key within shard
    - Partition key → must specify while put records into stream
    - Data blob
- Hot partition occurs when one shard is unevenly getting more data than others
- Producers: SDK, Kinesis Producer Library (KPL), Kinesis Agent (monitor log files)
- PutRecord API → can use batching to reduce cost and increase throughput

ProvisionedThroughputExceeded



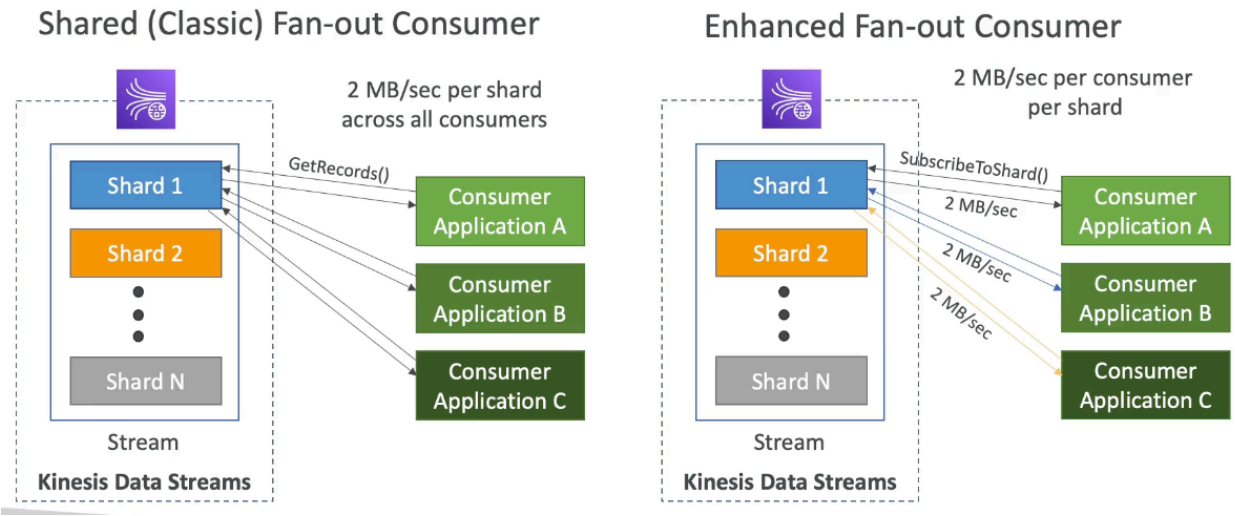
- Resolve by using highly distributed partition key and retries with exponential backoff or increase shards (scaling)

Kinesis Consumers

- Lambda, other Kinesis tools, Custom Consumer (classic or enhanced fan out), or KCL

Custom Consumer

Kinesis Consumers – Custom Consumer



- The difference between shared and enhanced is that shared uses a pull model while enhanced uses a push model

Kinesis Consumers Types

Shared (Classic) Fan-out Consumer - pull

- Low number of consuming applications
- Read throughput: 2 MB/sec per shard across all consumers
- Max. 5 GetRecords API calls/sec
- Latency ~200 ms
- Minimize cost ($)
- Consumers poll data from Kinesis using GetRecords API call
- Returns up to 10 MB (then throttle for 5 seconds) or up to 10000 records

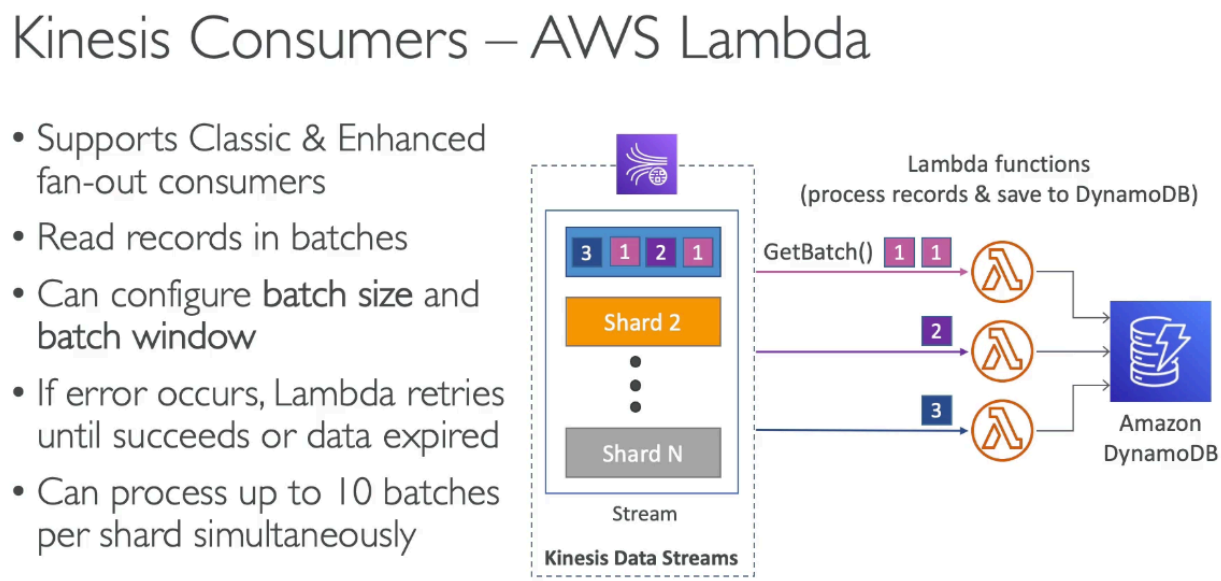
Enhanced Fan-out Consumer - push

- Multiple consuming applications for the same stream
- 2 MB/sec per consumer per shard
- Latency ~70 ms
- Higher costs ($$$)
- Kinesis pushes data to consumers over HTTP/2 (SubscribeToShard API)
- Soft limit of 5 consumer applications (KCL) per data stream (default)

- Shared (pull method):
    - Low number of consuming applications
    - Read throughput: 2 MB/sec per shard across all consumers
    - Minimizes cost
    - GetRecords API call
- Enhanced (push method):
    - Multiple consuming applications for the same stream

- 2 MB/sec per consumer per shard
- Highest cost

AWS Lambda

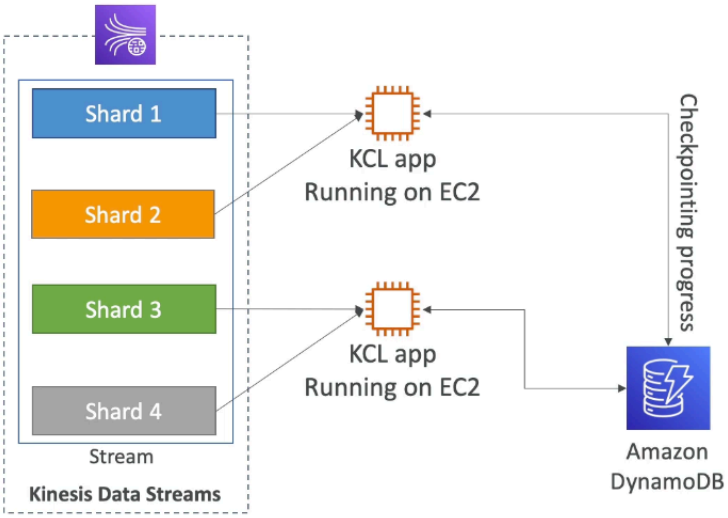


- Supports both classic and enhanced fan-out consumers
- Reads records in batches (configure size and window)
- If error, Lambda retries until success or data expired

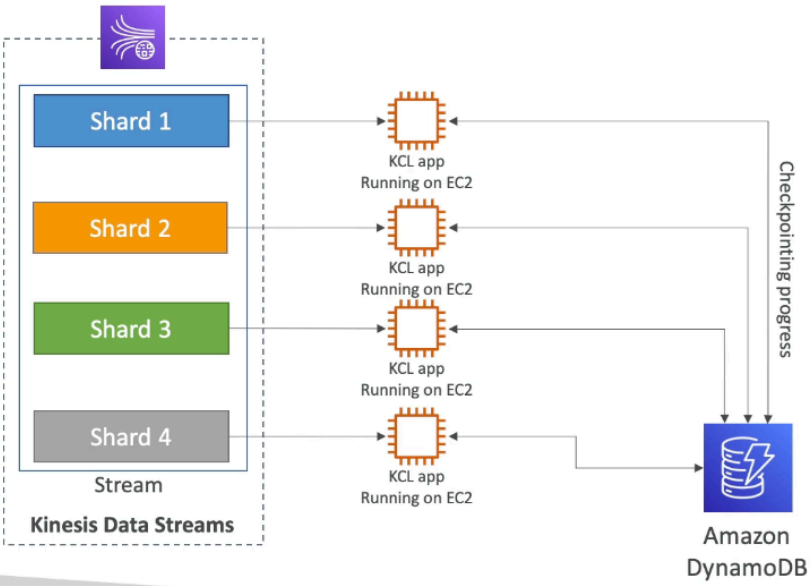
Kinesis Client Library (KCL)

- Java library that helps read record from Kinesis Data Stream with distributed applications sharing the read workload
- Each shard is to be read by one 1 KCL instance
    - 4 shards = max 4 KCL instances
- Progress is checkpointed into DynamoDB (needs IAM access to Dynamo)
    - Track other workers and share the work amongst shards using DynamoDB
- Records are read in order at shard level
- Versions:
    - KCL 1.x supports shared consumer 2.x supports both shared and enhanced

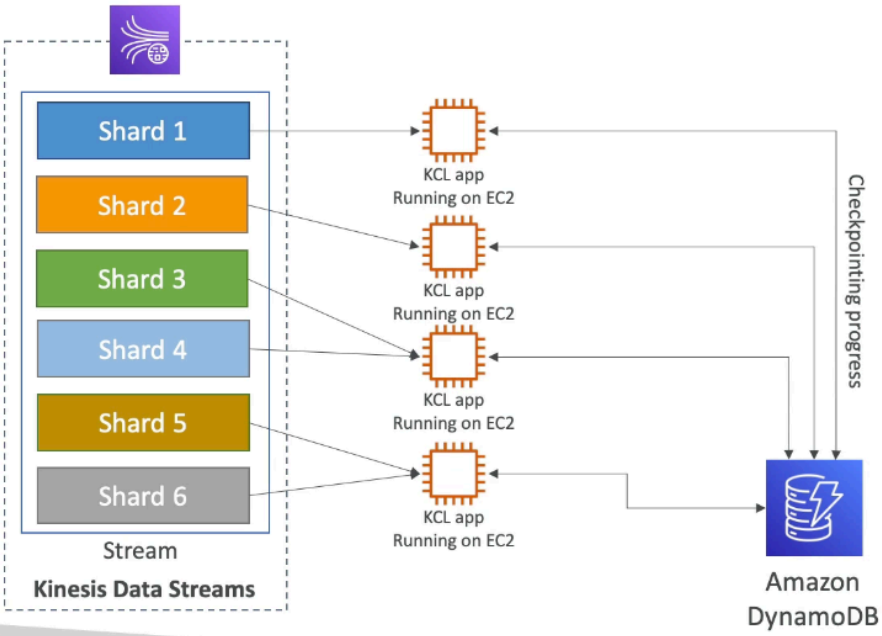
KCL Example: 4 shards



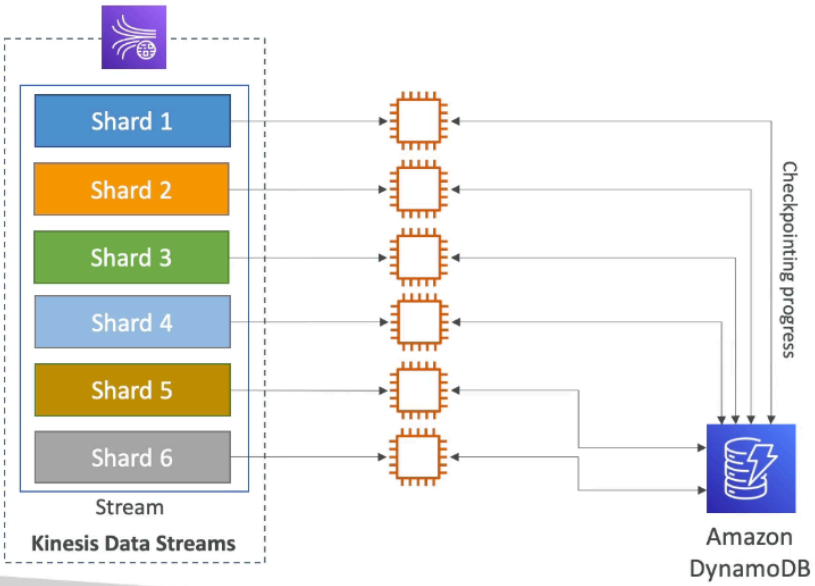
KCL Example: 4 shards, Scaling KCL App



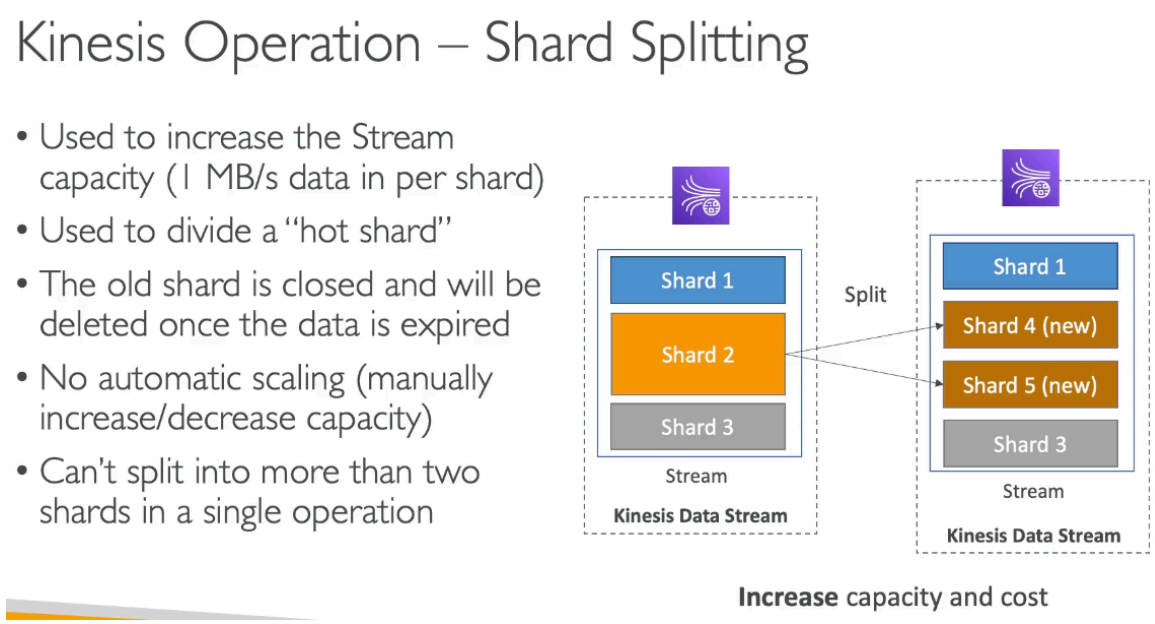
KCL Example: 6 shards, Scaling Kinesis



KCL Example: 6 shards, Scaling KCL App

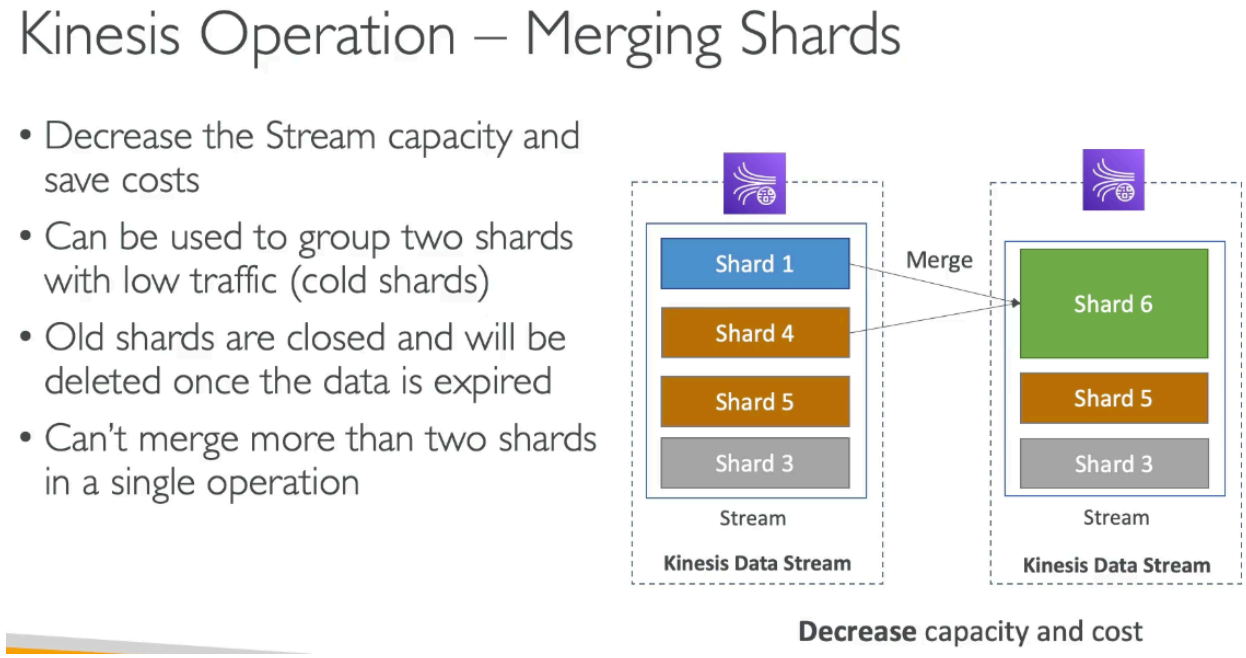


Shard Splitting



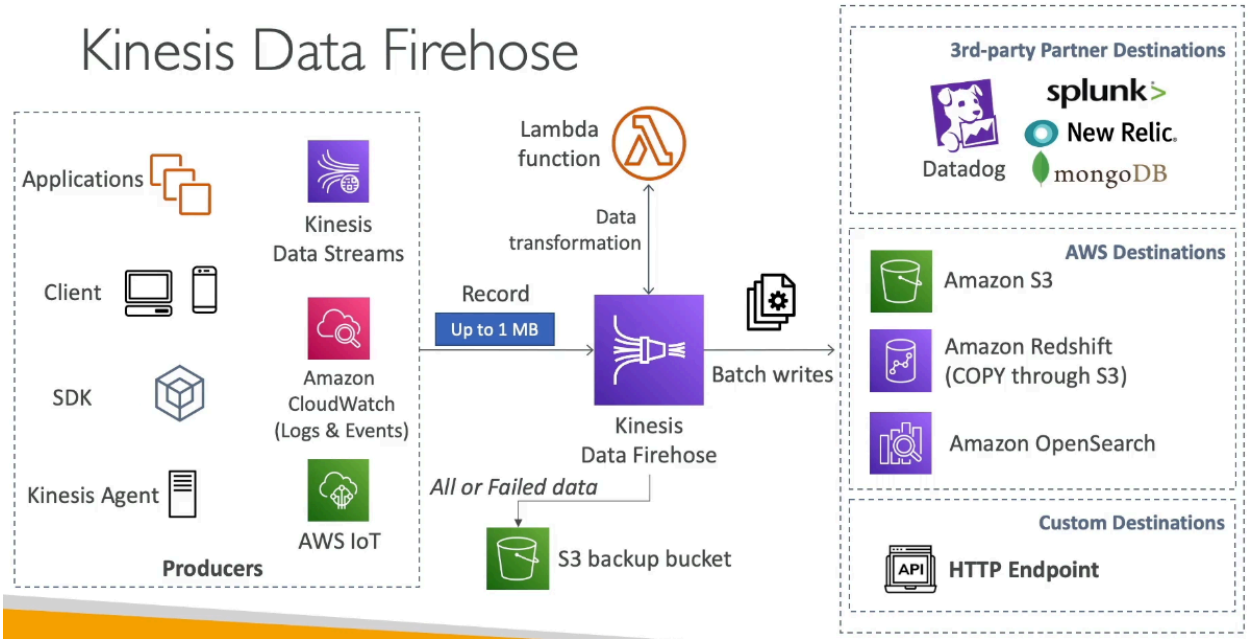
- Used to increase stream capacity (1 MB/s data in per shard) and cost
    - Divides a "hot shard"
- Old shard is closed and will be deleted once data expires
- No automatic scaling
- Cannot split into more than 2 shards in a single operation

Merging Shards



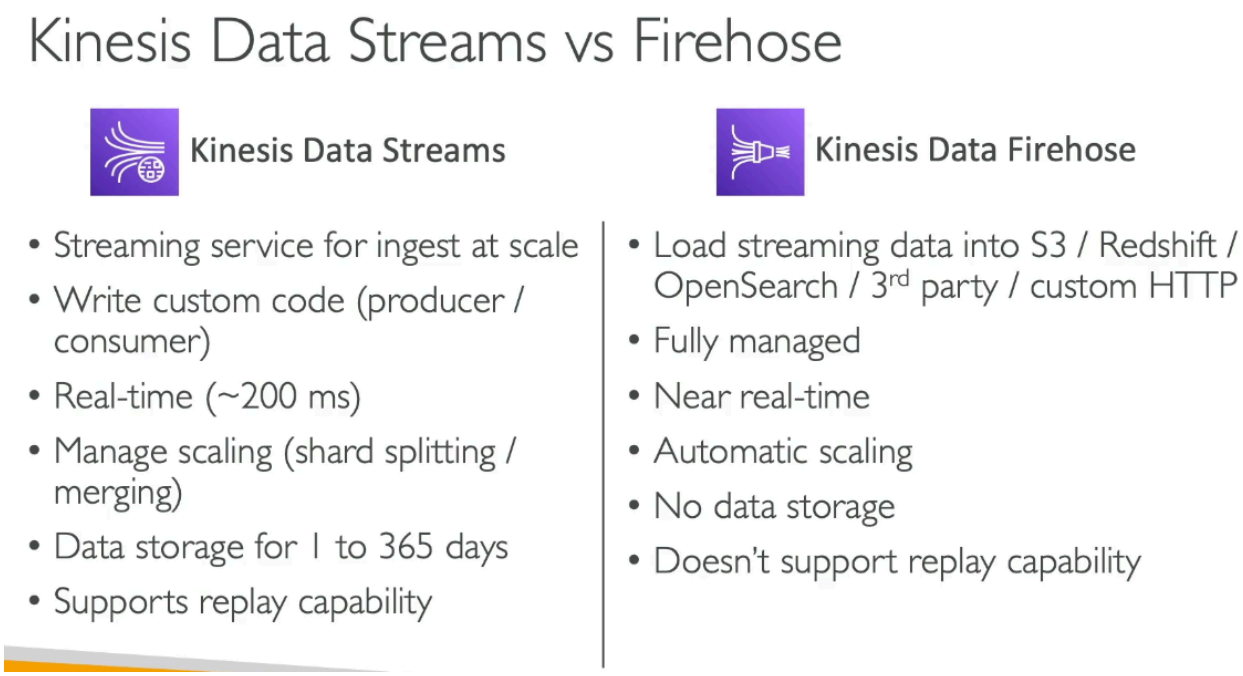
- Decrease stream capacity and save costs
- Group 2 shards with low traffic (cold shards)
    - Old shard is closed and data deleted

Kinesis Data Firehose



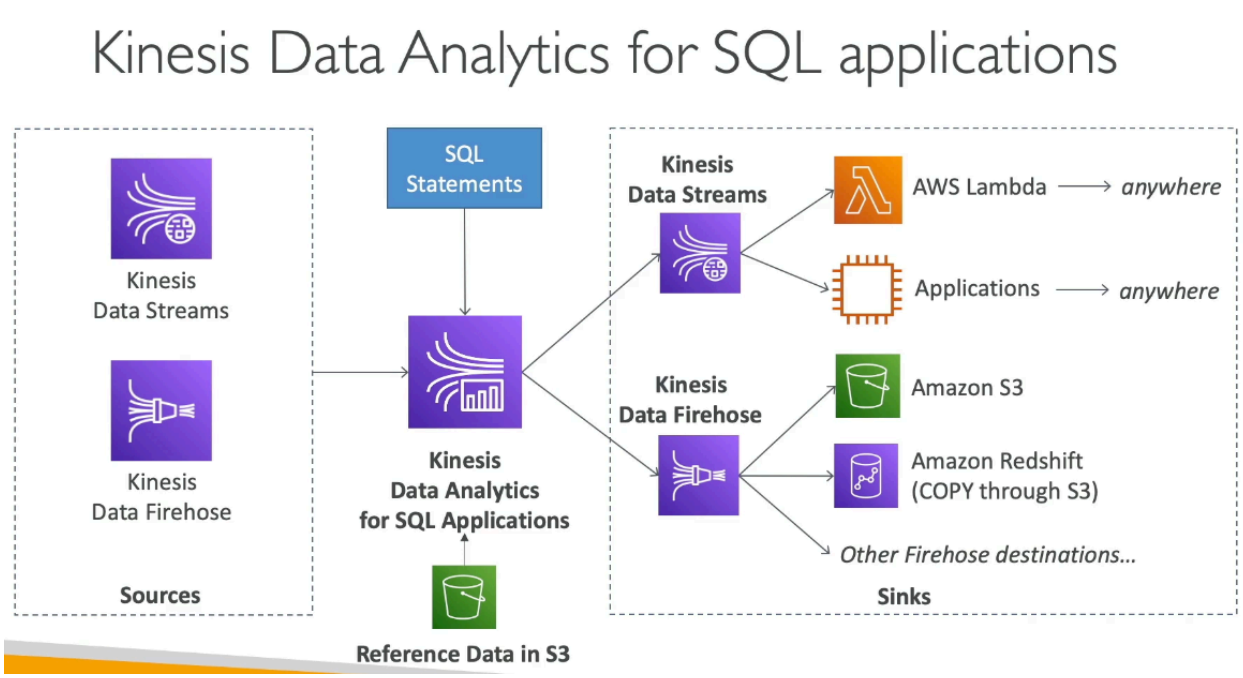
- Takes data from sources (Kinesis Data Streams) and send in batches to S3, Redshift (COPY through S3), Amazon OpenSearch
    - There is a buffer size and buffer interval. If the size isn't filled by the time the interval ends, the buffer is automatically flushed.
- Fully managed service (serverless)
    - Pay for data going through Firehost
    - Near real time
        - Buffering interval from 0 to 900 seconds and buffer size minimum of 1 MB
- Supports many data formats, conversions, transformations, conversions

Kinesis Data Streams vs Firehose



Kinesis Data Analytics

For SQL Applications



Kinesis Data Analytics (SQL application)

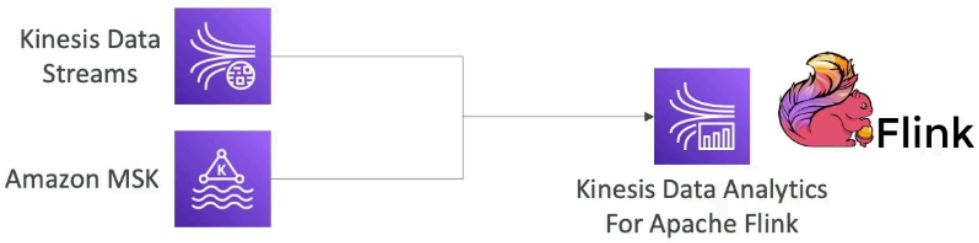
- Real-time analytics on **Kinesis Data Streams & Firehose** using SQL
- Add reference data from Amazon S3 to enrich streaming data
- Fully managed, no servers to provision
- Automatic scaling
- Pay for actual consumption rate
- Output:
  - Kinesis Data Streams: create streams out of the real-time analytics queries
  - Kinesis Data Firehose: send analytics query results to destinations
- Use cases:
  - Time-series analytics
  - Real-time dashboards
  - Real-time metrics

- Real time analytics on Kinesis Data Streams or Firehose using SQL
- Fully managed, no servers, automatic scaling, pay for consumption rate

For Apache Flink

Kinesis Data Analytics for Apache Flink

- Use Flink (Java, Scala or SQL) to process and analyze streaming data



- Run any Apache Flink application on a managed cluster on AWS
  - provisioning compute resources, parallel computation, automatic scaling
  - application backups (implemented as checkpoints and snapshots)
  - Use any Apache Flink programming features
  - Flink does not read from Firehose (use Kinesis Analytics for SQL instead)

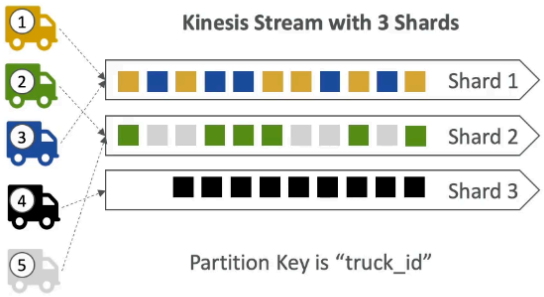
- Java, Scala, SQL to process streaming data
    - Flinks are special apps written as code and can be ran
        - Provisioning of compute resources, parallel computation, automatic scaling
        - Application backups
        - Does not read from Firehose, use SQL instead
- 2 main data sources: Kinesis Data Streams or Amazon MSK

Data Ordering into Kinesis vs SQS FIFO

Ordering with Kinesis

Ordering data into Kinesis

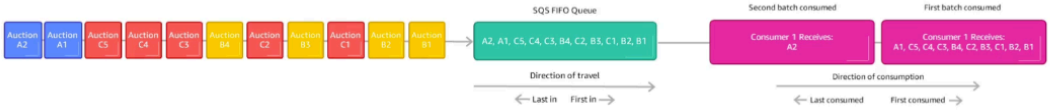
- Imagine you have 100 trucks (truck_1, truck_2, … truck_100) on the road sending their GPS positions regularly into AWS.
- You want to consume the data in order for each truck, so that you can track their movement accurately.
- How should you send that data into Kinesis?
- Answer: send using a "Partition Key" value of the "truck_id"
- The same key will always go to the same shard



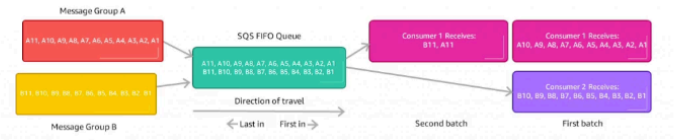
Ordering with SQS

Ordering data into SQS

- For SQS standard, there is no ordering.
- For SQS FIFO, if you don't use a Group ID, messages are consumed in the order they are sent, with only one consumer



- You want to scale the number of consumers, but you want messages to be "grouped" when they are related to each other
- Then you use a Group ID (similar to Partition Key in Kinesis)



Kinesis vs SQS ordering

- Let's assume 100 trucks, 5 kinesis shards, 1 SQS FIFO
- Kinesis Data Streams:
  - On average you'll have 20 trucks per shard
  - Trucks will have their data ordered within each shard
  - The maximum amount of consumers in parallel we can have is 5
  - Can receive up to 5 MB/s of data
- SQS FIFO
  - You only have one SQS FIFO queue
  - You will have 100 Group ID
  - You can have up to 100 Consumers (due to the 100 Group ID)
  - You have up to 300 messages per second (or 3000 if using batching)

SQS vs SNS vs Kinesis

SQS vs SNS vs Kinesis

SQS:

- Consumer "pull data"
- Data is deleted after being consumed
- Can have as many workers (consumers) as we want
- No need to provision throughput
- Ordering guarantees only on FIFO queues
- Individual message delay capability

SNS:

- Push data to many subscribers
- Up to 12,500,000 subscribers
- Data is not persisted (lost if not delivered)
- Pub/Sub
- Up to 100,000 topics
- No need to provision throughput
- Integrates with SQS for fan-out architecture pattern
- FIFO capability for SQS FIFO

Kinesis:

- Standard: pull data
  - 2 MB per shard
- Enhanced-fan out: push data
  - 2 MB per shard per consumer
- Possibility to replay data
- Meant for real-time big data, analytics and ETL
- Ordering at the shard level
- Data expires after X days
- Provisioned mode or on-demand capacity mode

Section 20: AWS Monitoring & Audit: CloudWatch, X-Ray and CloudTrail

Monitoring Overview

- Check for application outages and latency
- CloudWatch
    - Metrics, logs, events, alarms
- X Ray
    - Troubleshoot application performance and errors
    - Distributed tracing of microservices
- CloudTrail
    - Internal monitoring of API calls, audit changes of AWS Resources by users

CloudWatch Metrics

- Metrics for every service, can create CloudWatch dashboards
- Metric is a variable to monitor (CPU Utilization, NetworkIn…)
- Metrics belong to namespaces
- Dimension is an attribute of a metric (instance ID, environment, etc…)
    - Up to 30 dimensions per metric
- Metrics have timestamps

EC2 Detailed Monitoring

- Detailed monitoring disabled by default, but can be enabled for 1 minute periods for metrics
- EC2 instance metrics have metrics every 5 min, but with detailed monitoring it's faster
- EC2 Memory usage is by default not pushed, must be pushed from inside the instance as a custom metric)

CW Custom Metrics

- PutMetricData API call
    - Add dimensions (attributes) and metric resolution (StorageResolution API)
        - Standard resolution: 1 min, High res: 1 - 30 sec with higher cost
- Accepts metric data points 2 weeks in the past and 2 hours in the future

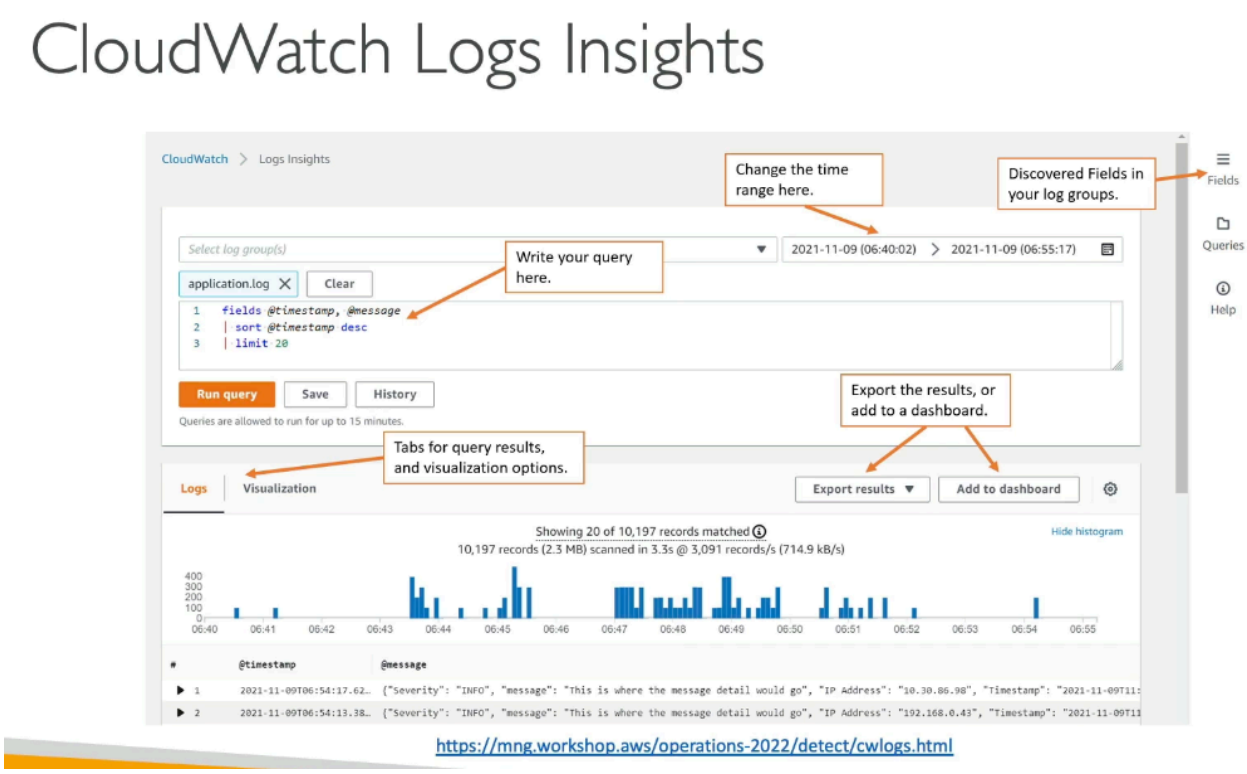
CloudWatch Logs

- Log group: name representing the application
- Log stream: instances within app / log files / containers
- Log expiration policies (never, 10 years)
- Can be sent to S3, Kinesis, Lambda, OpenSearch
- Encrypted by default with KMS based encryption possible

Log Sources

- SDK, CloudWatch Unified Agent, Beanstalk ECS, Lambda, VPC, API GW, CloudTrail, Route 53

Logs Insights



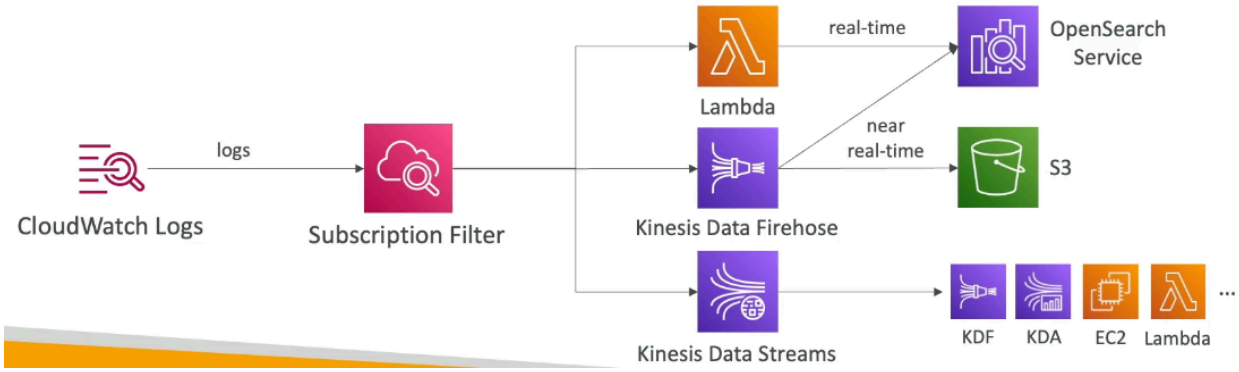
https://mng.workshop.aws/operations-2022/detect/cwlogs.html

- Can query logs to search and analyze; all fields automatically found, not real time
- Can save queries to CW dashboards and query multiple log groups in different AWS accounts

S3 Export

- Log data can take up to 12 hours to be available for export via CreateExportTask API call, thus not real time

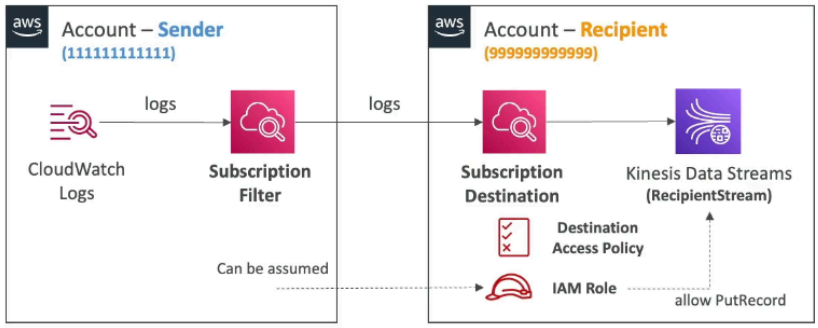
Log Subscriptions



- Real time log events to send to Kinesis or Lambda
- Subscription filter to filter logs delivered to destinations

CloudWatch Logs Subscriptions

- **Cross-Account Subscription** – send log events to resources in a different AWS account (KDS, KDF)



IAM Role (Cross-Account)

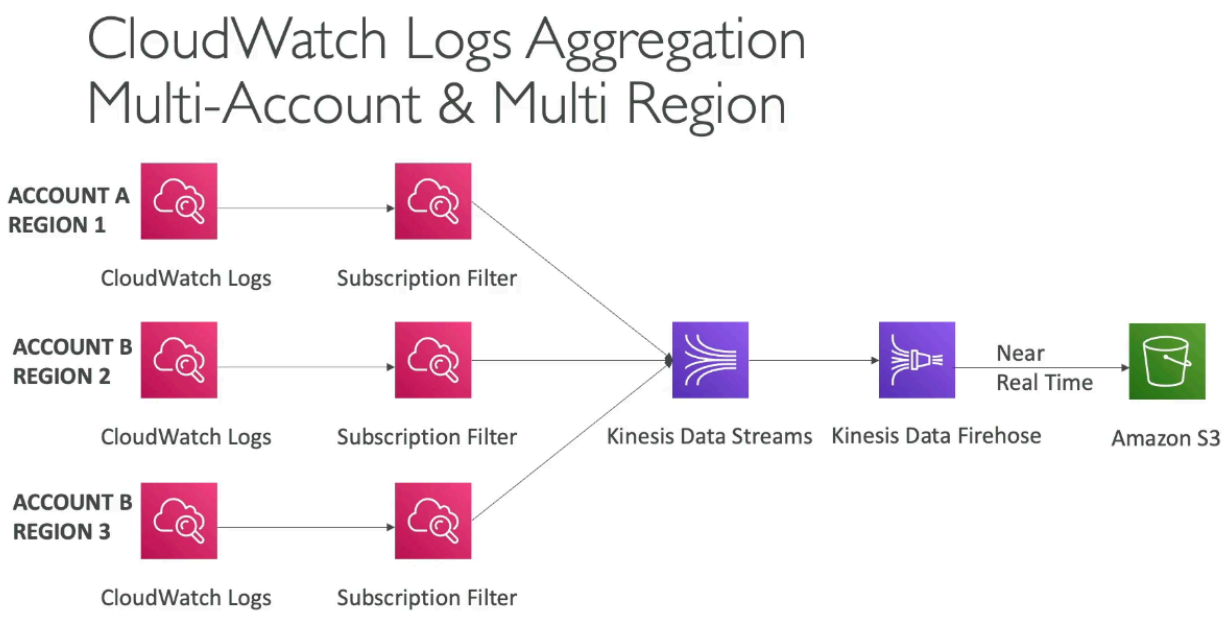
```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:us-east-1:
999999999999:stream/RecipientStream"
    }
  ]
}
```

Destination Access Policy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "AWS": "111111111111"
      },
      "Action": "logs:PutSubscriptionFilter",
      "Resource": "arn:aws:logs:us-east-1:999999999999:
destination:testDestination"
    }
  ]
}
```

- Cross Account Subscription: sends log events to different AWS account
  - The sender needs a Destination Access Policy to send logs via filter to another account.

Logs Aggregation

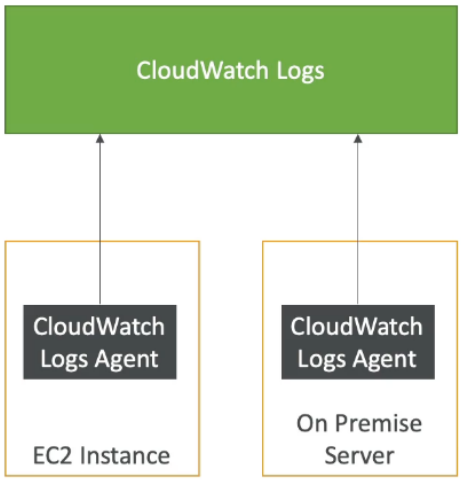


- Can send logs from different accounts in different regions via subscription filter to 1 account (to Kinesis)

CloudWatch Agent & CloudWatch Logs Agent

CW Logs for EC2

- By default no logs go from EC2 to CloudWatch. A CW agent on EC2 will push log files (need IAM permissions); can be on premise too



CW Logs Agent & Unified Agent

- For VMs
- CW Logs Agent
  - Older and can only send to CW logs
- Unified Agent
  - Collect additional system metrics and send to CW
    - CPU, Disk metrics, RAM, etc…
    - More granularity
  - Centralized configuration using SSM parameter store

Metric Filter

- Logs can use filter expressions and can be created to trigger alarms (up to 3 dimensions for metric filter)
- Filers do not retroactively filter data. It only publishes the metric data points for events after the event is created

CloudWatch Alarms

- Trigger notifications for any metric based on sampling, %, max, etc…
- Alarm States: ok, insufficient data, alarm for a period of time (can be high resolution in seconds)
    - High resolution alarm can be 10 or 30 second intervals, where regular alarms are 1 minute multiples
- Alarm Targets
    - Stop, terminate, reboot, etc… EC2 instance, trigger auto scaling, SNS notification
- Alarms can be created based on Log metric filters
- Test alarms via CLI set-alarm-state call

Composite Alarms

- Alarms are single metric, but composite alarms monitor states of multiple other alarms via AND and OR conditions
    - Reduces alarm noise

EC2 Instance Recovery

- Status check via instance status or system status and can recover
    - Recovery will have same private, public, elastic IP, metadata, placement group

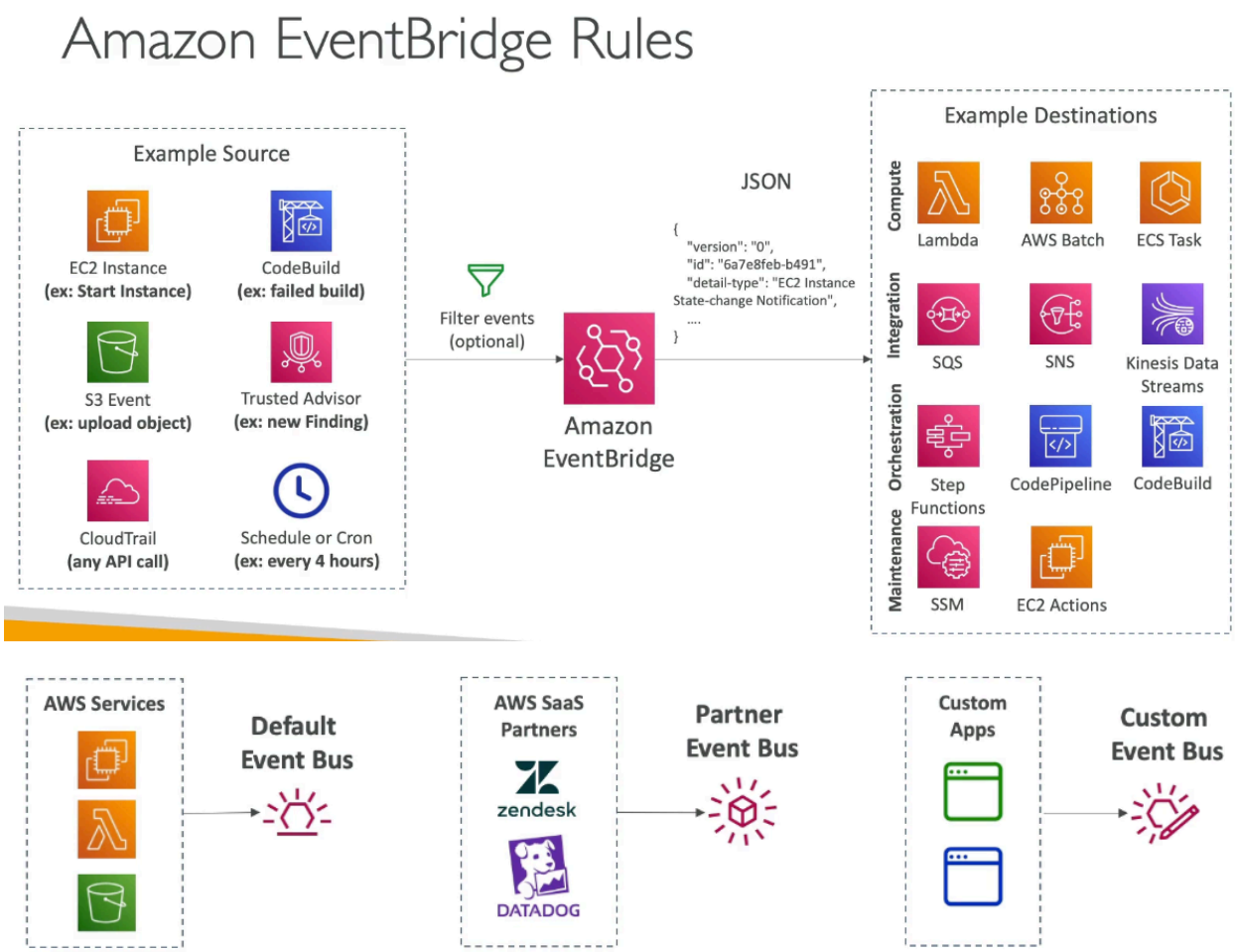
CloudWatch Synthetics Canary

- Configurable script to monitor API, URL to reproduce programmatically E2E testing
    - Check availability, latency, store load time data and UI screenshots
- Integration with CloudWatch Alarms

Blueprints

- Heartbeat monitor, API Canary, Broken link checker, Visual monitoring, Canary Recorder (used with CW Synthetics Recorder to record website actions), GUI workflow builder

EventBridge



- Default event bus or partner event bus with either AWS services or 3rd party
    - Custom event bus for custom rules
    - Event buses accessed by other AWS accounts using resource based policies
- Schedule cron jobs (scheduled scripts), event pattern (event rules to react to a service doing something), trigger lambda functions, send SNS/SQS messages
    - Archive events (all/filter) sent to event bus (indefinitely or set period) to replay archived events

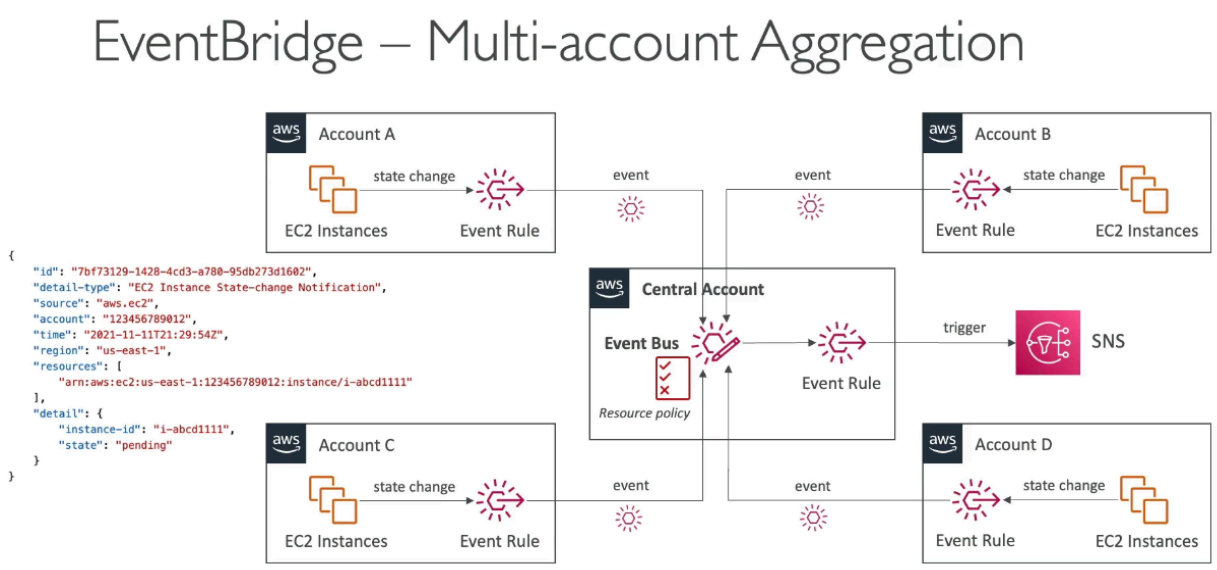
Scheme Registry

- EventBridge can analyze events in your bus and infer the schema
- Schema registry allows you to generate code for ap to know in advance how data is structured in event bus
    - Can be versioned

Resource Based Policy

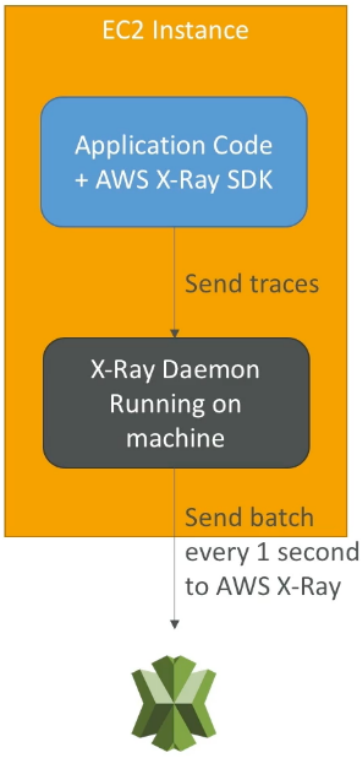
- Manage permissions for specific event bus, for example allow events from another AWS account or region → typically to aggregate all events from AWS Organization into 1 account

Event Bridge – Multi Account Aggregation



- Create event rules in branched accounts to send to a centralized event bus. Centralized event bus must have a resource policy to accept

AWS X Ray



- Visual analysis of applications for troubleshooting performance, understanding architecture dependencies, pinpoint issues and errors
    - Collects all data and generates service map from segments and traces
- Leverages tracing → E2E way to follow a request
    - Each component dealing with the request adds its own trace. Tracing is made of segments. Annotations can be added to traces to provide extra info
- Security: IAM, encryption
- Enabled via code from SDK:
    - Few code modification, SDK will capture AWS service calls, HTTP / HTTPS requests, DB calls, queue calls

- Install X Ray daemon or enable X Ray AWS integration
    - Daemon works as low level UDP packet interceptor
    - AWS services already run daemon
    - Must have IAM rights to write data to X Ray

Troubleshooting

- If not working on EC2
    - ENsure EC2 IAM role has proper permissions
    - Ensure EC2 instance is running X Ray daemon
- To enable on Lambda:
    - Ensure IAM execution role with proper policy (AWS X-Ray Write Only Access)
    - Ensure x-ray imported in code
    - Enable lambda x-ray active tracing

X Ray: Instrumentation and Concepts

- Instrumentation: the measure of product's performance, diagnose error, and to write trace info
    - Use X Ray SDK (only configuration changes)

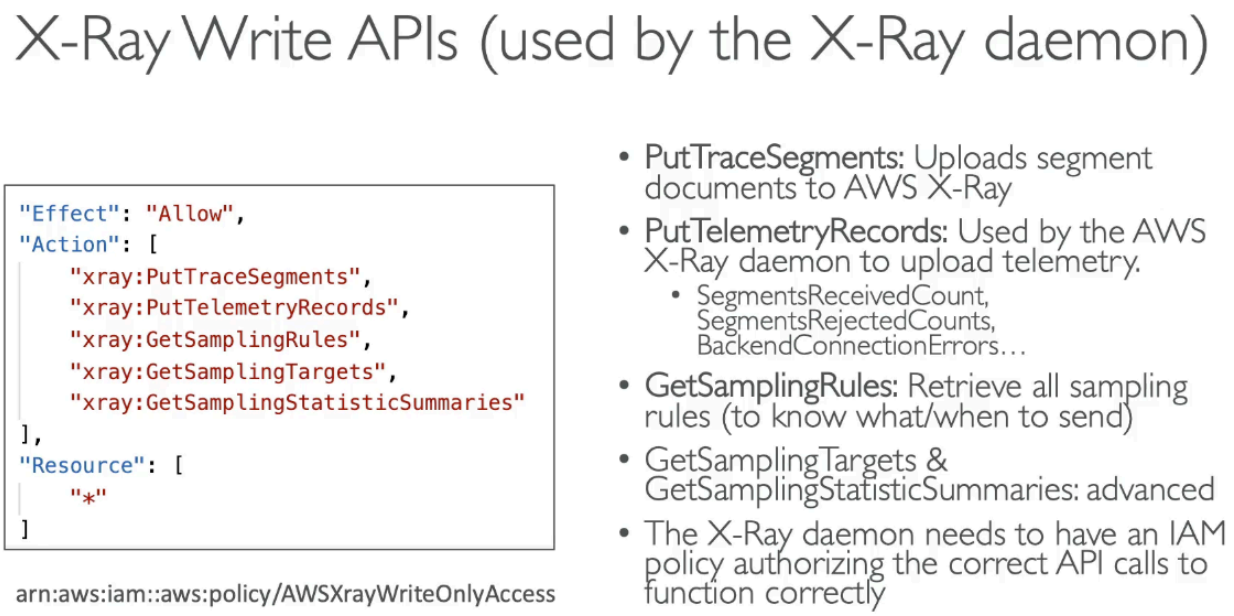
- Segments: each application / service will send them
- Subsegments: if you need more details in your segment
- Trace: segments collected together to form E2E trace
- Sampling: decrease the amount of requests sent to X-Ray, reduce cost
- Annotations: key pair values, indexed, used with filters
- Metadata: key value pairs, not indexed, not used for searching
- X Ray daemon / agent has config to send traces cross account (need IAM permissions)

Sampling Rules

- Control amount of data recorded
- Default: SDK records first request each second and 5% of any additional request
    - One request / second is the reservoir, ensures at least 1 trace is recorded each second as long as service has requests
    - 5% is the rate which additional requests after reservoir size are sampled
- Can create own rules with reservoir and rate
    - Reservoir: > 1
    - Rate: 0 → 1, 1 meaning everything is captured 100%

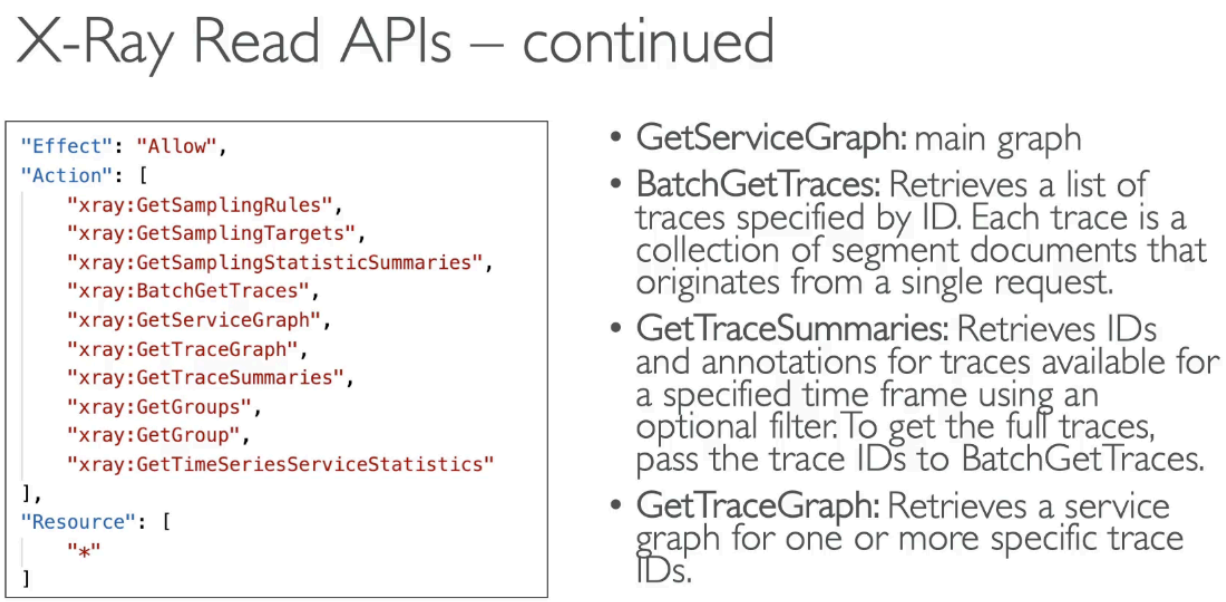
X Ray Write APIs

- Used by daemon to write data into x ray service



- The daemon needs GetSamplingRules to know its own rules of what to do

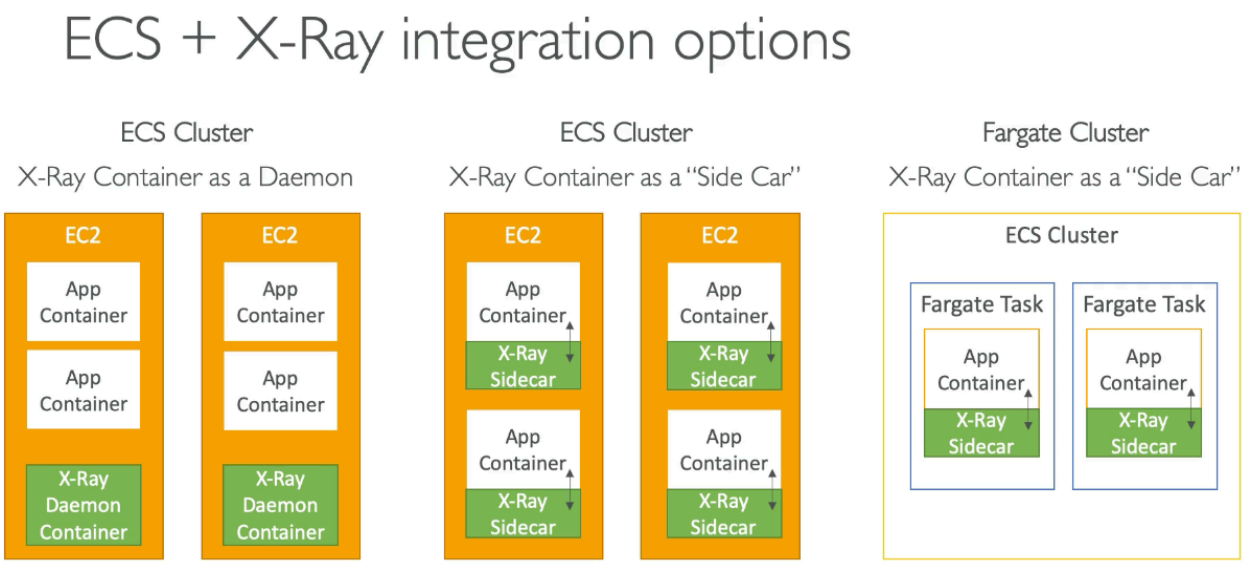
X Ray Read APIs

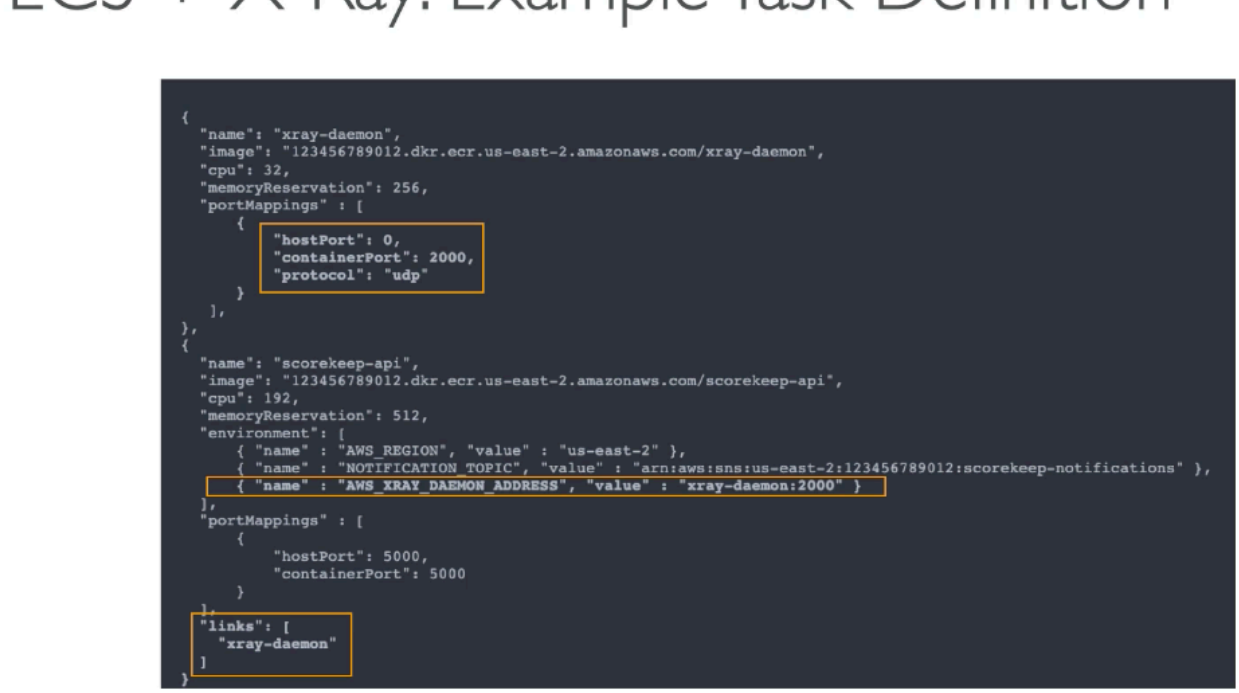


X Ray with Beanstalk

- Already includes daemon, can be run with console or config file in .ebextensions/xray-daemon.config (with IAM permissions and SDK in code)
- X Ray Daemon not provided for multi-container docker

X Ray + ECS

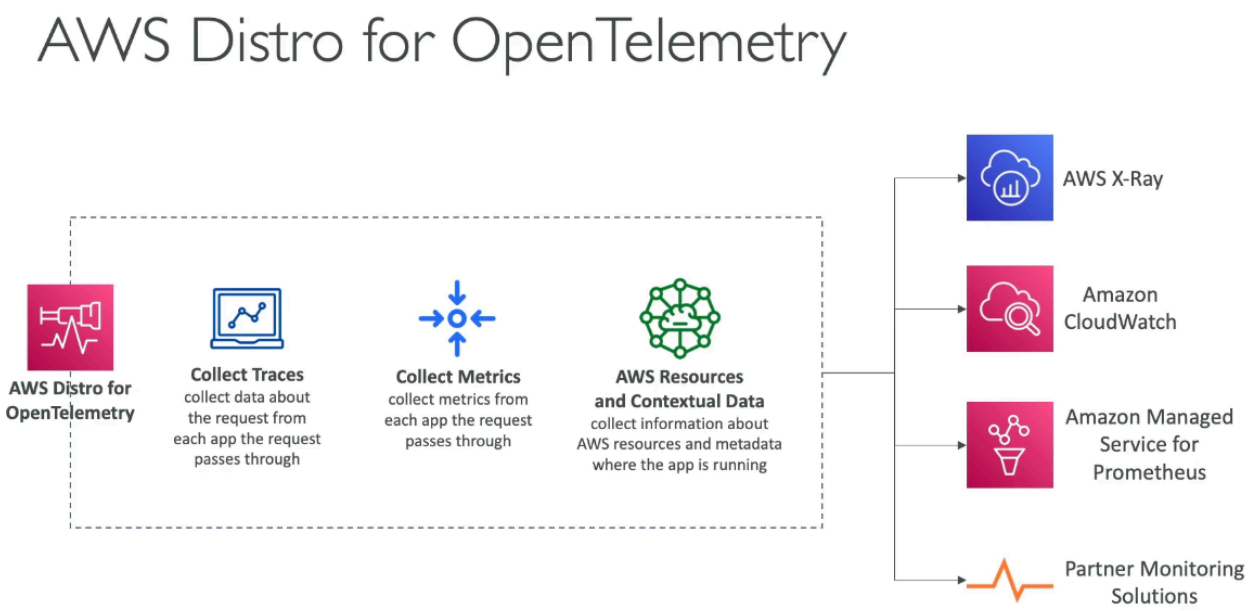




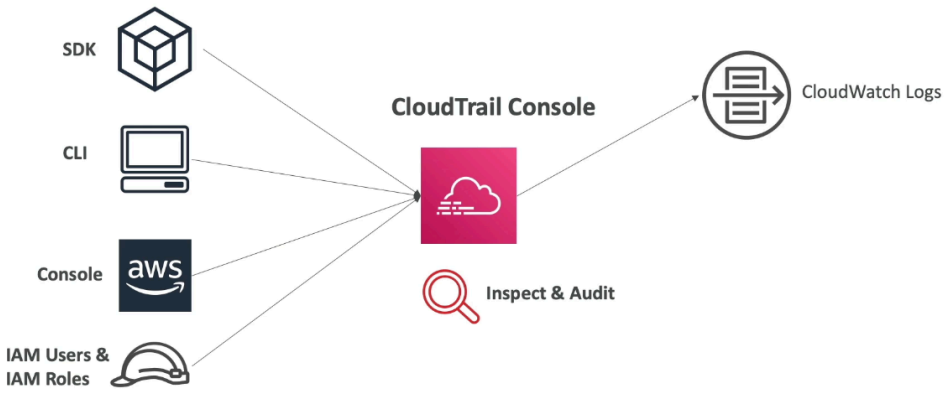
AWS Distro for OpenTelementry

- Secure, prod ready AWS supported version of the OpenTelementry project
    - Open source version of X Ray

    - Migrate from X Ray to AWS Distro to standardize with open source APIs from Telemetry or send traces to multiple destinations simultaneously
- Provides single set of APIs, libraries, agents, and collector services
- Collects distributed traces and metrics from your apps
- Collects metadata from AWS resources
- Auto instrumentation agents to collect traces without changing code
- Sends traces and metrics to AWS services and partners
- Instrument apps running on AWS or on premise

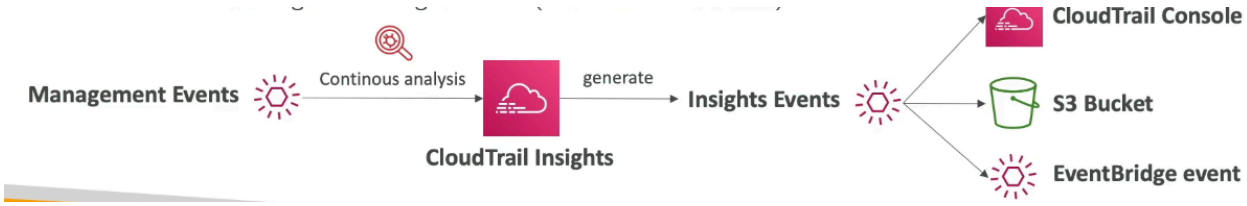


CloudTrail



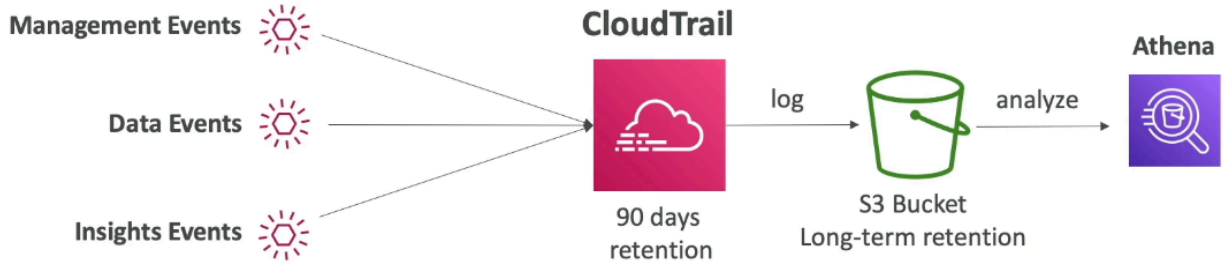
- Governance, compliance for AWS account (enabled by default)
- Gets history of events / API calls made within AWS account where CloudTrail can be sent into CloudWatch or S3
    - Trail applies to all regions by default or single region

CloudTrail Events



1. Management events: operations performed on resources
    a. Configuring security, logging, etc…
    b. Default trails configured to log management events
    c. Can separate read (no modification) and write events (modify resources)
2. Data events
    a. By default data events not logged (high volume operations)
        i. S3 object level activity, can separate read and write events
        ii. AWS Lambda execution activity (invoke API)
3. CloudTrail Insights Events:
    a. Enable to detect unusual activity in account
        i. Inaccurate resource provisioning, hitting service limits, IAM actions, etc…
    b. Analyzes normal management events for a baseline and continuously analyze write events for unusual patterns
        i. Anomalies show in CloudTrail and event sent to S3 and EventBridge

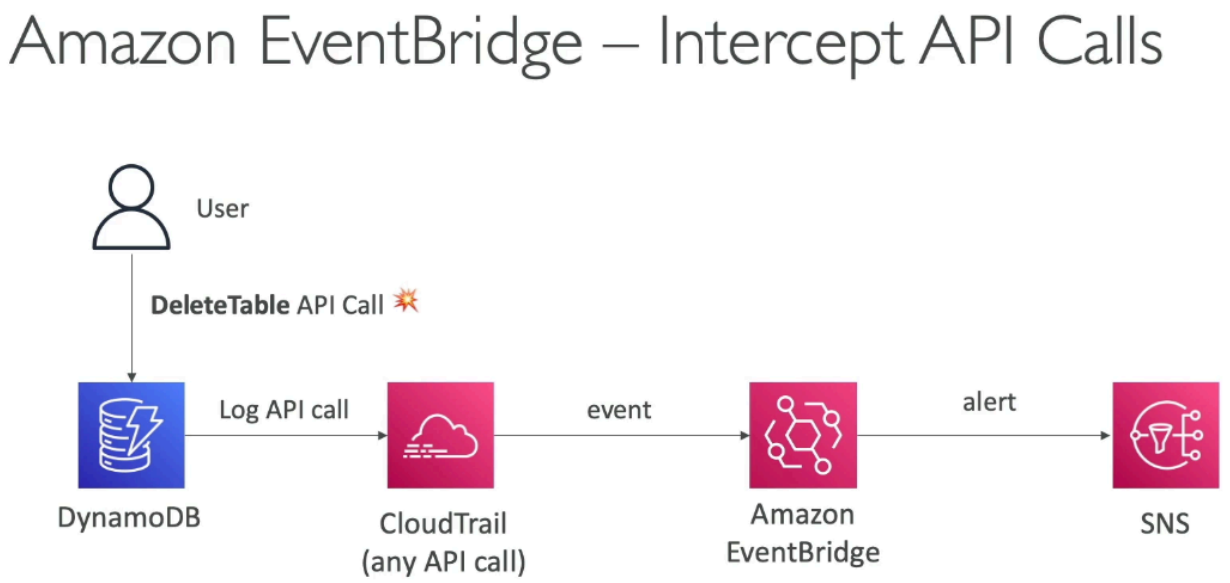
Retention

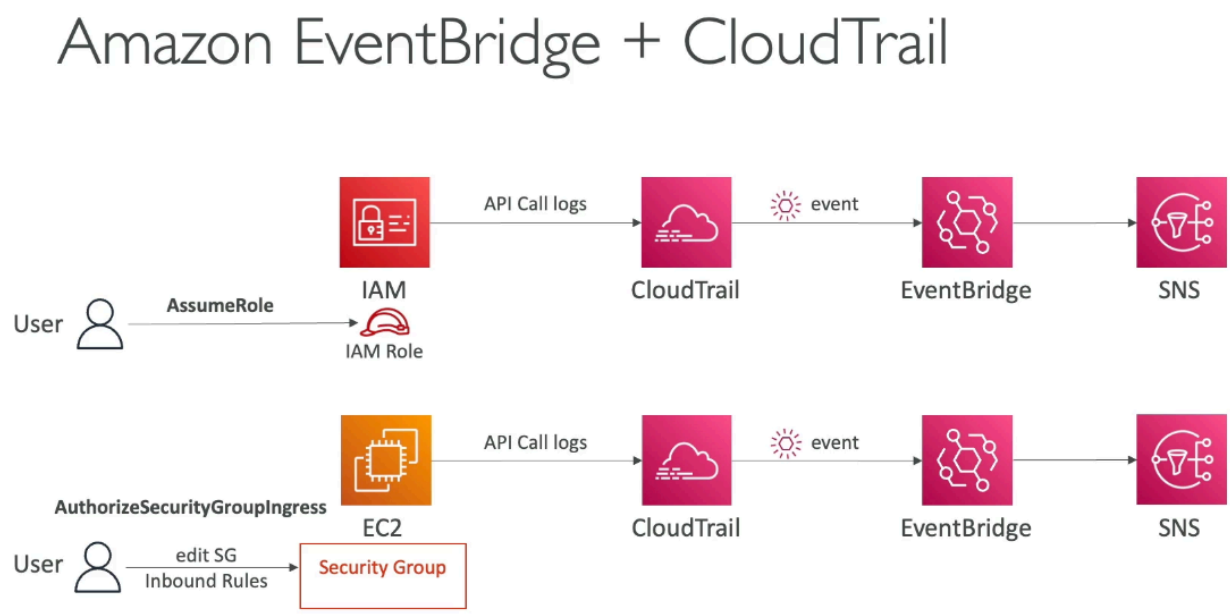


- Stored for 90 days by default, but to keep beyond use S3 and use Athena

CloudTrail + EventBridge

- Intercept API calls





CloudTrail vs CloudWatch vs X Ray

- CloudTrail is for API monitoring, CloudWatch is for metrics, X ray is for debugging

CloudTrail vs CloudWatch vs X-Ray

- CloudTrail:
  - Audit API calls made by users / services / AWS console
  - Useful to detect unauthorized calls or root cause of changes
- CloudWatch:
  - CloudWatch Metrics over time for monitoring
  - CloudWatch Logs for storing application log
  - CloudWatch Alarms to send notifications in case of unexpected metrics
- X-Ray:
  - Automated Trace Analysis & Central Service Map Visualization
  - Latency, Errors and Fault analysis
  - Request tracking across distributed systems

Section 21: AWS Serverless: Lambda

Serverless Intro

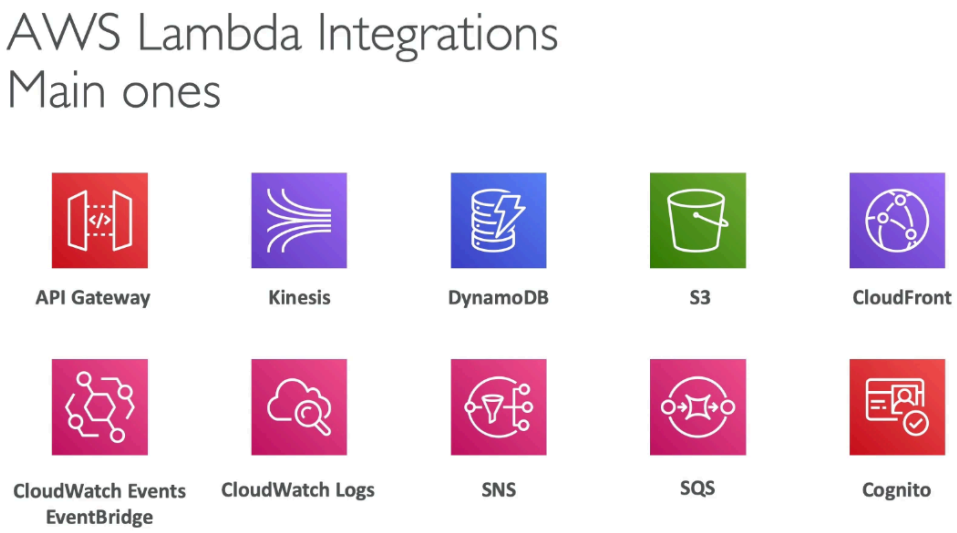
- No server management / provisionment
- Lambda, DynamoDB, Cognito, API GW, S3, SNS / SQS, Kinesis Data Firehose, Aurora Serverless, Step Functions, Fargate

Lambda Overview

- Virtual functions, limited by time (short executions), running on demand, automated scaling
- Max timeout of 15 minutes

Benefits:

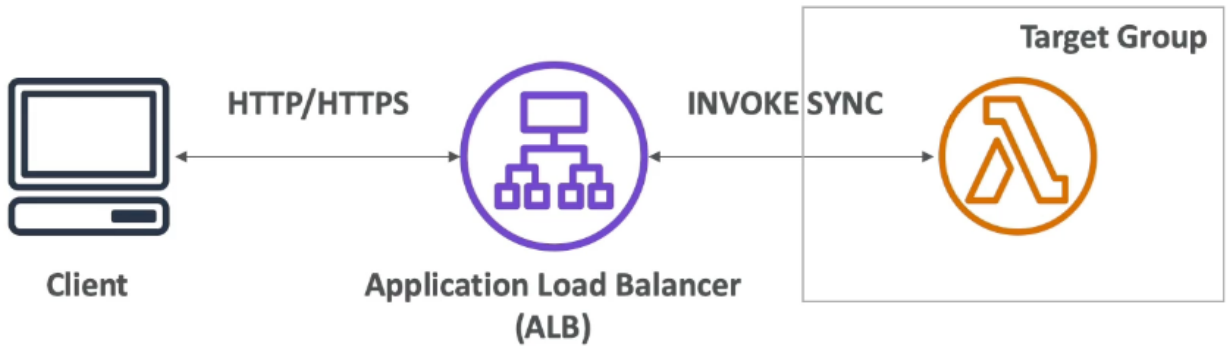
- Easy pricing based on pay per request and compute time + generous free tier
- Integration with AWS services, programming languages, monitoring
    - Custom runtime API for any language, community supported and Lambda container image
- Increasing RAM, increases CPU and network functionality



Lambda Synchronous Invocations

- Synchronous: CLI, SDK, API GW, ALB
  - Results returned right away, error handling must happen client side (retries, exponential backoff, etc…)
- User invoked:
  - ELB (ALB), API GW, CloudFront (Lambda @ Edge)
- Service invoked:
  - Amazon Cognito, Step Functions

Lambda & ALB

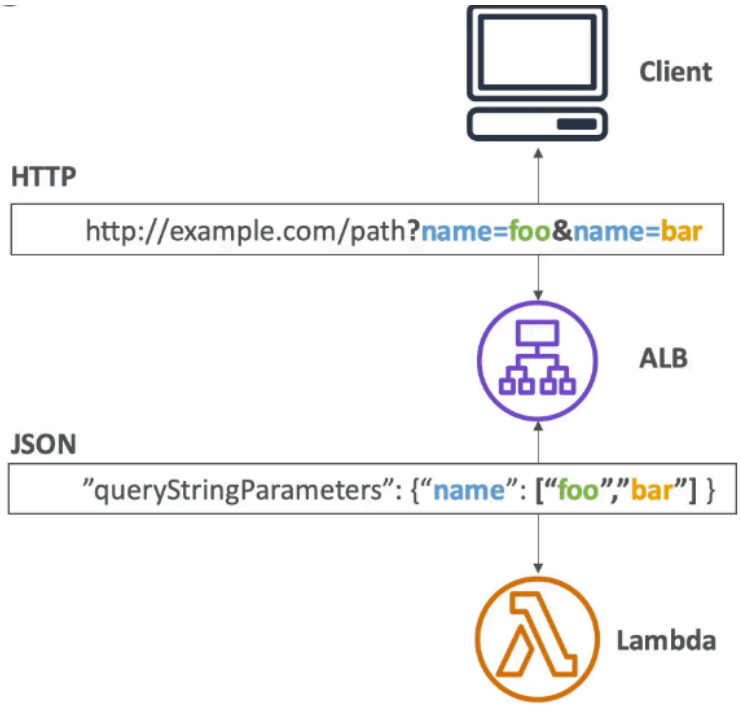


- Expose lambda function as HTTP(S) endpoint via ALB or API GW
  - Lambda function must be registered in a target group, where the lambda function reside
- From the ALB to lambda, HTTP requests convert to JSON. From lambda to ALB, JSON to HTTP

ALB Multi-Header Values

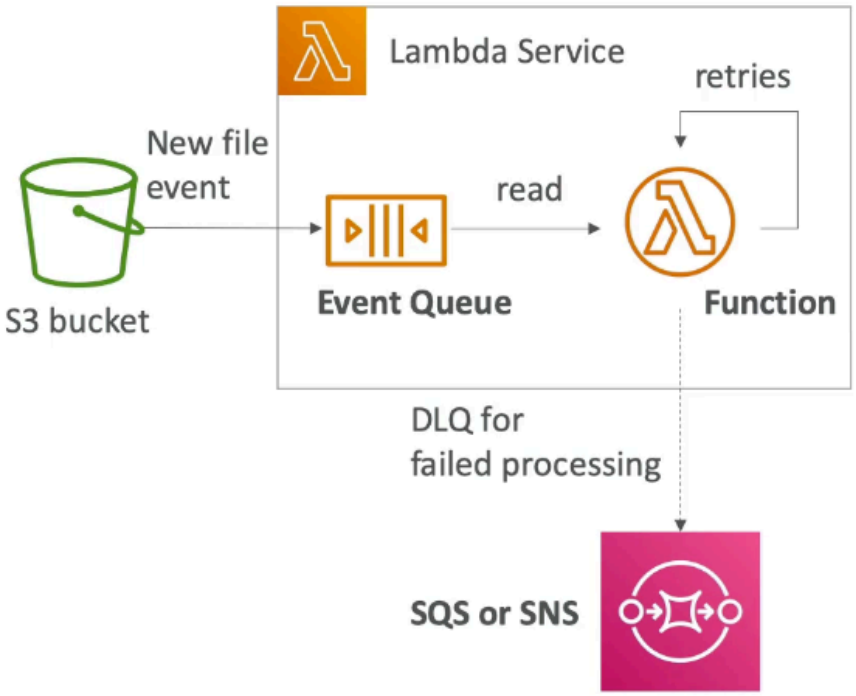
- ALB can support multi header values (setting)
- HTTP headers and query string parameters that are sent with multiple values are shown as arrays within the Lambda event and response objects

- When using ALB + lambda, the return body must have a “body” with HTML to display text. Otherwise, it will download the lambda response



Asynchronous Invocations & DLQ

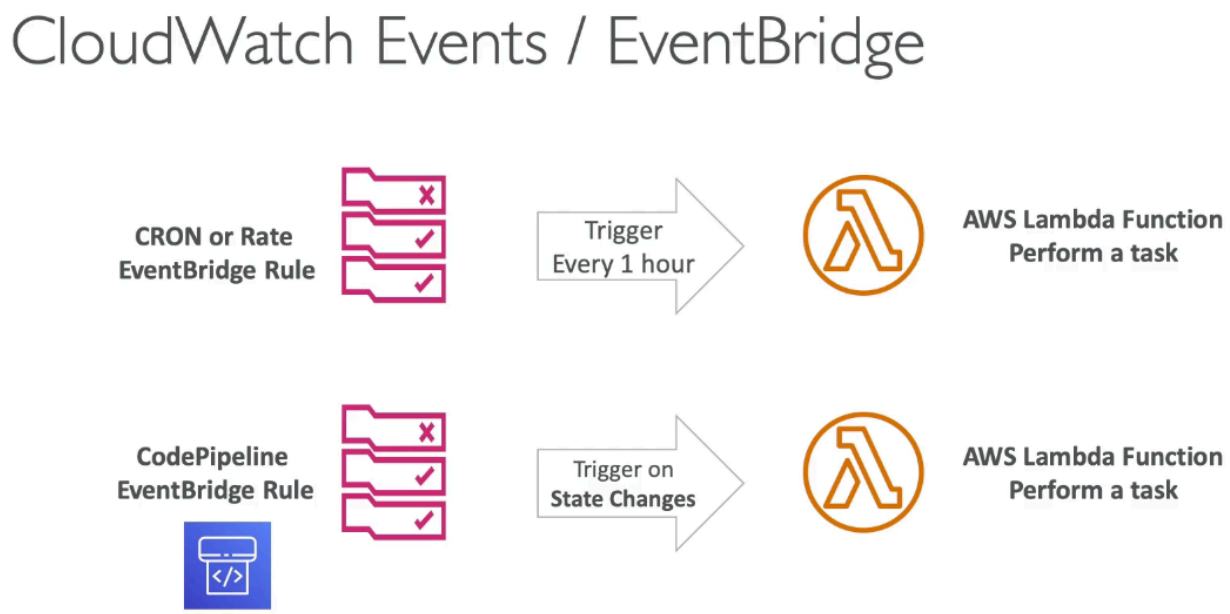
- S3, SNS, CloudWatch Events
- Events are placed in an Event queue which the lambda function reads
    - Lambda attempts to retry on errors: 3 retries, first fail immediately redone, 1 minute wait after 1st, then 2 minute wait (5 min total)



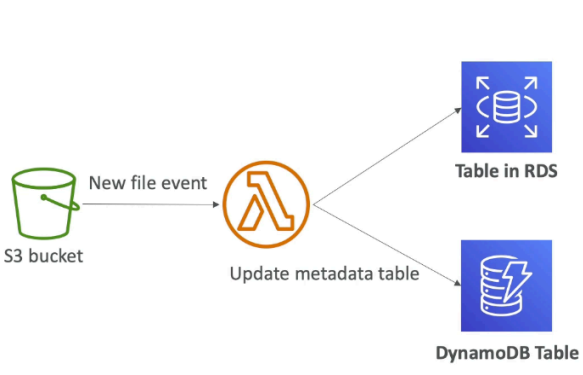
- Make sure the processing is idempotent (in case of retries, the result is the same)

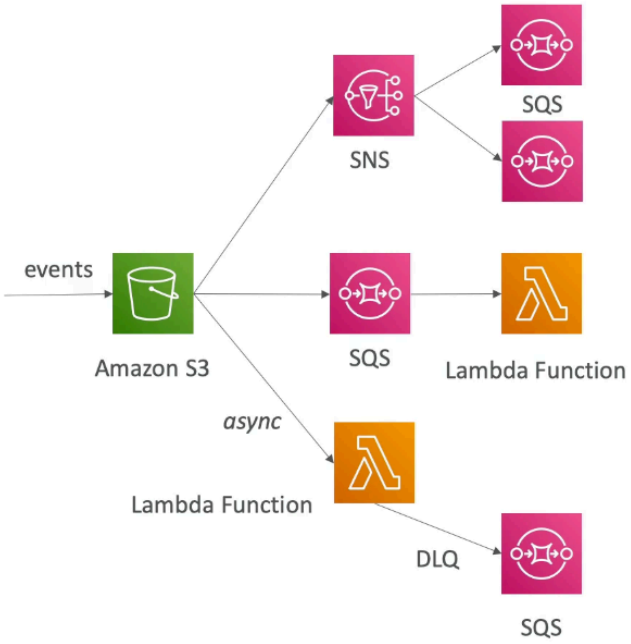
- If functions are retried, duplicate log entries are in CloudWatch logs
    - Don't know if they fail, status code 202 for asynchronous calls
- DLQ (dead letter queue) – SNS or SQS for failed processing (need IAM permissions)
- Asynchronous invocations allow you to speed up processing if you don't need to wait for results

Lambda & CloudWatch Events / EventBridge



Lambda & S3 Event Notifications

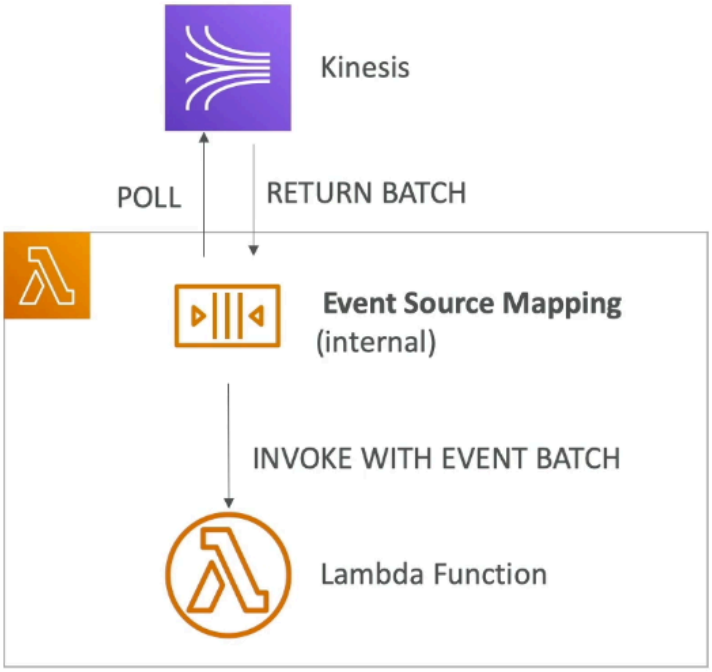




- Deliver events in seconds, but can take longer
- If 2 writes made to a non-versioned object at the same time, it is possible only a single event notification will be sent

- If you want to ensure that an event notification is sent for every successful write, you can enable versioning on your bucket

Lambda Event Source Mapping



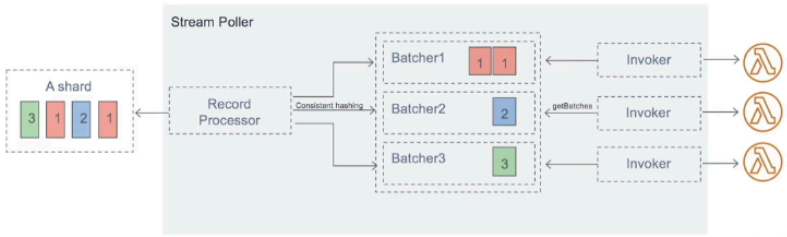
- Kinesis Data Streams, SQS / SNS FIFO queue, DynamoDB streams
- Common denominator: records need to be polled from the source, thus lambda function is invoked synchronously

Streams & Lambda (Kinesis & DynamoDB)

- Event source mapping creates an iterator for each shard, processes items in order at shard level
    - Starts with new items from beginning of the shard or from timestamp
- Processed items aren't removed from stream for other consumers to read
- Low traffic: batch window to accumulate records before processing
- Process multiple batches in parallel for high traffic
    - Up to 10 batches per shard with in order processing guaranteed for each partition key

Streams & Lambda (Kinesis & DynamoDB)

- An event source mapping creates an iterator for each shard, processes items in order
- Start with new items, from the beginning or from timestamp
- Processed items aren't removed from the stream (other consumers can read them)
- Low traffic: use batch window to accumulate records before processing
- You can process multiple batches in parallel
  - up to 10 batches per shard
  - in-order processing is still guaranteed for each partition key,



https://aws.amazon.com/blogs/compute/new-aws-lambda-scaling-controls-for-kinesis-and-dynamodb-event-sources/

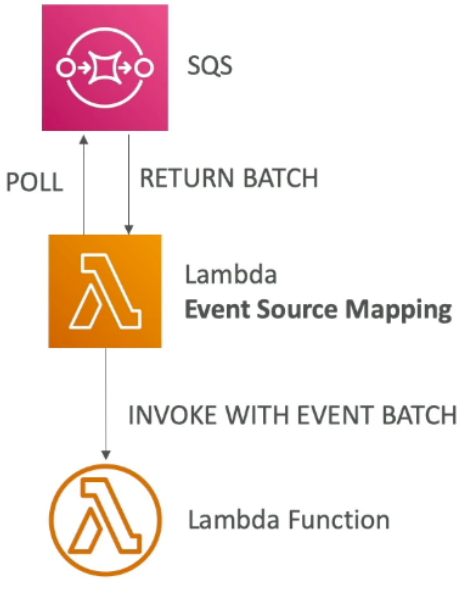
- Basically 1 shard can be split into several batches, with a lambda function processing each batch individually. In order processing is guaranteed via partition keys

Error Handling

- By default, when errors happen the entire batch is reprocessed until the function succeeds or items in the batch expire
  - To ensure in order processing, processing for the affected shard is paused until the error is resolved
- Configure the event source mapping to:
  - Discard old events
    - Go to a destination
  - Restrict retry count
  - Split the batch on error (to work around lambda timeout issues)

Lambda Queues SQS / SQS FIFO

- Event Source Mapping will poll SQS with Long Polling of batch size (1-10 messages)
- Set queue visibility timeout to 6x timeout of lambda function
- To use a DLQ, set the SQS queue not Lambda. Or use a Lambda Destination
  - DLQ for lambda is for async invocations



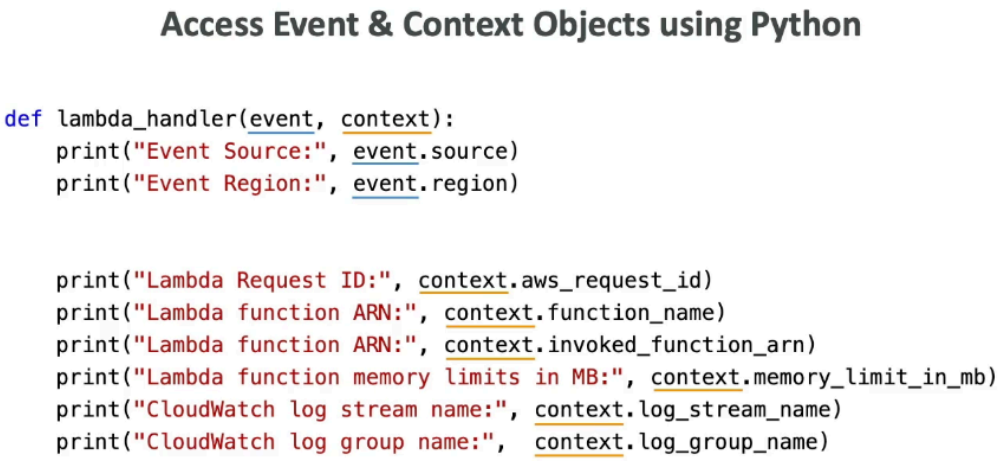
Queues & Lambda

- Lambda supports in order processing for FIFO, scaling up to the # of active message groups
- Standard queues items aren't necessarily processed in order
    - Lambda scales up to processes a standard queue as quick as possible
- When error occurs, batches are returned to the queue as individual items and might be processed in a different grouping than original batch
- Occasionally, the event source mapping might receive the same item from the queue 2x, even if no function error occurred
- Lambda deletes from the queue after successfully processed
- Can configure the source queue to send items to DLQ if they can't be processed

Lambda Event Mapper Scaling

- Kinesis Data Streams & DynamoDB Streams:
    - 1 Lambda function per stream shard
    - Using parallelization, up to 10 batches processed per shard simultaneously
- SQS Standard
    - Lambda adds 60 more instances per minute to scale up
    - Up to 1000 batches of messages processed simultaneously
- SQS FIFO
    - Messages with same group ID will be processed in order
    - Lambda function scales to the number of active message groups

Lambda Event & Context Objects



- Event Object: passed to lambda function; has detail around the event itself
    - JSON document for data for the function to process
    - Contains info from invoking service

        - Lambda runtime converts the event to an object
- Context Object: metadata around lambda function
    - Methods and properties about invocation, function and runtime event
    - Passed to function by Lambda at runtime

Lambda Destinations

- Send result of async invocation to somewhere
- Async: can define destinations for successful and failed events to
    - SQS, SNS, Lambda, EventBridge bus
- Use instead of DLQ (but both can be used at the same time)
    - DLQ only allows failures to SNS / SQS, while Destinations allows success and failures to more places
- Event Source Mapping: for discarded event batches
    - SQS, SNS
        - You can send events to a DLQ directly from SQS

Lambda Permissions – IAM Roles & Resource Policies

Lambda Execution Role (IAM Role)

- Grants Lambda permissions to AWS services / resources (Lambda calls other services)
- When use event source mapping to invoke your function, Lambda uses the execution role to read event data
    - One Lambda execution role per function

Lambda Resource Based Policies

- Gives other services permission to use Lambda resources
    - Similar to S3 bucket policies
    - When S3 calls Lambda function, must use resource based policy
- IAM principal can access Lambda if:
    - IAM policy attached to principal authorizes it
    - If resource based policy authorizes (services across)

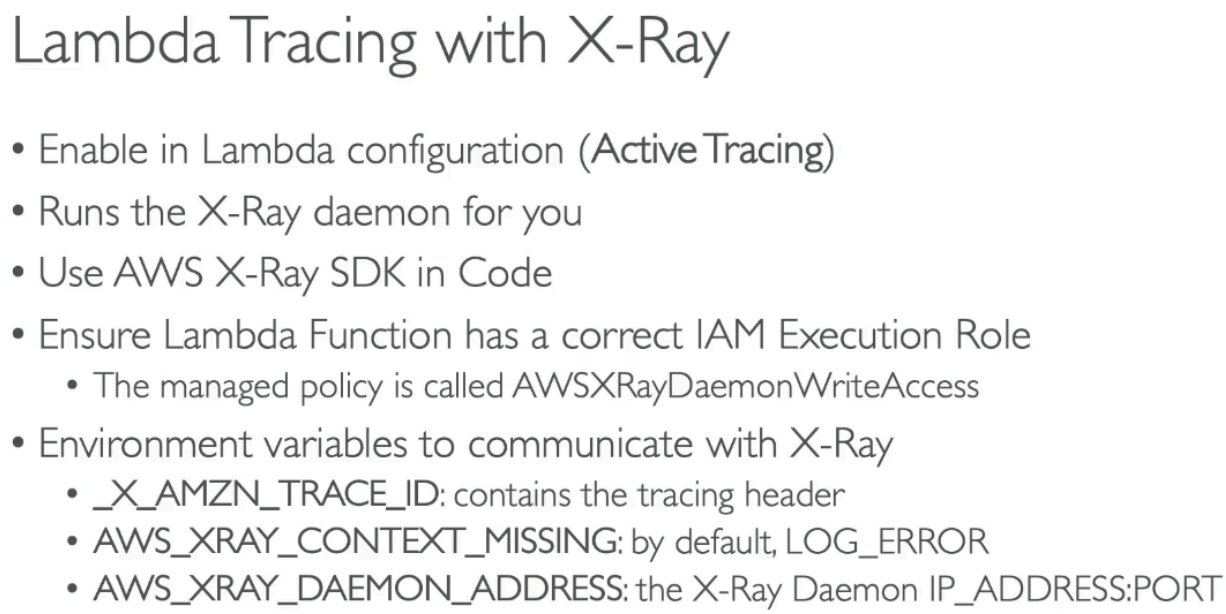
Lambda Environment Variables

- Key value pair, adjust function behavior without updating code; available to code, has default system environment variables
- Secrets encrypted via Lambda service key or KMS

Lambda Monitoring & X Ray Tracing

- CloudWatch Logs

    - Lambda execution logs stored in CloudWatch
- CloudWatch Metrics
    - Lambda metrics are found in CloudWatch metrics for invocations, duration, concurrent executions, error count…
- Enable Active Tracing for X Ray to enable X Ray daemon for you and have SDK in code
    - Needs execution role and environment variables:



Lambda @ Edge & CloudFront Functions

- 2 types CloudFront Functions & Lambda @ Edge
- Edge Functions:
    - Code to attach to CloudFront distributions; runs close to users to minimize latency
    - Serverless, deployed globally, pay for what you use
    - Customize CDN content
- Use cases:
    - Security + privacy, dynamic web app at the edge, SEO, AB testing, etc…
- CloudFront Functions:
    - Lightweight functions in JS to modify viewer request/response
        - Viewer Request: after CF receives a request from viewer
        - Viewer Response: before CF forwards response to viewer
    - High scale latency sensitive CDN customizations (millions request/second)
        - High performance, high scale
    - Native feature of CloudFront (managed within CF)
- Lambda @ Edge
    - Node or Python, scales to 1000s request / second
    - Change CF requests and responses:
        - Viewer request: after CF receives a request from a viewer

    - Origin request: before CF forwards response from origin
    - Origin response: after CF receives response from origin
    - Viewer response: before CF forwards response to viewer
- Author functions in 1 AWS region (us east 1), then CF replicates to other locations

CloudFront Functions vs Lambda @ Edge

CloudFront Functions vs. Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime Support	JavaScript	Node.js, Python
# of Requests	**Millions** of requests per second	**Thousands** of requests per second
CloudFront Triggers	- Viewer Request/Response	- Viewer Request/Response - Origin Request/Response
Max. Execution Time	< 1 ms	5 – 10 seconds
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6th price of @Edge	No free tier, charged per request & duration

- Major differences: high scale for CF functions, Lambda @ Edge triggers from both viewer and origin, max execution time lower for CF functions,

Uses Cases

CloudFront Functions vs. Lambda@Edge - Use Cases

CloudFront Functions

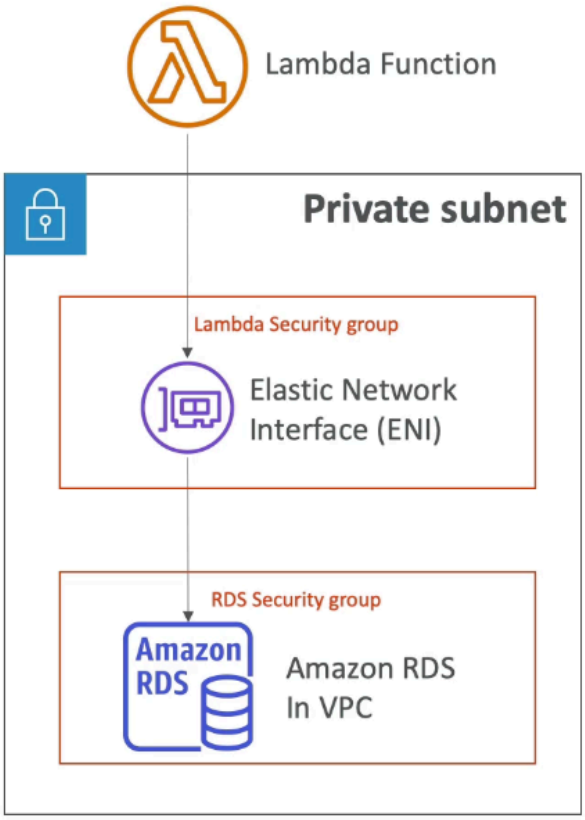
- Cache key normalization
  - Transform request attributes (headers, cookies, query strings, URL) to create an optimal Cache Key
- Header manipulation
  - Insert/modify/delete HTTP headers in the request or response
- URL rewrites or redirects
- Request authentication & authorization
  - Create and validate user-generated tokens (e.g., JWT) to allow/deny requests

Lambda@Edge

- Longer execution time (several ms)
- Adjustable CPU or memory
- Your code depends on a 3rd libraries (e.g., AWS SDK to access other AWS services)
- Network access to use external services for processing
- File system access or access to the body of HTTP requests

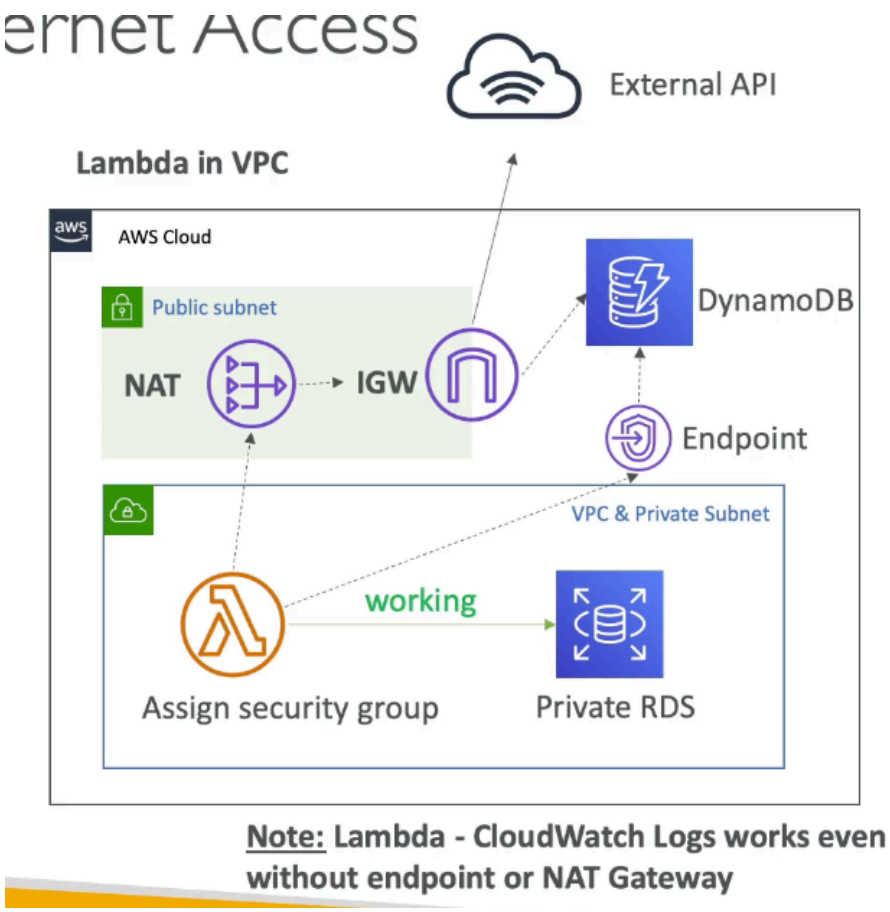
Lambda in VPC

- By default lambda functions are launched outside VPC and cannot access resources in VPC
- Must define VPC ID, subnets and SG. Lambda will create an ENI (Elastic Network interface) in subnets via AWSLambdaVPCAccessExecutionRole



Internet Access

- Lambda function in VPC has no internet access by default
  - Deploying a lambda function in a public subnet does not give internet access or public IP
- Must deploy lambda function in private subnet and give it NAT Gateway / Instance access
- VPC endpoints are used to privately access AWS services without a NAT

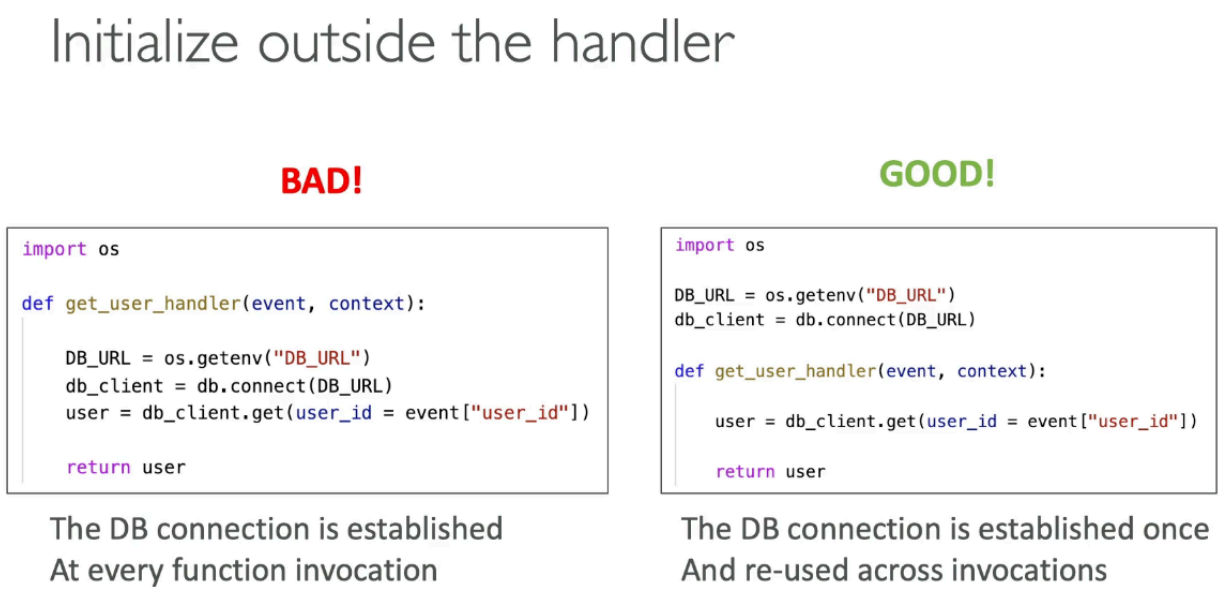


Note: Lambda - CloudWatch Logs works even without endpoint or NAT Gateway

Lambda Function Configuration and Performance

- RAM: more RAM, more vCPI credits from 128MB to 1 GB in 1 MB increments
    - 1792 MB = 1 full vCPU, past that you need multithreading for benefits
    - If CPU bound, increase RAM
- Timeout: default 3 seconds, max 15 minutes

Lambda Execution Context



- Temporary runtime env that initializes any external dependencies of lambda code
- Great for DB connections, HTTP clients, SDK clients…
- Execution context is maintained for some time in anticipation of another Lambda function invocation
    - Next invocation can reuse context to execution time and save time in initializing connection objects
- Includes /tmp directory available across executions

/tmp space

- Temp disk space for downloads, perform operations… with max size of 10 GB
- Content remains when execution context is frozen, but for permanent persistence use S3
- Encrypt content you must generate KMS data keys

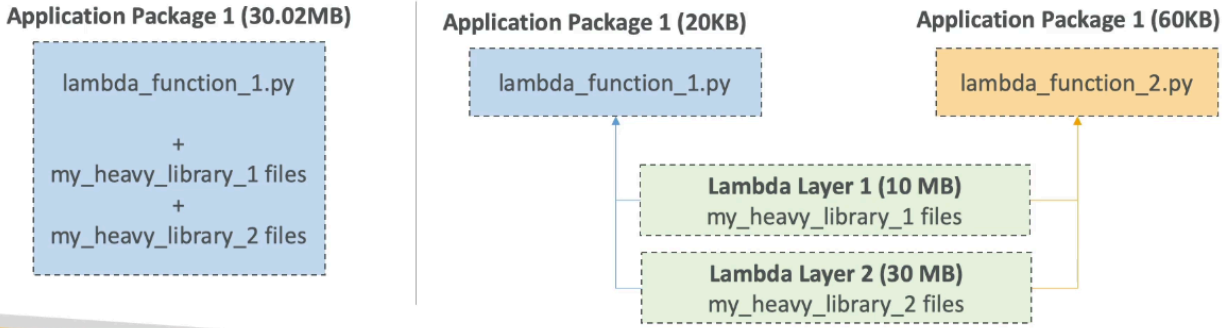
Lambda Layers

- Custom runtimes, externalize dependencies to reuse them

- Packages dependencies to reuse across lambda functions, separate application package (app code) and dependencies

Lambda Layers

- Custom Runtimes
  - Ex: C++ https://github.com/awslabs/aws-lambda-cpp
  - Ex: Rust https://github.com/awslabs/aws-lambda-rust-runtime
- Externalize Dependencies to re-use them:



Lambda File System Mounting

Lambda – Storage Options

	Ephemeral Storage /tmp	Lambda Layers	Amazon S3	Amazon EFS
Max. Size	10,240 MB	5 layers per function up to 250MB total	Elastic	Elastic
Persistence	Ephemeral	Durable	Durable	Durable
Content	Dynamic	Static	Dynamic	Dynamic
Storage Type	File System	Archive	Object	File System
Operations supported	any File System operation	Immutable	Atomic with Versioning	any File System operation
Pricing	Included in Lambda	Included in Lambda	Storage + Requests + Data Transfer	Storage + Data Transfer + Throughput
Sharing/Permissions	Function Only	IAM	IAM	IAM + NFS
Relative Data Access Speed from Lambda	Fastest	Fastest	Fast	Very Fast
Shared Across All Invocations	No	Yes	Yes	Yes

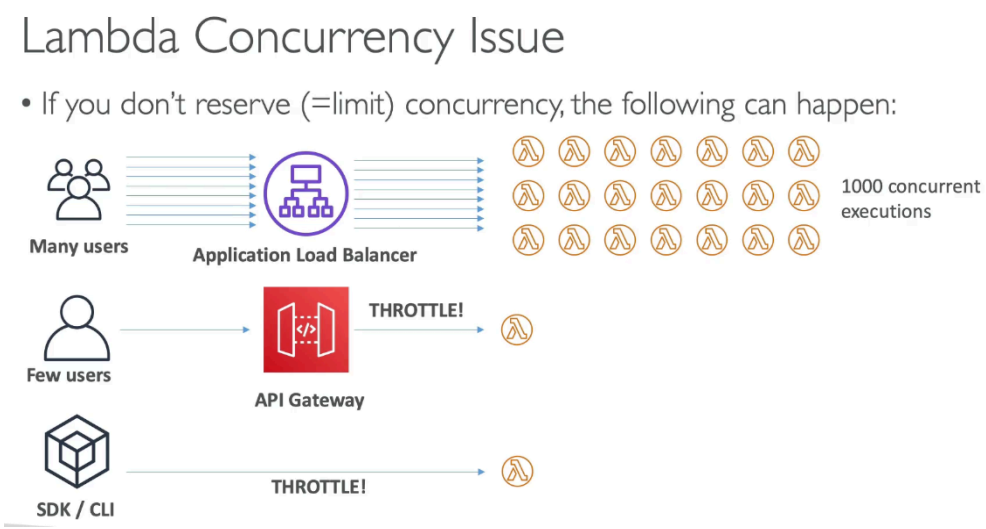
- Lambda functions can access EFS if running on VPC, just configure lambda to mount EFS file systems to local directory during initialization

- Uses EFS Access Points
- Limitations:
    - Each lambda connected = 1 connection, might reach connection limits or connection burst limit where too many lambda's accessing at the same time

Lambda Concurrency

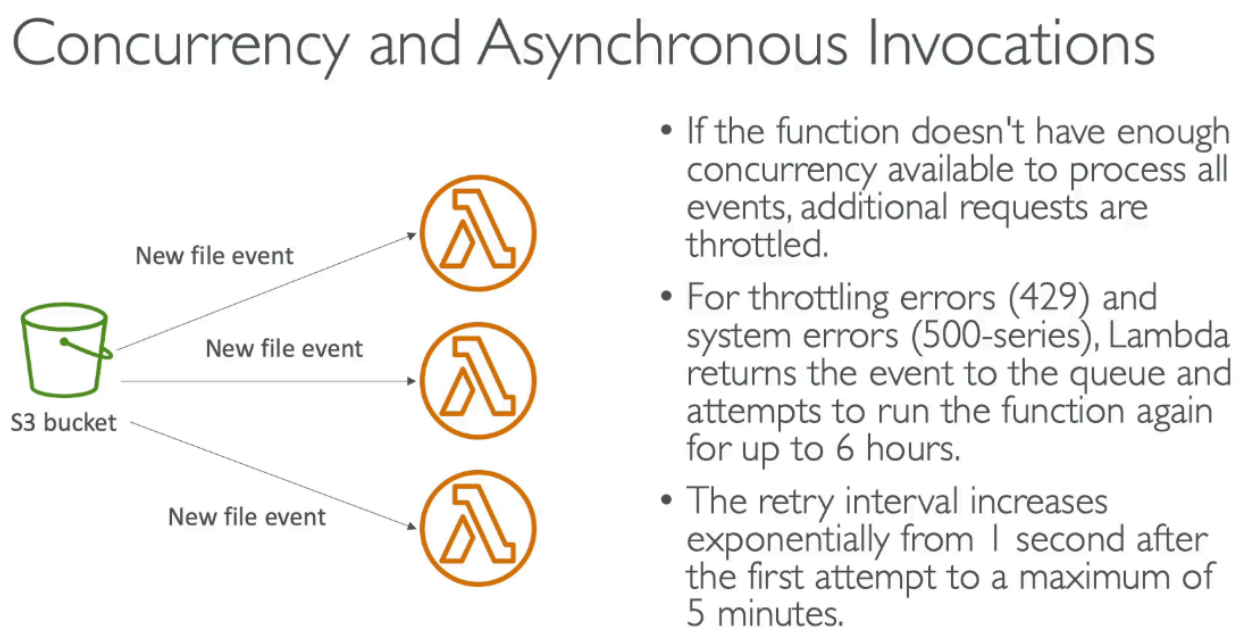
- Concurrency limit: 1000 concurrent Lambda function executions, support ticket for more
- Reserved concurrency at function level = limit where each invocation over the limit triggers a throttle
- Throttle behavior:
    - Synchronous: return ThrottleError 429
    - Async: retry, then DLQ

Issues



- No limit set means concurrency limit applies to all lambda functions in the account, thus if one lambda function takes up all the limit, others get throttled

Concurrency and Async Invocations

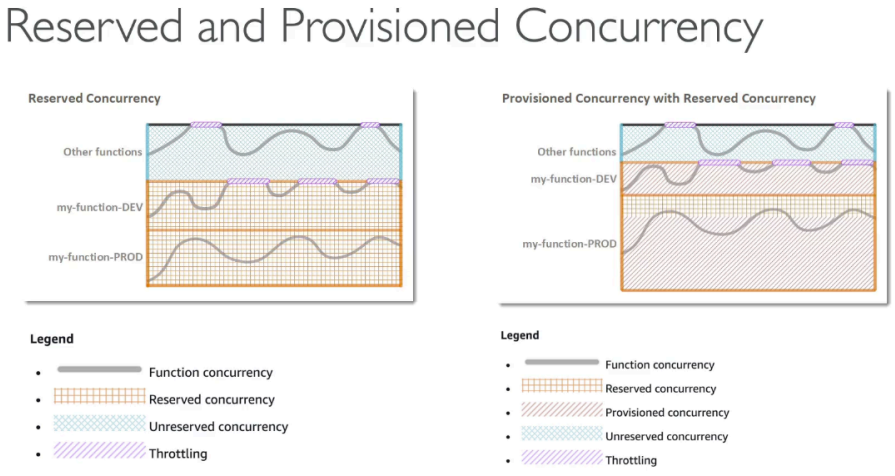


- Past the limit, requests are throttled. Lambda creates a queue and runs again for up to 6 hours before exponentially increasing retry from 1 second after first attempt to 5 min

Cold Starts & Provisioned Concurrency

- Cold start: new instance where code must be initialized before running
    - First request served by new instances has higher latency than rest
- Provisioned Concurrency: allocate concurrency before function is invoked
    - No cold start, using Application auto scaling to manage concurrency

Reserved and Provisioned Concurrency

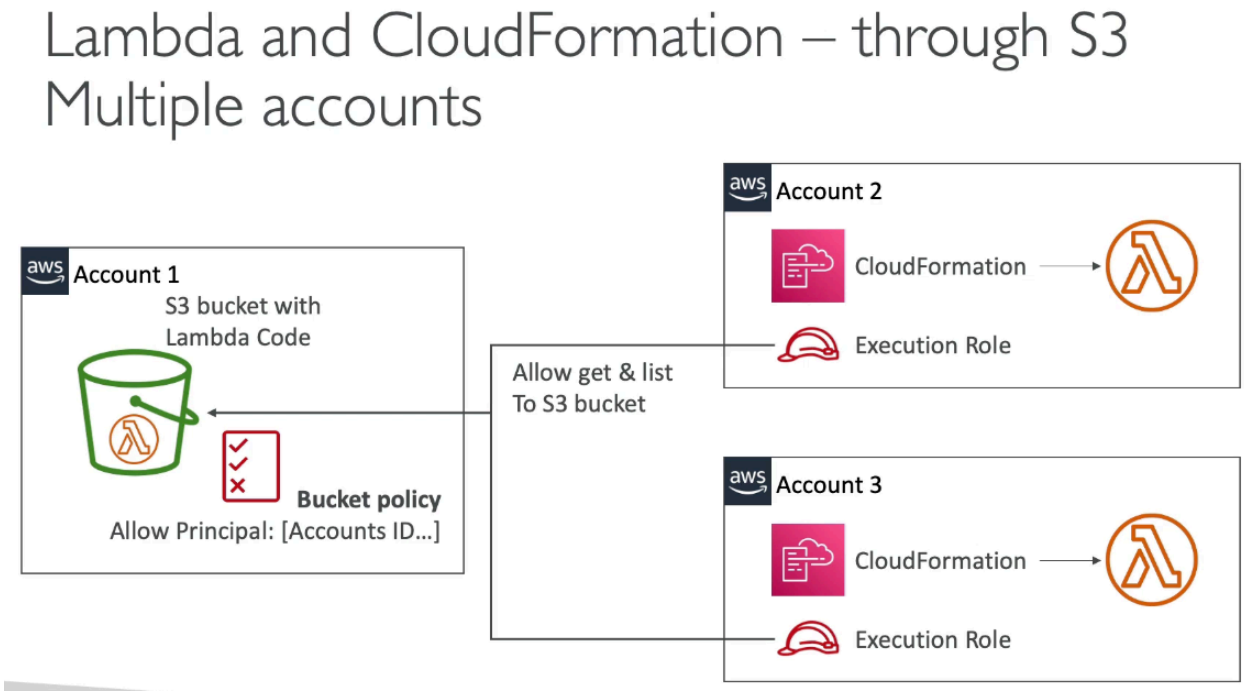


Lambda External Dependencies

- Using packages means you must zip code + dependencies to upload straight to Lambda if < 50 MB, else to S3 first

Lambda & CloudFormation

- Inline: define lambda function code inline for very simple functions
    - Cannot include function dependencies
- Via S3: store function zip in S3 and refer to S3 zip location
    - If you update code in S3 but don't update bucket, version, etc… CF won't update function
- Through S3 Multiple Accounts



Lambda Container Images

- Deploy lambda function as container images up to 10 GB from ECR → pack complex or large dependencies in a container
    - Base image must implement Lambda Runtime API, allowing test containers locally via Lambda Runtime Interface Emulator for unified workflow

Lambda Container Images

- Example: build from the base images provided by AWS

```
# Use an image that implements the Lambda Runtime API
FROM amazon/aws-lambda-nodejs:12

# Copy your applciation code and files
COPY app.js package*.json ./

# Install the dependencies in the container
RUN npm install

# Function to run when the Lambda function is invoked
CMD [ "app.lambdaHandler" ]
```

Lambda Container Images – Best Practices

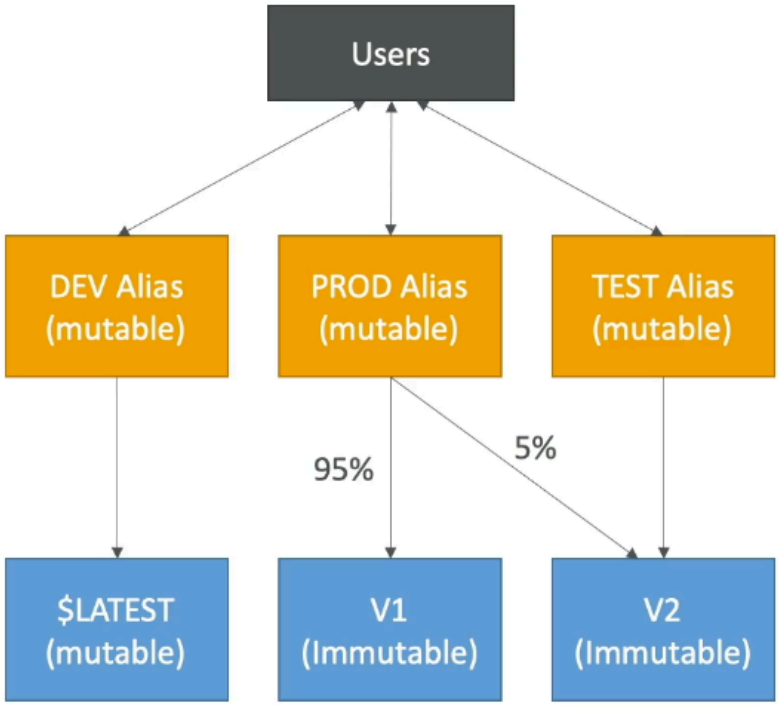
- Strategies for optimizing container images:
  - Use AWS-provided Base Images
    - Stable, Built on Amazon Linux 2, cached by Lambda service
  - Use Multi-Stage Builds
    - Build your code in larger preliminary images, copy only the artifacts you need in your final container image, discard the preliminary steps
  - Build from Stable to Frequently Changing
    - Make your most frequently occurring changes as late in your *Dockerfile* as possible
  - Use a Single Repository for Functions with Large Layers
    - ECR compares each layer of a container image when it is pushed to avoid uploading and storing duplicates
- Use them to upload large Lambda Functions (up to 10 GB)

Lambda Versions & Aliases

- $LATEST is the latest (mutable), but publishing a lambda version is immutable and has version numbers + ARN → version = code + configuration

- Aliases are “pointers” to Lambda functions to point to different versions
  - Mutable, have own ARNs, cannot reference other aliases, only versions

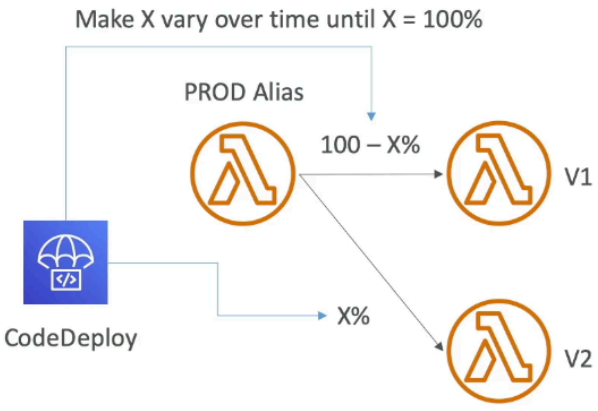
- Aliases enable canary deployment by assigning weights to lambda to enable stable configuration of event triggers or destinations



Lambda & CodeDeploy

Lambda & CodeDeploy

- CodeDeploy can help you automate traffic shift for Lambda aliases
- Feature is integrated within the SAM framework
- Linear: grow traffic every N minutes until 100%
  - Linear10PercentEvery3Minutes
  - Linear10PercentEvery10Minutes
- Canary: try X percent then 100%
  - Canary10Percent5Minutes
  - Canary10Percent30Minutes
- AllAtOnce: immediate
- Can create Pre & Post Traffic hooks to check the health of the Lambda function



Lambda & CodeDeploy – AppSpec.yml

```
version: 0.0

Resources:
  - myLambdaFunction:
      Type: AWS::Lambda::Function
      Properties:
        Name: myLambdaFunction
        Alias: myLambdaFunctionAlias
        CurrentVersion: 1
        TargetVersion: 2
```

- Name (required) – the name of the Lambda function to deploy
- Alias (required) – the name of the alias to the Lambda function
- CurrentVersion (required) – the version of the Lambda function traffic currently points to
- TargetVersion (required) – the version of the Lambda function traffic is shifted to

- CodeDeploy helps automate traffic shift for Lambda aliases
- Linear: grow traffic every N numbers until 100%
- Canary: try X% then 100%
- All at once: immediate
- Pre & Post traffic hooks to check the health of the lambda functions

Lambda Function URL

- Dedicated HTTP(s) endpoint, unique URL generated (never changes)
    - Accessed via public internet only via web, postman, etc… → no support to Private Link because it's public
    - Supports resource based policies, CORS
        - Resource based policies: authorizes accounts, CIDR, IAM principals access
        - CORS: if calling lambda function from different domain
    - Can be applied to any alias or $LATEST, but cannot be applied to other function versions
- Throttle via reserved concurrency

- https://<url-id>.lambda-url.<region>.on.aws (dual-stack IPv4 & IPv6)

Function URL Security

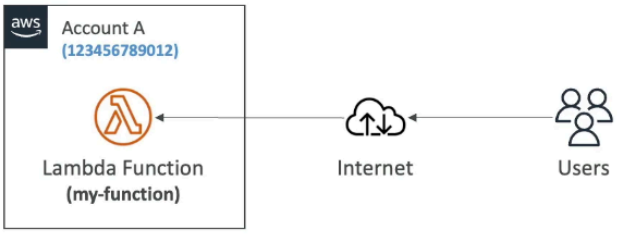
- AuthType NONE: allow public and unauthenticated access
    - Resource based policy always in effect (must grant public access)

Lambda – Function URL Security

- AuthType NONE – allow public and unauthenticated access
  - Resource-based Policy is always in effect (must grant public access)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "lambda:InvokeFunctionUrl",
            "Resource": "arn:aws:lambda:us-east-1:123456789012:
function:my-function",
            "Condition": {
                "StringEquals": {
                    "lambda:FunctionUrlAuthType": "NONE"
                }
            }
        }
    ]
}
```

Resource-based Policy

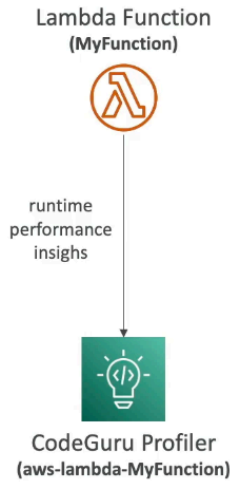


- AuthType AWS_IAM: IAM used to authenticate and authorize requests
  - Both principal's identity based policy & resource based policy are evaluated
  - Principal must have InvokeFunctionUrl permissions
  - Same account: identity based policy OR resource based policy as ALLOW
  - Cross account: identity based policy AND resource based policy as ALLOW

Lambda + CodeGuru

Lambda and CodeGuru Profiling

- Gain insights into runtime performance of your Lambda functions using CodeGuru Profiler
- CodeGuru creates a Profiler Group for your Lambda function
- Supported for Java and Python runtimes
- Activate from AWS Lambda Console
- When activated, Lambda adds:
  - CodeGuru Profiler layer to your function
  - Environment variables to your function
  - AmazonCodeGuruProfilerAgentAccess policy to your function



- Insights for runtime performance of lambda function via CodeGuru Profiler
- CodeGuru creates a Profiler Group for Lambda function
- Lambda adds:
    - CodeGuru Profiler layer to function
    - Env variables
    - IAM policy to function

Lambda Limits

- Execution
    - Memory allocation: 128 MB to 10 GB
    - More RAM, more vCPU
    - Max execution time: 15 min, 900 seconds
    - 4KB for env vars
    - /temp space for 512 MB to 10 GB
    - 1000 concurrency executions
- Deployments
    - Zip size: 50 MB
    - Uncompressed deployment (code + dependencies): 250 MB
    - /temp to load files at startup
    - 4 KB for env vars

Lambda Best Practices

- Perform heavy duty work outside function handler
- Use env vars for:
    - Secrets, passwords…
- Minimize deployment package size to its runtime necessities
- Avoid recursive code

Section 22: AWS Serverless: DynamoDB

DynamoDB Overview

- Traditional DB use RDBMS DB with SQL, but have strong requirements on how data is structured and vertical/horizontal scaling methods
- NoSQL are non-relational and distributed with no query joints, meaning all data needs to be in 1 row
    - Scales horizontally

- DynamoDB fully managed, NoSQL (not like RDS), highly available multi AZ, fast performance, integration with IAM, auto scaling, Access classes
- Made of tables with each table having a primary key (must be decided at creation time)

- Infinite number of items or rows with attributes (can be added over time or null) with item size up to 400 KB
    - Supported types: Scalar, document, set types

Primary Key

1. Partition Key (hash)
    a. Must be unique for each item and “diverse” so data is distributed
2. Partition + Sort Key (hash + range)
    a. Must be unique, grouped by partition key
        i. Can have the same partition key, but need sort key to be different
    b. Both are the “primary key”

Dynamo WCU & RCU – Throughput

- Control table’s capacity (read / write throughput)
- Switch between modes every 24 hours

1. Provisioned Mode (default)
    a. Specify number of read/writes per second, planning capacity beforehand
    b. Pay for provisioned read & write capacity units
2. On Demand Mode
    a. Read/writes auto scaled without capacity planning, pay for what you use but more expensive

Provisioned

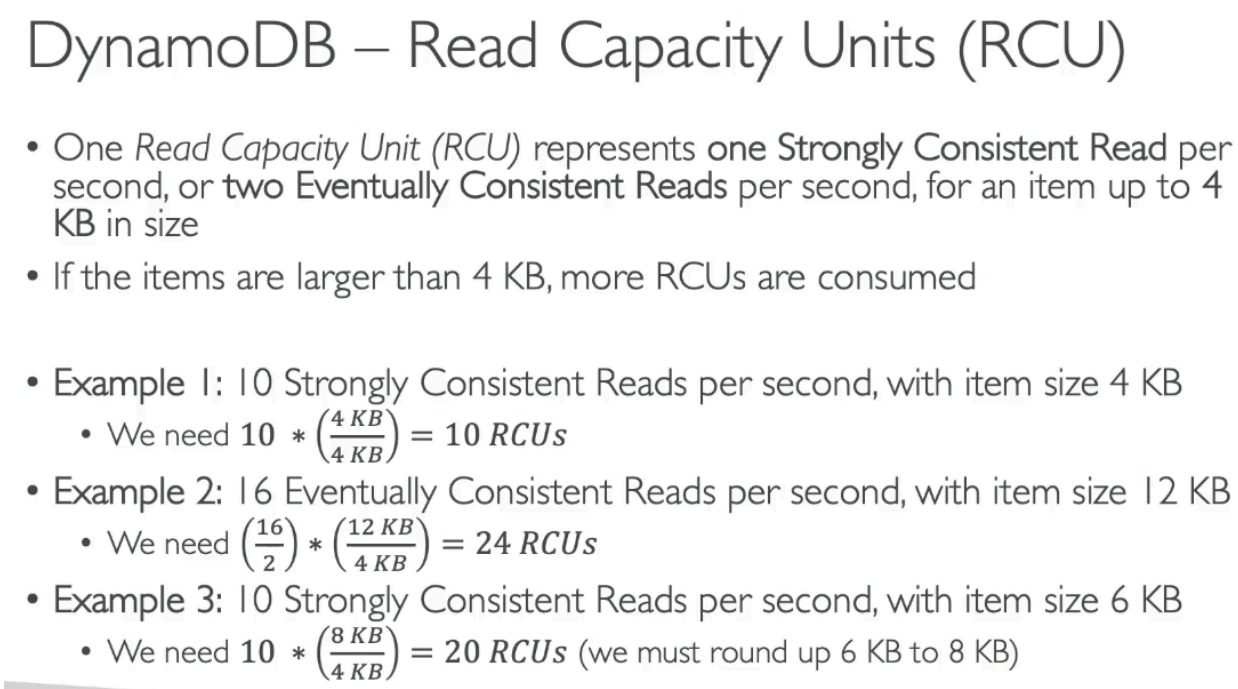
DynamoDB – Write Capacity Units (WCU)

- One *Write Capacity Unit (WCU)* represents one write per second for an item up to 1 KB in size
- If the items are larger than 1 KB, more WCUs are consumed

- Example 1: we write 10 items per second, with item size 2 KB
    - We need $10 * (\frac{2\ KB}{1\ KB}) = 20\ WCUs$
- Example 2: we write 6 items per second, with item size 4.5 KB
    - We need $6 * (\frac{5\ KB}{1\ KB}) = 30\ WCUs$ (4.5 gets rounded to the upper KB)
- Example 3: we write 120 items per minute, with item size 2 KB
    - We need $\left(\frac{120}{60}\right) * (\frac{2\ KB}{1\ KB}) = 4\ WCUs$

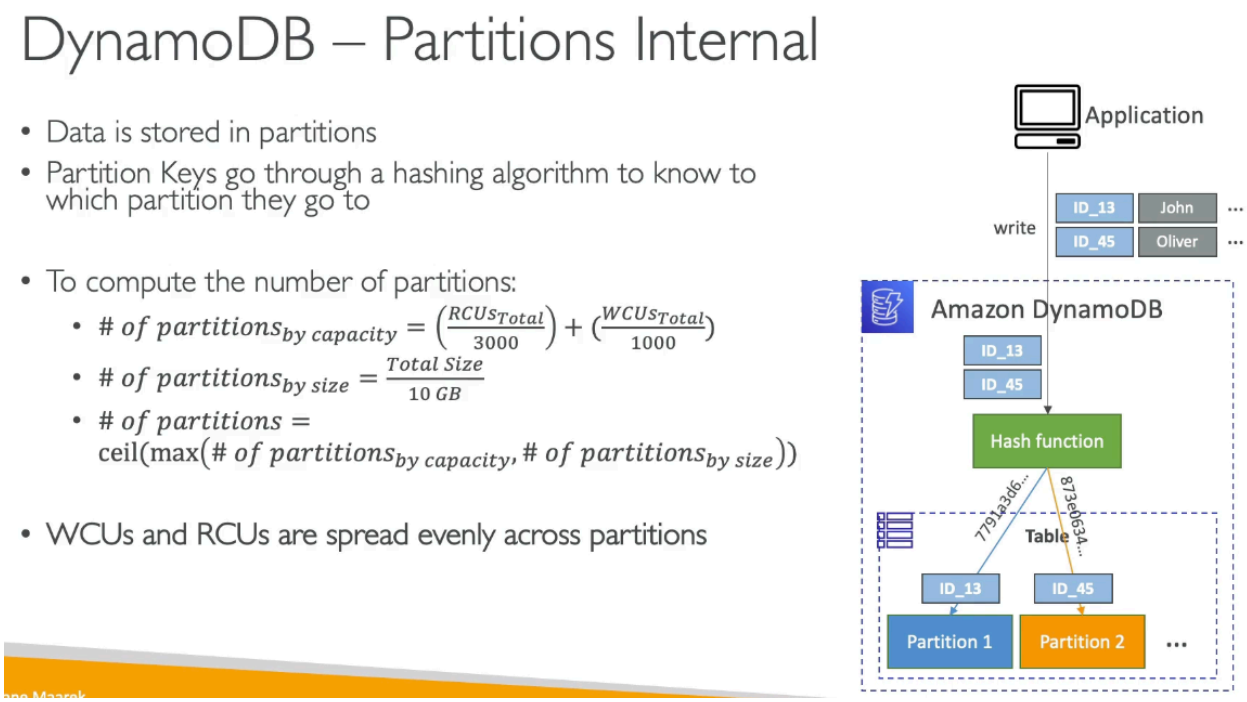
- Tables must have provisioned read/write capacity units
    - Can setup auto scaling to meet demand and throughput can be exceeded temporarily using “burst capacity”
        - ProvisionedThroughputExceededException if burst capacity reached, then starting exponential backoff retry
- Read Capacity Units (RCU): read throughput
    - 1 Strongly consistent read / second or 2 eventually consistent reads / second for up to 4 KB
    - Larger than 4 KB, more RCU consumed
- Write Capacity Units (WCU): write throughput
    - 1 write / second for item up to 1 KB; larger than 1 KB consumes more WCUs by rounding up

Strongly Consistent Read vs Eventually Consistent Read



- Eventually Consistent Read (default)
    - If we read after a write, it’s possible to get stale data because of replication time
- Strongly Consistent Read
    - Reading after a write we get correct data
    - Set “consistent read” parameter to true in API calls
    - Consumes 2x RCU

Partitions Internal



- Data is stored in partitions
- Partition keys go through hashing algorithm to know which partition to go to
- WCUs and RCUs are spread evenly across partitions

Throttling

- If exceeded provisioned RCU or WCU get ProvisionedThroughputExceededException
- Reasons:
    - Hot keys: one partition key getting read too many times
    - Hot partitions
    - Very large items (RCU and WCU depends on item size)
- Solutions:
    - Exponential backoff, distribute partition keys, if RCU use DynamoDB Accelerator

R/W Capacity Mode – On Demand

- R/W auto scaled with no capacity needed, thus unlimited WCU/RCU and no throttle, but more expensive (2.5x more than provisioned)
    - Charged for read/writes used in terms of RRU and WRU
- Read Request Units (RRU): throughput for reads (same as RCU)
- Write Request Units (WRU): throughput for writes (same as WCU)
- Use cases: unknown workloads, unpredictable application traffic

DynamoDB Basic Operations

Writing Data

- PutItem
    - Creates new item or fully replaces old item (same primary key), consuming WCU
- UpdateItem
    - Edits existing item's attributes or adds a new item if it doesn't exist
    - Can be used to implement atomic counters – a numeric attribute that's unconditionally incremented
- Conditional Writes
    - Accept a write/update/delete only if conditions are met, otherwise error
    - Helps with concurrent access to items

Read Data

- GetItem
    - Read based on primary key (hash or hash + range)
    - Eventually consistent read (default) or strongly consistent reads (more RCU, takes longer)
    - ProjectionExpression can be specified to retrieve only certain attributes
- Query
    - Returning items based on:
        - KeyConditionExpression
            - Partition key value (must be = operator) - required
            - Sort key value – optional
        - FilterExpression
            - Additional filtering after query operation (before data returned)
            - Use only with non-key attributes (does not allow hash or range attributes)
    - Returns:
        - Number of items specified in limit or up to 1 MB data
    - Can query tables, local secondary indexes, etc…
- Scan
    - Scan entire table then filter (inefficient)
    - Returns up to 1 MB data, consuming lots of RCU so use limit statement to reduce size of result
    - Parallel Scan for faster scan – multiple workers to scan data segments, but increases throughput and RCU
    - Can use ProjectionExpression and FilterExpression (no changes to RCU)

Deleting Data

- DeleteItem

    - Delete individual item (can be conditional)
- DeleteTable
    - Delete whole table and all items, quicker than calling DeleteItem on all items

Batch Operations

- Save latency by reducing API calls, where operations done in parallel for better efficiency
- Part of batch can fail for a retry of only failed items
- BatchWriteItem
    - Up to 25 PutItem and/or DeleteItem in one call
    - Up to 16 MB data written 400 KB data per item
    - Can't update items (use UpdateItem)
    - UnprocessedItems for failed write operations (exponential backoff or add WCU)
- BatchGetItem
    - Return items from 1+ tables, up to 100 items, 16 MB of data
    - Items are retrieved in parallel to minimize latency
    - UnprocessedKeys for failed read operations (exponential backoff or add RCU)

PartiQL

- SQL for DynamoDB to access data using SQL
- Run queries but cannot do JOIN, only SELECT, INSERT, UPDATE, DELETE

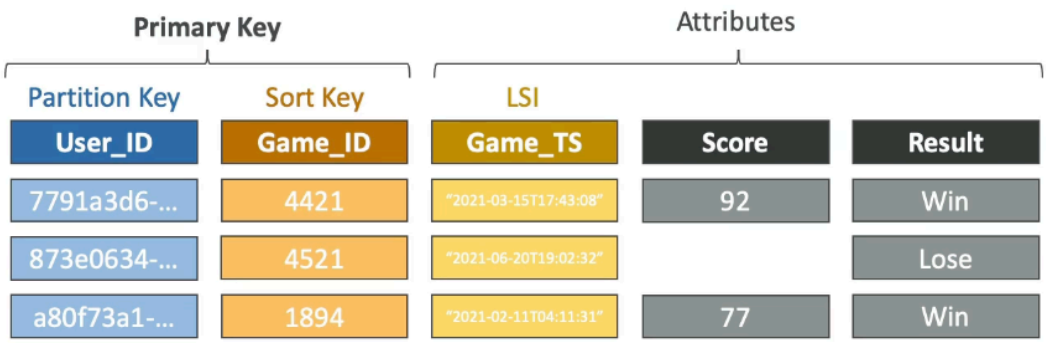
Conditional Writes

- Specified expressions for what items should be modified; filtered expressions only filter read queries while conditional expressions are for write operations
- Used to not overwrite elements
    - Attribute_not_exists(partition_key) – make sure item isn't overwritten
    - Attribute_not_exists(partition_key) and Attribute_not_exists(sort_key) – make sure partition / sort key combo not overwritten

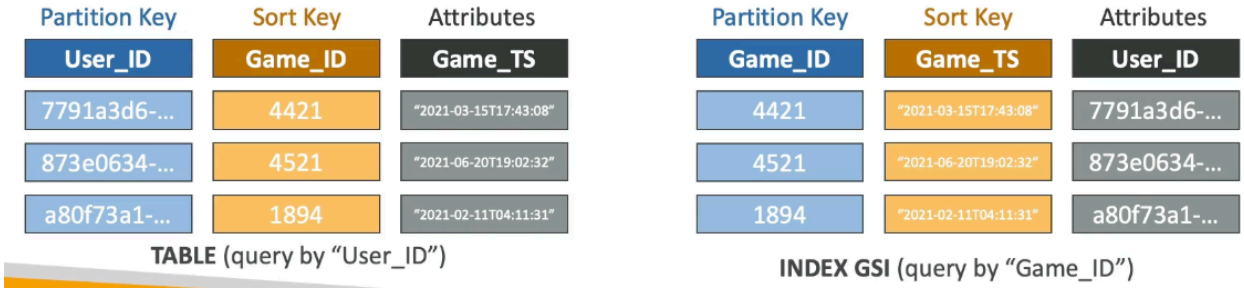
Indexes (GSI + LSI)

Local Secondary Index (LSI)

- Alternative sort key for table (same partition key as base table), up to 5 local secondary indexes per table but must be defined at table creation time
- Sort key consists of 1 scalar attribute
    - Attribute projections: can contain some or all attributes of the base table



Global Secondary Index (GSI)



- Alternative primary key (hash or hash + range) from base table, helpful to speed up queries on non-key attributes
- Index key consists of scalar attributes
    - Attribute projections: can contain some or all attributes of the base table
- Must provision RCUs & WCUs for the index
- Can be added/modified after table creation

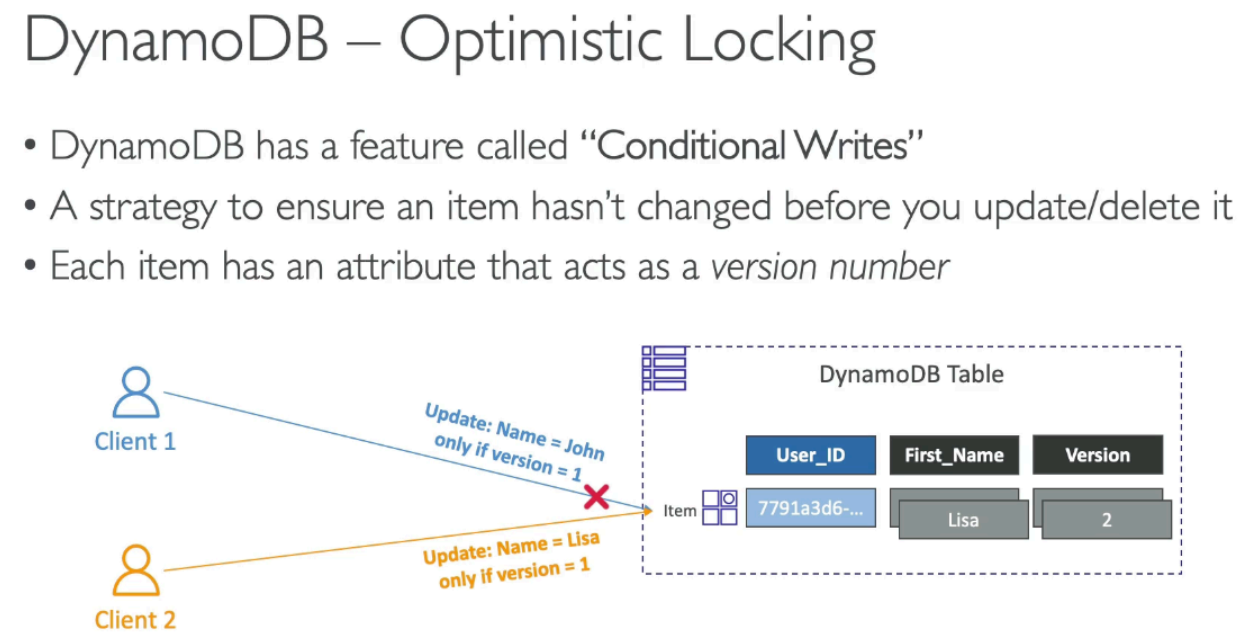
Indexes and Throttling

- GSI:
    - If writes are throttled on GSI, main table will be throttled, even if WCU on main tables are fine; must choose GSI partition key carefully and assign WSU capacity carefully
- LSI
    - Uses WCU and RCU of main table, no special throttling considerations

PartiQL

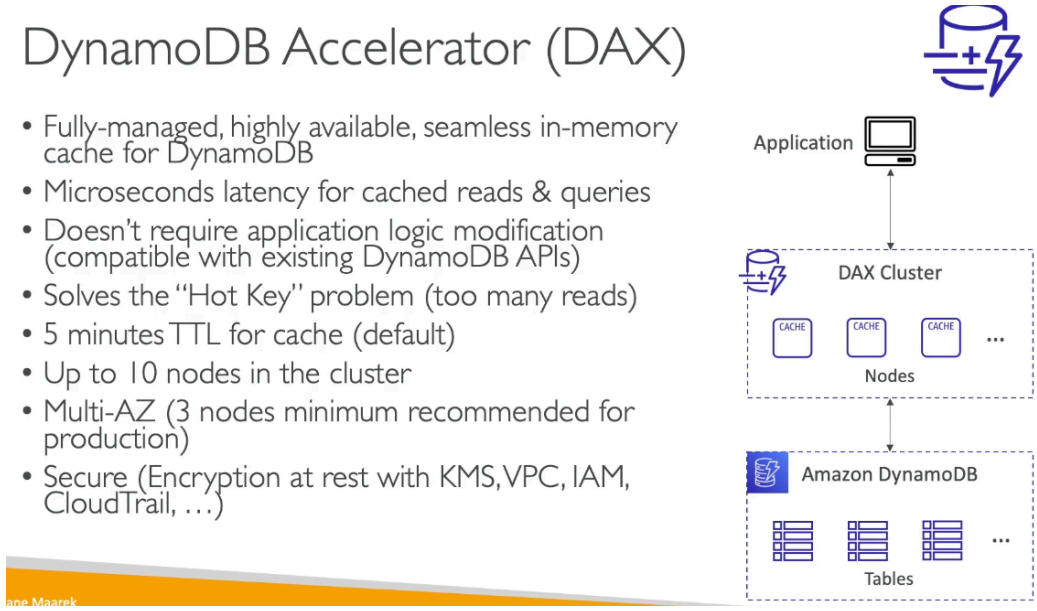
- SQL like syntax to manipulate DynamoDB tables
- Supports batch operations

Optimistic Locking



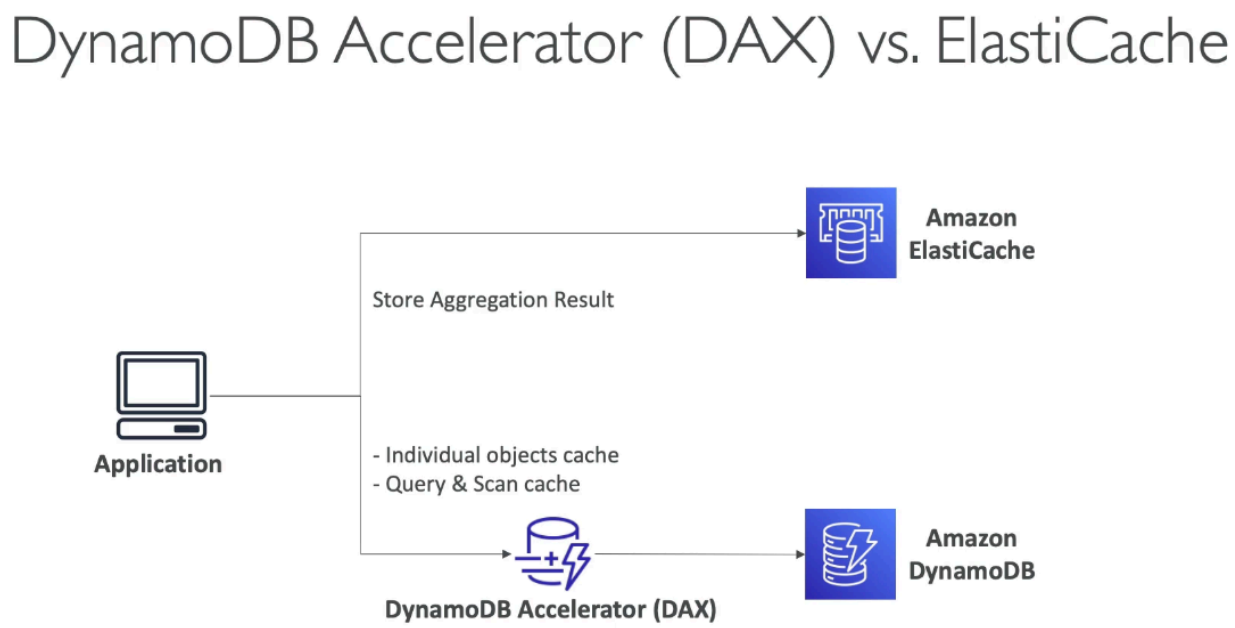
- Conditional writes, which is a strategy to ensure an item hasn't changed before you update/delete it
    - Each item has attribute that acts as version number

DynamoDB Accelerator (DAX)

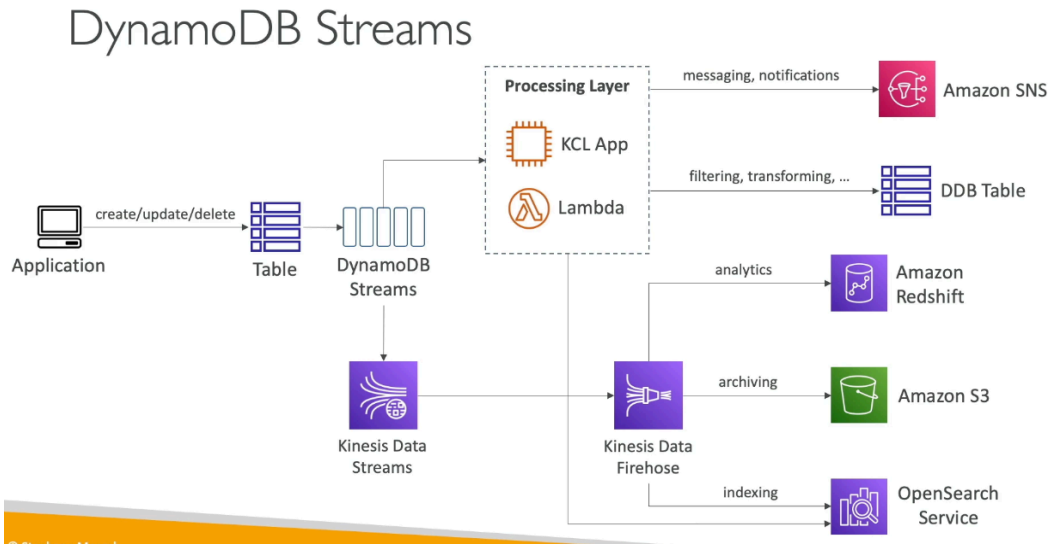


- Fully managed, highly available, low latency, in memory cache, no need to update application logic with multi AZ support and encryption
- Solves “hotkey” problem (too many reads)
- 5 min TTL (default), up to 10 nodes in cluster
    - T type: baseline + bursting for low throughput
    - R type: always ready

DAX vs ElastiCache



DB Streams



- Ordered stream of item level modifications in a table
    - Records are not retroactively populated in a stream after enabling
- Made of shards, similar to Kinesis, but no shard provisioned, done by AWS
- Stream records can be sent to: Kinesis Data Streams, Lambda and have retention of 24 hours
- Ability to choose info sent to stream:

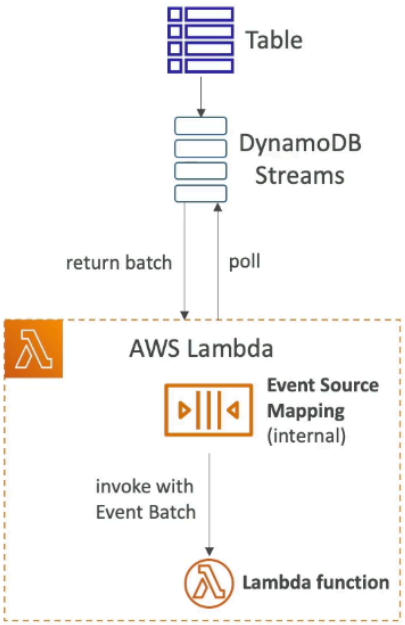
KEYS_ONLY – only the key attributes of the modified item
NEW_IMAGE – the entire item, as it appears after it was modified
OLD_IMAGE – the entire item, as it appeared before it was modified
NEW_AND_OLD_IMAGES – both the new and the old images of the item

- Uses cases: react to changes in real time, analytics, send data to other services, etc…

Streams + Lambda

DynamoDB Streams & AWS Lambda

- You need to define an Event Source Mapping to read from a DynamoDB Streams
- You need to ensure the Lambda function has the appropriate permissions
- Your Lambda function is invoked synchronously

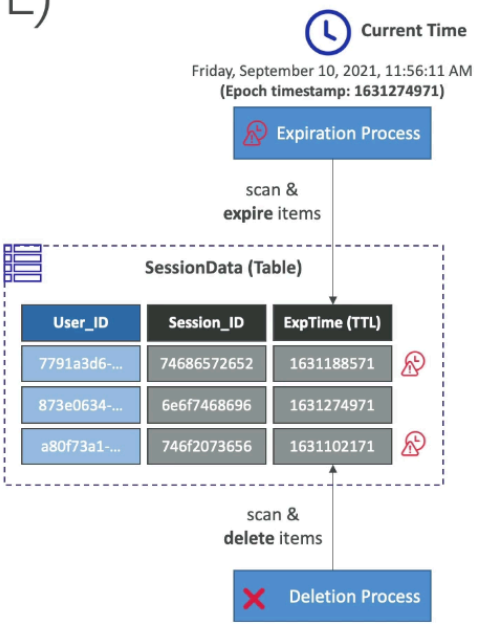


- Need Event Source Mapping to read from Streams, where Lambda is invoked synchronously

TTL

DynamoDB – Time To Live (TTL)

- Automatically delete items after an expiry timestamp
- Doesn't consume any WCUs (i.e., no extra cost)
- The TTL attribute must be a "Number" data type with "Unix Epoch timestamp" value
- Expired items deleted within 48 hours of expiration
- Expired items, that haven't been deleted, appears in reads/queries/scans (if you don't want them, 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, …



- Automatically delete items within 48 hours after TTL expires, doesn't consume WCUs
- Expired items that haven't been deleted appears in reads/queries/scans, must filter to not see them
- Expired items are deleted from both LSIs and GSIs
- Delete operation for each expired item enters Streams

DynamoDB CLI

DynamoDB CLI – Good to Know

- --projection-expression: one or more attributes to retrieve
- --filter-expression: filter items before returned to you

- General AWS CLI Pagination options (e.g., DynamoDB, S3, …)
  - --page-size: specify that AWS CLI retrieves the full list of items but with a larger number of API calls instead of one API call (default: 1000 items)
  - --max-items: max. number of items to show in the CLI (returns NextToken)
  - --starting-token: specify the last NextToken to retrieve the next set of items

- Note:
    - Page size API call: if you want 10000 items, it might timeout. Instead specify page size of 100, which behind the scenes makes 100 calls of page size 100 to make sure the complete call succeeds
    - Starting token: when you fetch individual items, it returns a next token parameter that is a pointer to where the next query is. Starting token = next token value

DynamoDB Transactions

- Coordinated all or nothing operations to multiple items across 1+ tables
- Provides atomicity, consistency, isolation, and durability (ACID)
- Read modes: eventual consistency, strong consistency, transactional
- Write mode: standard, transactional
- Consumes 2x WCU & RCUs
    - Performs 2 operations for every item (prepare & commit)
        - TransactGetItems: 1+ GetItem operations
        - TransactWriteItems: 1+ PutItem, UpdateItem, DeleteItem operations
- Uses cases: financial transactions, managing orders, multiplayer games… for consistency purposes

Transaction Capacity Computations

DynamoDB Transactions – Capacity Computations

- Important for the exam!

- Example 1: 3 Transactional writes per second, with item size 5 KB
    - We need $3 * \left(\frac{5\ KB}{1\ KB}\right) * 2\ (transactional\ cost) = 30\ WCUs$

- Example 2: 5 Transaction reads per second , with item size 5 KB
    - We need $5 * \left(\frac{8\ KB}{4\ KB}\right) * 2\ (transactional\ cost) = 20\ RCUs$
    - (5 gets rounded to the upper 4 KB)

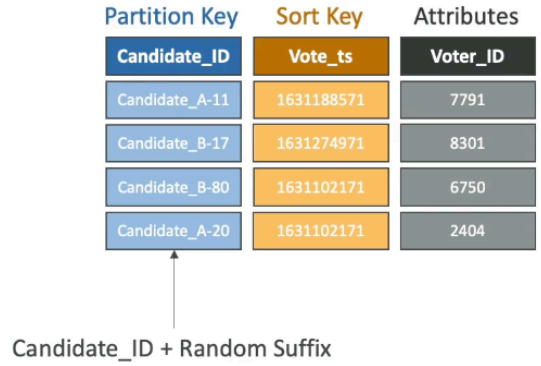
DynamoDB as Session State Cache

- vs ElastiCache:
  - ElastiCache is in memory, Dynamo is serverless
  - Both are key/value stores
- vs EFS:
  - EFS must be attached to EC2 instances as network drive
  - EFS is a file system, DynamoDB is a DB
- vs EBS & Instance Store:
  - Can only be used for local caching, not shared caching
- vs S3
  - Higher latency, not meant for small objects

Partitioning Strategies

DynamoDB Write Sharding

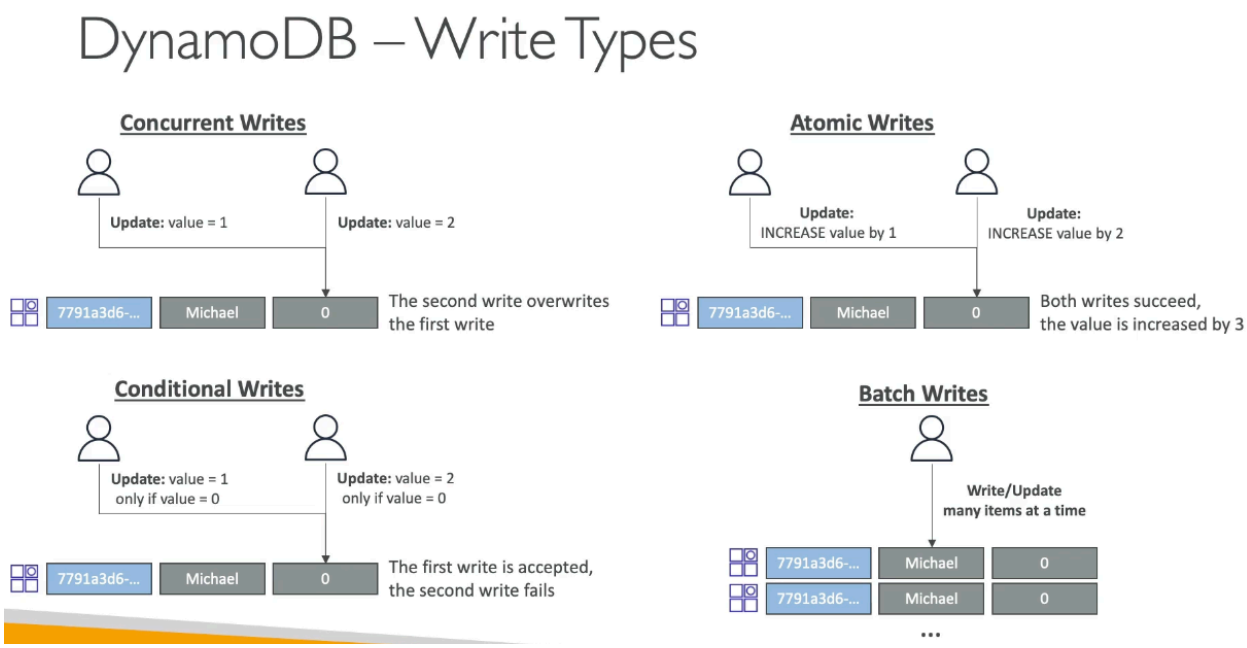
- Imagine we have a voting application with two candidates, candidate A and candidate B
- If Partition Key is "Candidate_ID", this results into two partitions, which will generate issues (e.g., Hot Partition)



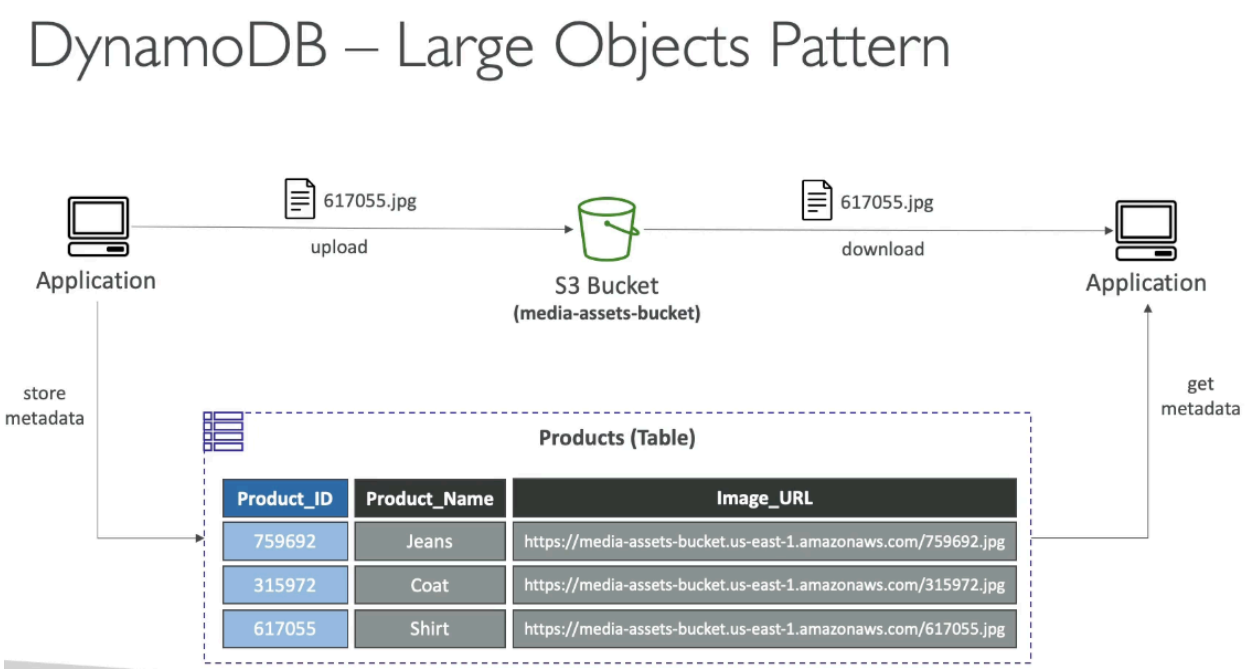
- A strategy that allows better distribution of items evenly across partitions
- Add a suffix to Partition Key value
- Two methods:
  - Sharding Using Random Suffix
  - Sharding Using Calculated Suffix

- Write sharding
  - If you don't have a distributed partition key, add a suffix to partition key to avoid hot partition

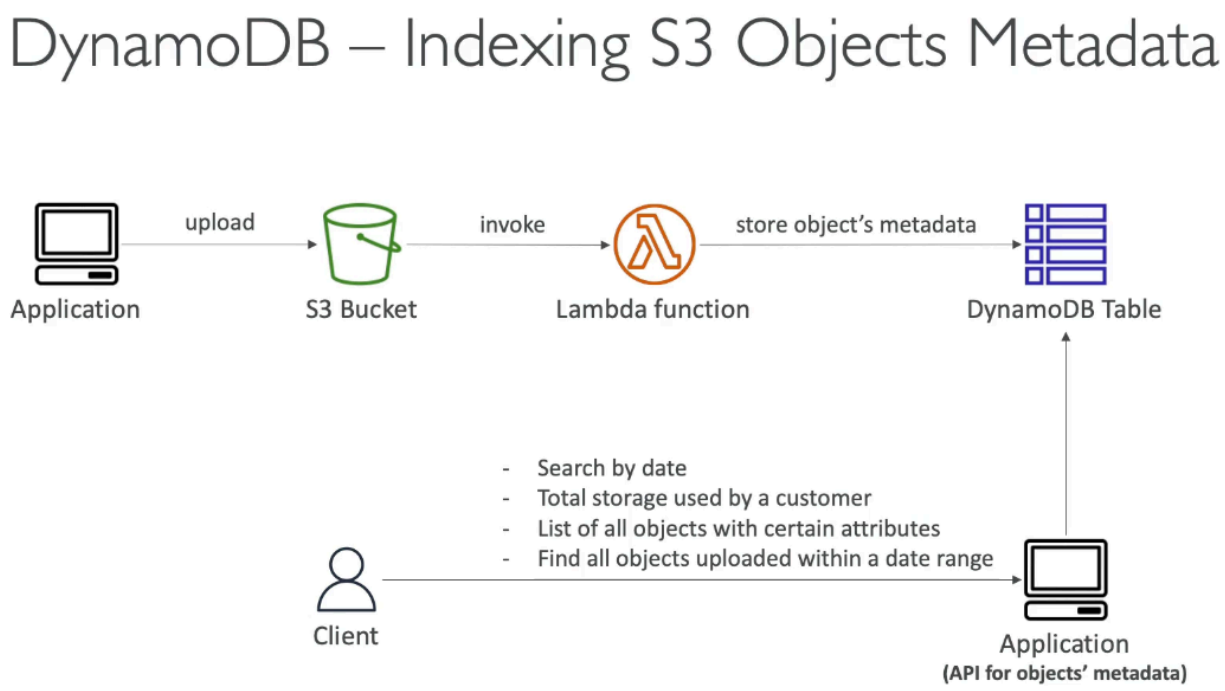
Conditional Writes, Concurrent Writes & Atomic Writes



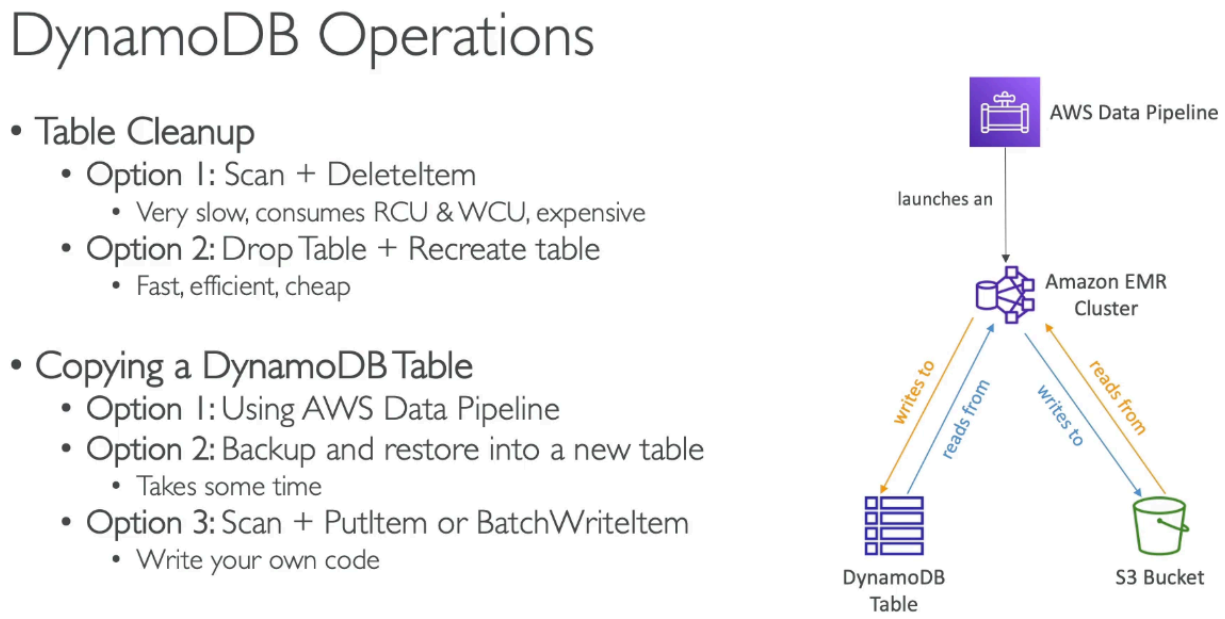
DynamoDB Patterns with S3



- DB only stores up to 400 KB data, for larger files we need S3. Instead store the image URL / metadata of the S3 file acting as a pointer



DynamoDB Operations



- Table Cleanup
    1. Scan + Delete Item – slow, expensive, consumes RCU, WCU
    2. Drop table + recreate – fast, efficient, cheap
- Copying DB table

1. AWS Data Pipeline
2. Backup + restore to new table – takes time
3. Scan + Put Item or Batch Write Item – need to write code

Security + Other

DynamoDB – Security & Other Features

- Security
  - VPC Endpoints available to access DynamoDB without using the Internet
  - Access fully controlled by IAM
  - Encryption at rest using AWS KMS and in-transit using SSL/TLS
- Backup and Restore feature available
  - Point-in-time Recovery (PITR) like RDS
  - No performance impact
- Global Tables
  - Multi-region, multi-active, fully replicated, high performance
- DynamoDB Local
  - Develop and test apps locally without accessing the DynamoDB web service (without Internet)
- AWS Database Migration Service (AWS DMS) can be used to migrate to DynamoDB (from MongoDB, Oracle, MySQL, S3, …)

DynamoDB – Fine-Grained Access Control

- Using Web Identity Federation or Cognito Identity Pools, each user gets AWS credentials
- You can assign an IAM Role to these users with a Condition to limit their API access to DynamoDB
- LeadingKeys – limit row-level access for users on the Primary Key
- Attributes – limit specific attributes the user can see

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:GetItem", "dynamodb:BatchGetItem", "dynamodb:Query",
                "dynamodb:PutItem", "dynamodb:UpdateItem", "dynamodb:DeleteItem",
                "dynamodb:BatchWriteItem"
            ],
            "Resource": "arn:aws:dynamodb:us-west-2:123456789012:table/MyTable",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "dynamodb:LeadingKeys": ["${cognito-identity.amazonaws.com:sub}"]
                }
            }
        }
    ]
}
```

More at: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/specifying-conditions.html

Section 23: AWS Serverless: API Gateway

API Gateway Overview

- Support WebSocket, handle API versioning, different environments, security
    - User Auth via IAM, Cognito, custom authorizer
    - Custom domain name HTTPS security through ACM
        - For edge optimized endpoint certificate must be in us east 1
        - For regional endpoint, certificate must be in API GW region
        - Must setup CNAME or A record in Route 53
- Transform/validate requests/responses, cache API responses, etc…
    - Default timeout of 29 seconds regardless of how long lambda default timeout is
- Integration with:
    - Lambda function: expose REST API backed by Lambda
    - HTTP: ALB, etc…
    - AWS Service
- Must have lambda proxy integration to see the full requests being passed and sent

Endpoint Types

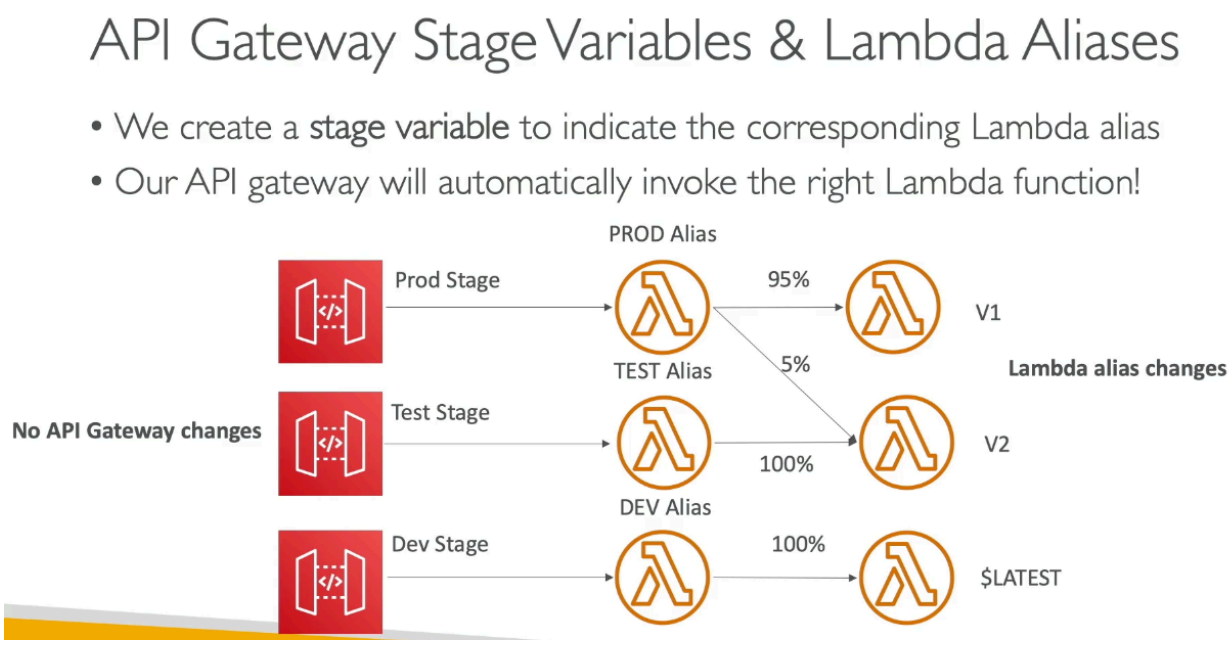
1. Edge Optimized (default): for global clients where requests routed through CF Edge location, but API GW still in 1 region
2. Regional: clients within same region, could manually combine with CloudFront
3. Private: accessed from VPC using VPC endpoint ENI using resource policy

API GW Stages & Deployment

- Making changes in API GW does not mean it's effective, you need to make a deployment to take effect. Changes are deployed to "stages" with any name, with each stage having its own configuration parameters and can be rolled back since deployment history is kept
    - Stages will have new URL with /stage_name

Stage Variables

- Similar to env variables for API GW, can be changed without redeploying
- Use cases: configure HTTP endpoints your stages talk to, pass parameters to Lambda
- Passed to "context" object in AWS Lambda
- Format to access in API GW: ${stageVariables.variableName}



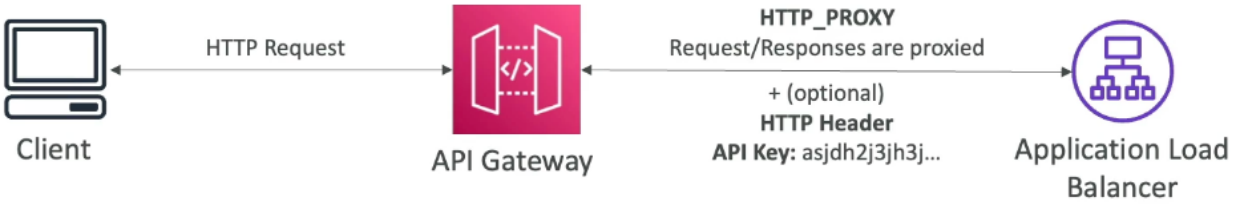
API GW Canary Deployments

- % of traffic to move traffic to a canary channel
- Metrics & Logs are separate
- Possibility to override stage variables for canary
- Blue / green deployment with Lambda + API GW

API GW Integration Types & Mappings

- Integration Types
  - Mock: API GW returns a response without sending request to backend
  - HTTP / AWS (Lambda & AWS Services:
    - Must configure both integration request and response
    - Setup data mapping using mapping templates for the request & response
  - AWS PROXY (lambda proxy)
    - Incoming request from client is Lambda input

- Function is responsible for the logic of request / response
    - All the work done by the lambda function, API GW is just to proxy requests through
- No mapping template, headers, query string parameters… are passed as arguments



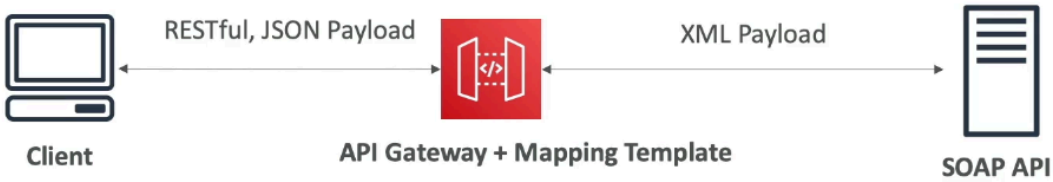
- HTTP PROXY
    - No mapping template
    - HTTP request passed to backend and response is forwarded by API GW
    - Can add HTTP headers if needed

Mapping Templates (AWS & HTTP Integration)

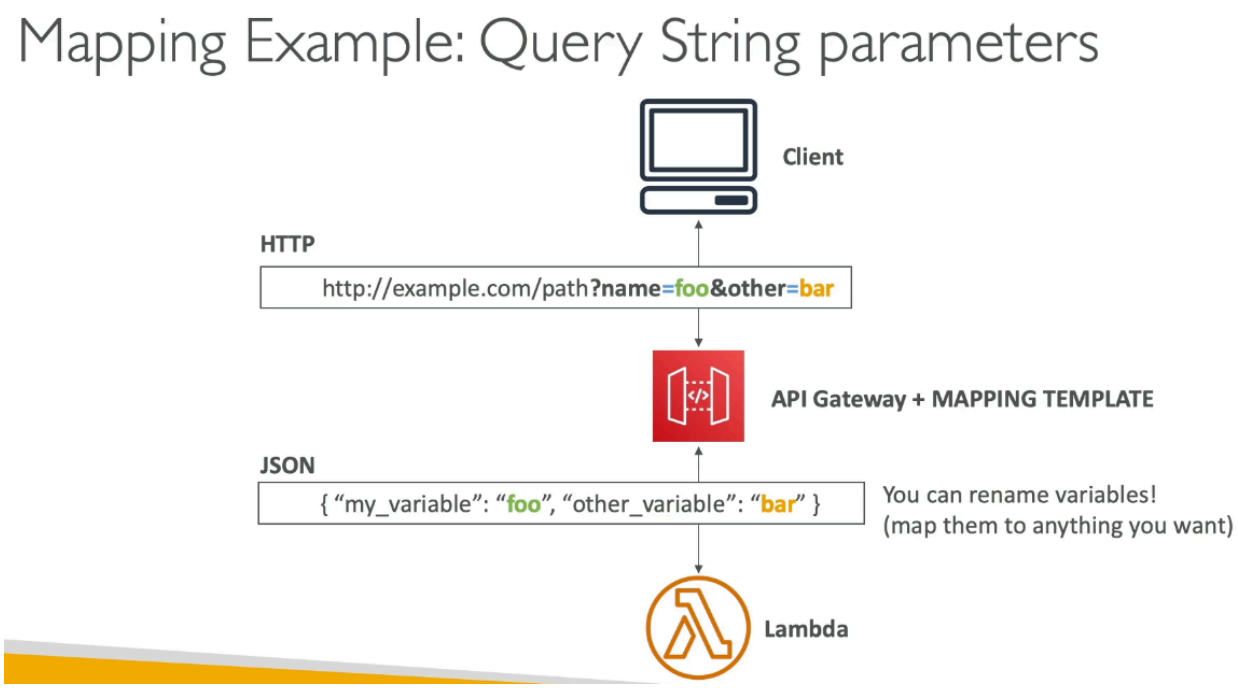
- Used to modify request / responses
- Rename / modify query string parameters, modify body content, add headers
- Can filter output results
- Content type must be application/json or application/xml

Mapping Example: JSON to XML with SOAP

- SOAP API are XML based, whereas REST API are JSON based



- In this case, API Gateway should:
    - Extract data from the request: either path, payload or header
    - Build SOAP message based on request data (mapping template)
    - Call SOAP service and receive XML response
    - Transform XML response to desired format (like JSON), and respond to the user



API GW Open API

- Open API spec is a common way of defining REST APIs where the API definition is code
  - Import existing spec to API GW, defining the method and all method/integration requests/responses
  - Can export current API as OpenAPI spec

Rest API Request Validation – Open API

- Configure API GW to perform validation of API request before integration request
  - 400 error and immediately fails to reduce calls to backend
  - Checks:
    - Required request parameters, query string, headers of incoming request are included and non-blank
    - Request payload follows JSON schema

API GW Caching

- Default TTL 300 seconds, max 1 hour and defined at the stage level
- Possible to override cache settings per method and can be encrypted
- Cache capacity between 0.5 GB to 237 GB, but caching is expensive

Cache Invalidation

- Able to flush entire cache immediately
- Clients can invalidate cache with header: Cache-Control: max-age=0 with proper IAM authorization
    - If no InvalidateCache policy or don't pick require authorization, any client can invalidate the API cache

API GW Usage Plans & API Keys

API Gateway – Usage Plans & API Keys

- If you want to make an API available as an offering ($) to your customers
- Usage Plan:
    - who can access one or more deployed API stages and methods
    - how much and how fast they can access them
    - uses API keys to identify API clients and meter access
    - configure throttling limits and quota limits that are enforced on individual client
- API Keys:
    - alphanumeric string values to distribute to your customers
    - Ex: WBjHxNtoAb4WPKBC7cGm64CBiblb24b4jt8jJHo9
    - Can use with usage plans to control access
    - Throttling limits are applied to the API keys
    - Quotas limits is the overall number of maximum requests

API Gateway – Correct Order for API keys

- To configure a usage plan

1. Create one or more APIs, configure the methods to require an API key, and deploy the APIs to stages.
2. Generate or import API keys to distribute to application developers (your customers) who will be using your API.
3. Create the usage plan with the desired throttle and quota limits.
4. Associate API stages and API keys with the usage plan.

- Callers of the API must supply an assigned API key in the x-api-key header in requests to the API.

API GW Monitoring, Logging & Tracing

- CW Logs
    - Info about request/response body
    - Enable CW logging at stage level (with log level ERROR, DEBUG, INFO)
    - Can override settings on per API basis
- X Ray
    - Enable tracing about requests in API GW
    - X Ray API Gateway + Lambda for full picture
- CW Metrics
    - Metrics are by stage with possibility to enable detailed metrics
    - CacheHitCount & CacheMissCount: efficiency of cache
    - Count: total number API requests in given period
    - IntegrationLatency: time between when API GW relays a request to the backend and when it receives a response from the backend
        - If higher than 29 seconds, timeout from API GW
    - Latency: time between when API GW receives a request from client and when it returns to the client
        - Latency includes integration latency + API GW overhead
        - If higher than 29 seconds, timeout from API GW
    - 4xx Error (client side) & 5xx Error (server side)

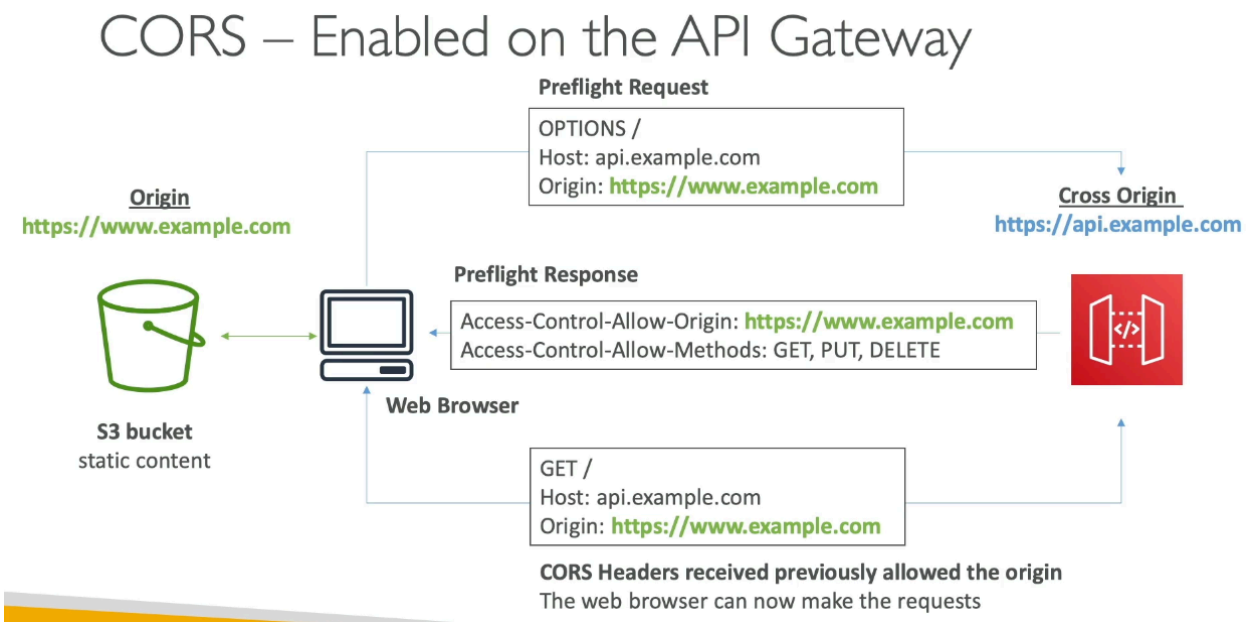
API GW Throttling

- Account limit: API GW throttles requests at 10,000 requests / second across all APIs
    - If one API is overloaded if not limited can cause other APIs to be throttled (similar to Lambda concurrency)
    - Soft limit and can be higher with request
- In case of throttling → 429 too many requests (retry possible)
- Can set stage limit & method limits to improve performance or usage plan to throttle per customer

API GW Errors

- 4xx = Client errors
    - 400: bad request
    - 403: access denied, WAF filtered
    - 429: quota exceeded, throttle
- 5xx = server error
    - 502: bad gateway exception, for an incompatible output returned from Lambda proxy integration backend and occasionally for out of order invocations due to heavy loads
    - 503: service unavailable
    - 504: integration failure, such as API GW timeout after 29 seconds

API GW CORS



- Must be enabled to receive API calls from other domains
- OPTIONS preflight request must contain:
    - Access Control Allow Method
    - Access Control Allow Headers
    - Access Control Allow Origin

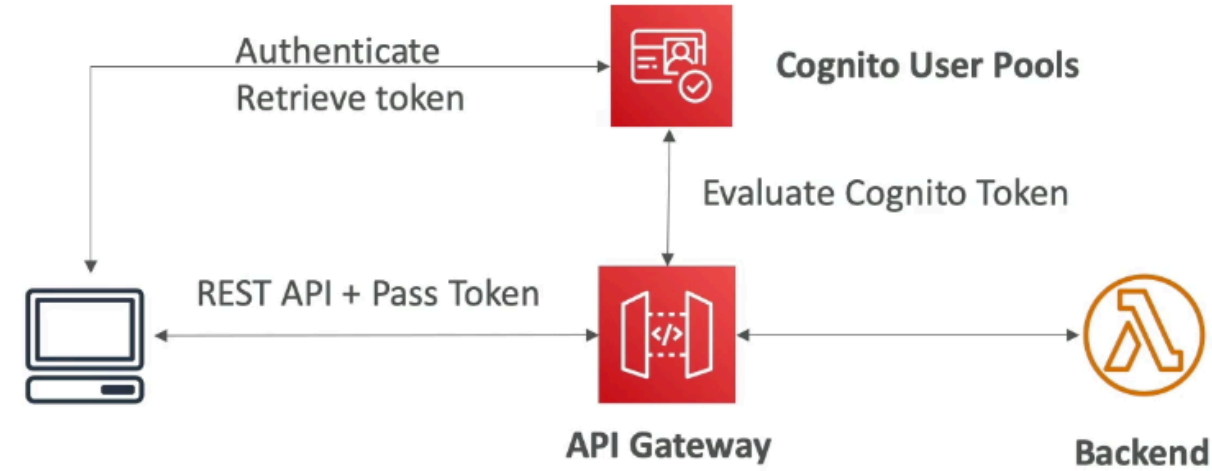
API GW Authentication and Authorization

IAM Permissions

- IAM policy for authentication and authorization done with IAM policy
    - Good to provide access within AWS
- Leverages Sig v4 capability where IAM credentials are in headers

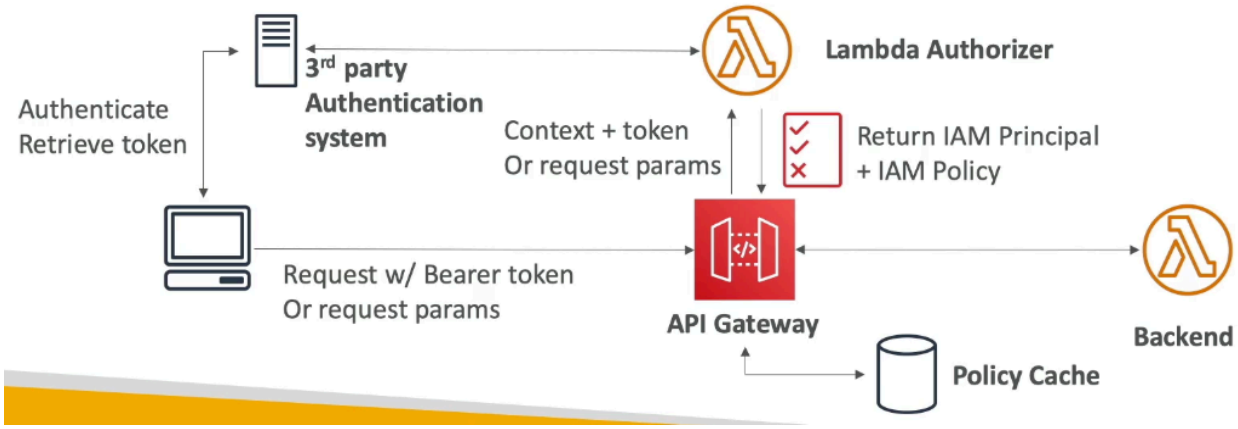
Resource Policies

- Set JSON policy to define who/what can access API GW mainly for cross account access combined with IAM security
    - Allow for specific IP or VPC endpoint



Cognito User Pools

- Manages user lifecycle, token expires automatically
- API GW verifies identity automatically from Cognito with no custom implementation
- Authentication = Cognito user pools, authorization = API GW Methods



Lambda Authorizer

- Token based authorizer (bearer token) – JWT, Oauth (3rd party authentication)
- Request parameter based lambda authorizer (headers, query string, etc…)
- Lambda must return IAM policy for user, result policy is cached
- External authentication, authorization done in lambda function

API GW Security Summary

API Gateway – Security – Summary

- IAM:
  - Great for users / roles already within your AWS account, + resource policy for cross account
  - Handle authentication + authorization
  - Leverages Signature v4
- Custom Authorizer:
  - Great for 3rd party tokens
  - Very flexible in terms of what IAM policy is returned
  - Handle Authentication verification + Authorization in the Lambda function
  - Pay per Lambda invocation, results are cached
- Cognito User Pool:
  - You manage your own user pool (can be backed by Facebook, Google login etc…)
  - No need to write any custom code
  - Must implement authorization in the backend

Rest API vs HTTP API

- HTTP API cheaper than REST API and REST API supports resource policies while HTTP does not

API Gateway – HTTP API vs REST API

- HTTP APIs
  - low-latency, cost-effective AWS Lambda proxy, HTTP proxy APIs and private integration (no data mapping)
  - support OIDC and OAuth 2.0 authorization, and built-in support for CORS
  - No usage plans and API keys
- REST APIs
  - All features (except Native OpenID Connect / OAuth 2.0)

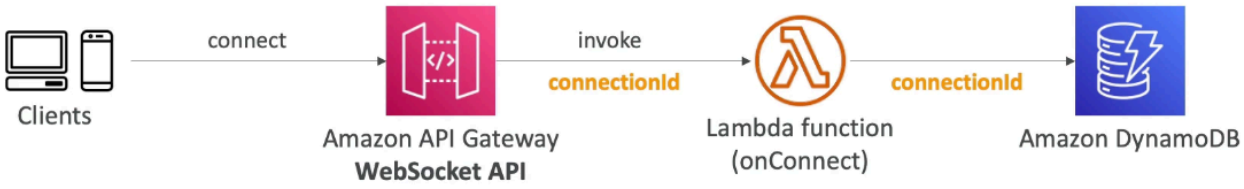
Authorizers	HTTP API	REST API
AWS Lambda	✓	✓
IAM	✓	✓
Resource Policies		✓
Amazon Cognito	✓*	✓
Native OpenID Connect / OAuth 2.0 / JWT	✓	

WebSocket API

- What is websocket? 2 way interactive communication between user's browser and server, where server can push info to the client (is persistent connection)
    - Enables stateful application use cases
- Often used for real time apps

Connecting to API

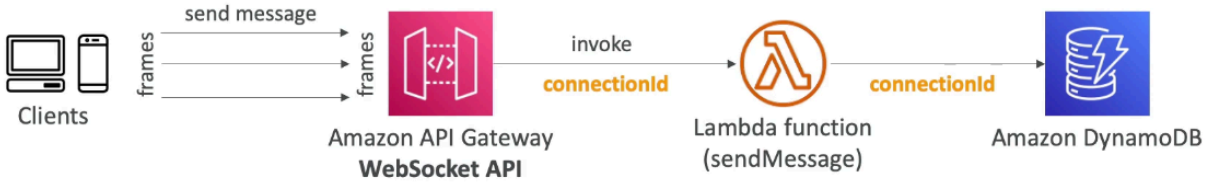
WebSocket URL
wss://[some-uniqueid].execute-api.[region].amazonaws.com/[stage-name]

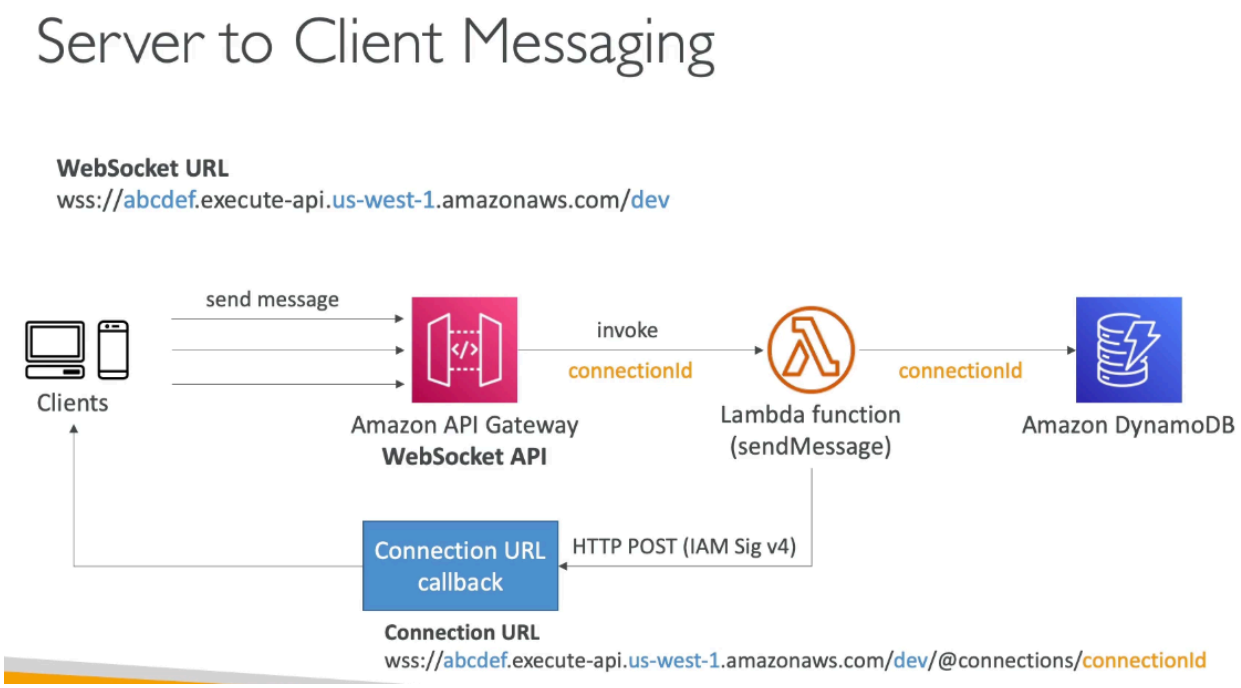


- Connection ID is persistent as long as the user is connected to API GW

Client to Server Messaging
ConnectionID is re-used

WebSocket URL
wss://abcdef.execute-api.us-west-1.amazonaws.com/dev

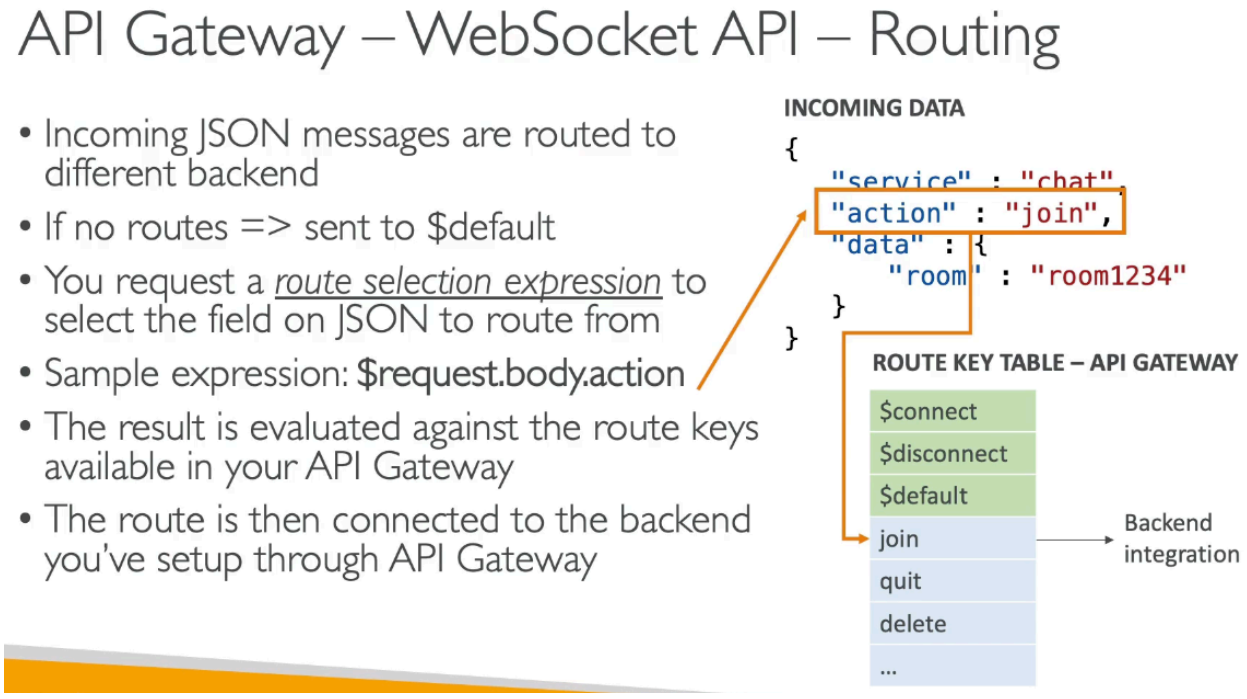




Connection URL Operations

- POST, GET, DELETE

Routing

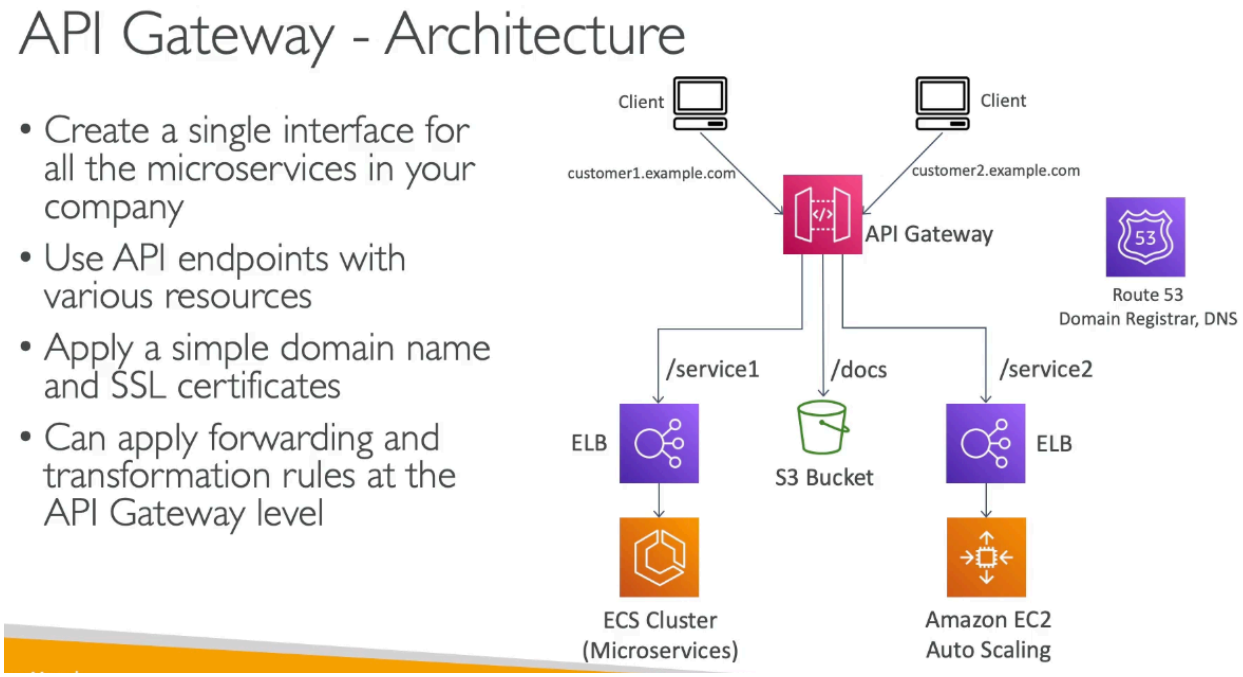


- Incoming JSON messages routed to different backend, if no routes, sent to $default

- You can request a route selection expression to select the field on JSON to route them

API GW Architecture

- API endpoints to various resources while providing an external unified URL, hiding the complexity



Section 24: AWS CICD: CodeCommit, CodePipeline, CodeBuild, CodeDeploy

AWS CICD

- Continuous Integration
  - Push code often, test / build server for pass / fail, find and fix bugs, deploy often
- Continuous Delivery
  - Ensures software can be released reliably whenever needed, ensuring deployments often and quick

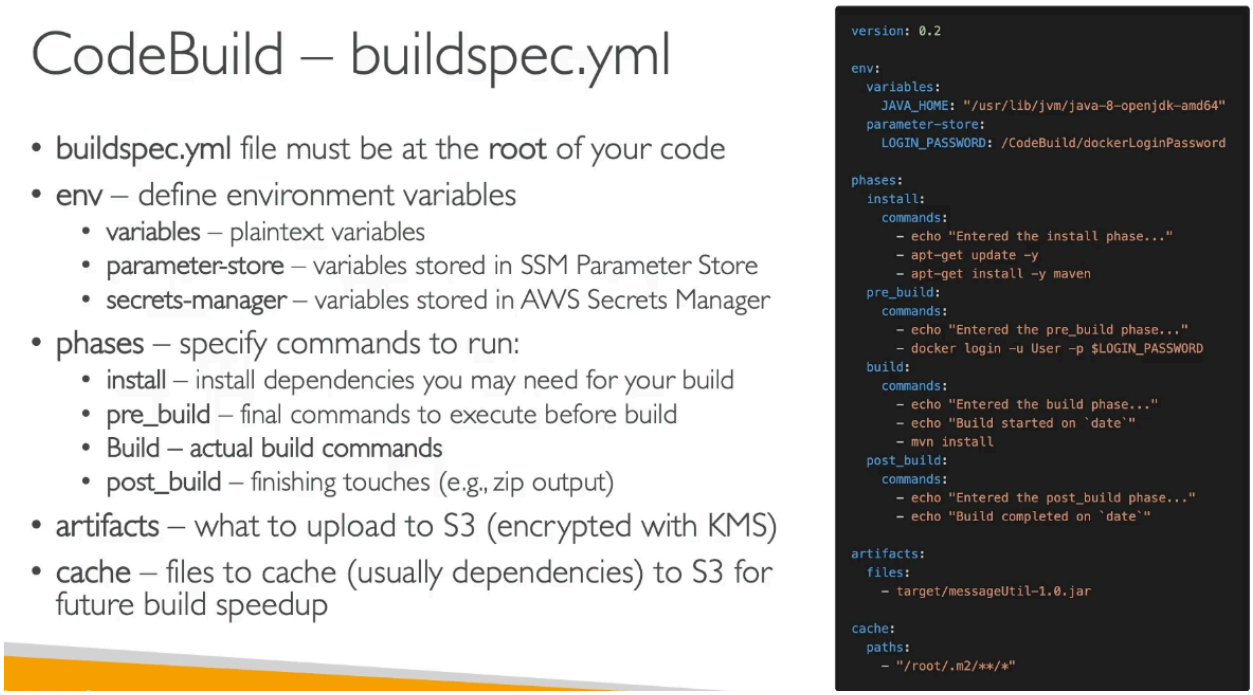
CodeCommit Overview

- Version control via Git for centralized backup in a repository
- Private Git repos, no size limit, fully managed/ highly available, code only in AWS, security, integration with other CI tools
- Security: SSH / HTTPS, IAM, encryption, cross account access via IAM role

CodePipeline Overview

- Visual workflow for CICD
- Stages:
    - Source: CodeCommit, ECR, S3, GitHub
    - Build, Test: CodeBuild
    - Deploy: CodeDeploy, Elastic Beanstalk, CF, ECS, S3…
    - Invoke: Lambda, step functions
    - Stages have multiple action groups sequentially or in parallel
- Artifacts:
    - Stored in S3 bucket and passed to next stage
- CloudWatch Events (Amazon EventBridge) to see failed or canceled stages or CloudTrail to audit
    - If a pipeline can't perform an action, it needs IAM roles

CodeBuild Overview

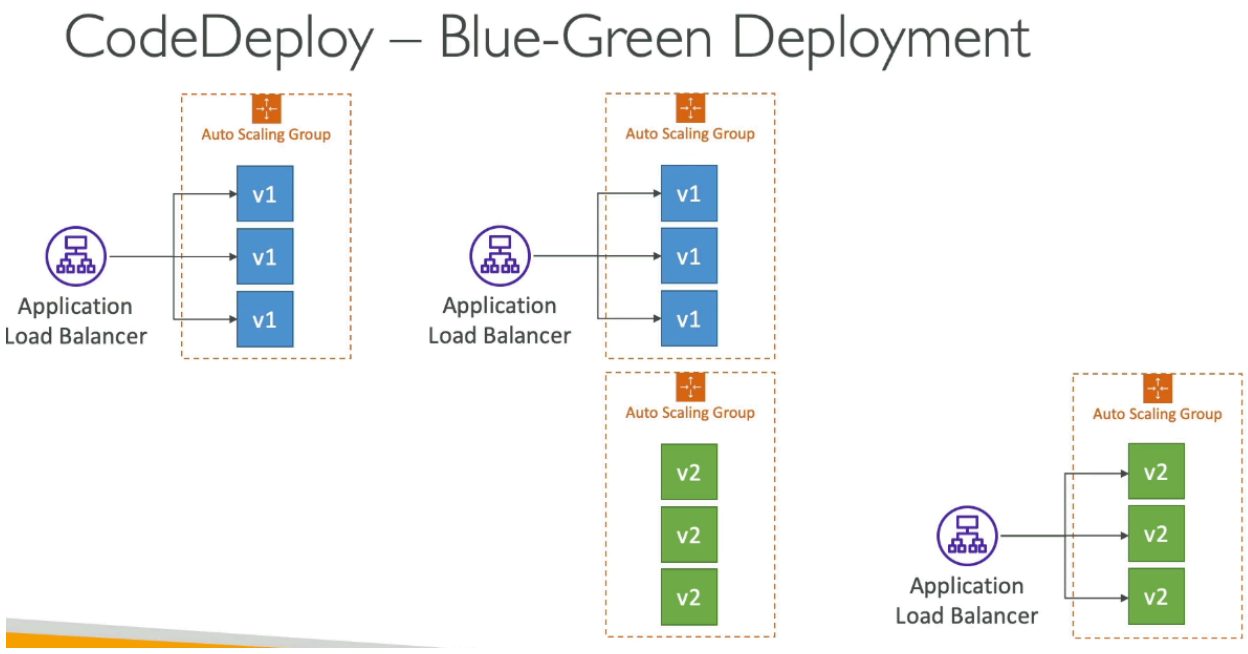


- Source code → build instructions in buildspec.yml
    - Can cache files in build stage in S3
    - Output logs in S3 & CloudWatch Logs, and use CW Metrics, EventBridge, or Alarms for monitoring and trigger notifications
- Build projects can be defined within CodePipeline or CodeBuild
- Support many environments + Docker for any unsupported environment

CodeDeploy Overview

- Deployment service to automate application deployment
    - Deploy to EC2 instances, on premise, lambda functions, ECS
    - Automated rollback in case of failure and gradual deployment control
    - appspec.yml

EC2 / On Prem



- In place or blue green deployments
- Must run CodeDeploy Agent on target instances
- Deployment speed:
    - All at once, half at a time (50% capacity), one at a time (slowest, lowest availability), custom %

CodeDeploy Agent

- Must be running on EC2 instances as prerequisite, and can be installed/updated automatically via Systems Manager
    - EC2 instances need IAM permissions to access S3 to get deployment bundles
    - EC2 will pull code from S3 bucket

Lambda Platform

- CodeDeploy can automate traffic shift for lambda alias
- Feature integrated within SAM framework

- CodeDeploy varies % until 100% moves from old to new version
    - Linear: grow traffic every N minutes until 100%
    - Canary: try x% then 100% after x minutes
    - All at once

ECS Platform

- Automate deployment of new ECS task definition, only works for blue / green deployments
    - Requires ALB with ECS cluster target group
    - Linear, Canary, All at once

CodeDeploy for EC2 & ASG

EC2

- Need appspec.yml + deployment strategy
- Can use hooks to verify the deployment after each deployment phase

ASG

- In place deployment
    - Updates existing EC2 instances with newly created EC2 instances by ASG will get automated deployments
- Blue Green
    - New ASG created (copied settings), must be using ELB
    - Choose how long to keep old EC2 instances (old ASG)

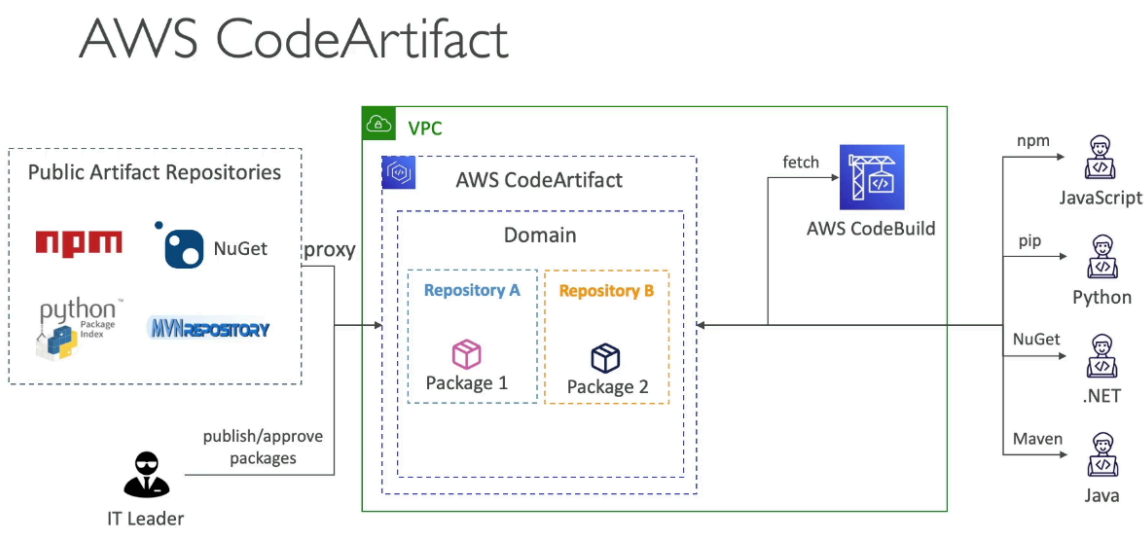
Redeploy & Rollbacks

- Rollback = redeploy a previously deployed revision of application
- Deployments can be rolled back manually or automatically if it fails or CloudWatch alarms triggered
    - If rollback occurs, CodeDeploy redeploys last known good revision as a new deployment (not restored version)
    - Disable rollbacks to not perform rollback for deployment

CodeStar OverView

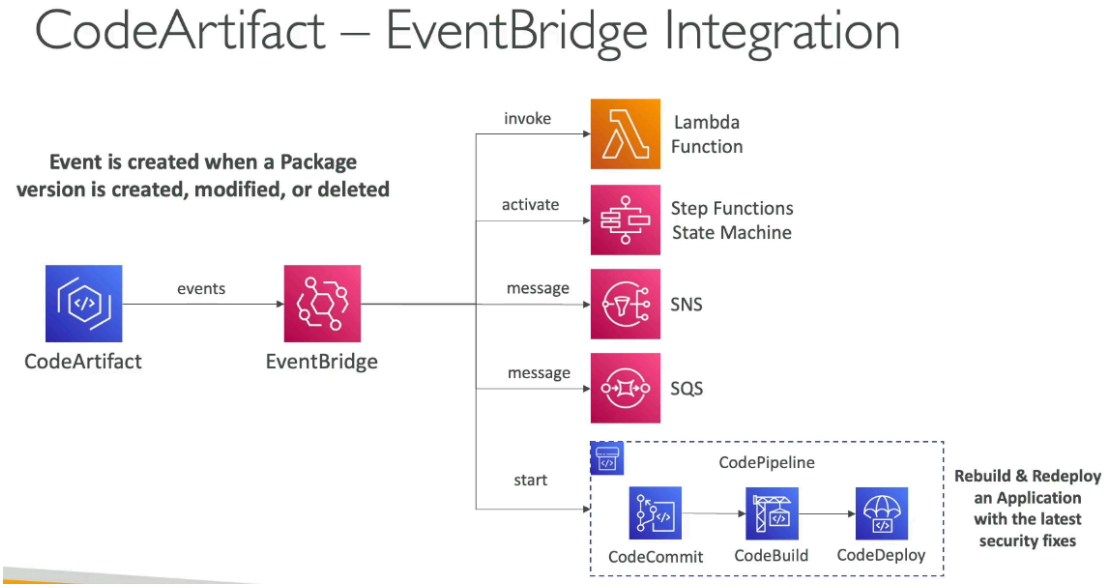
- Integrates all CICD ready projects for EC2, Lambda, Elastic Beanstalk with issue tracking integration with JIRA and web IDE
    - One dashboard for all components
    - Free service, paid for other service usage

CodeArtifact Overview



- Software packages depend on each other to be built (code dependencies)
    - Storing and retrieving them called artifact management
- CodeArtifact is a secure, scalable, cost effective artifact management
    - Developers and CodeBuild can retrieve dependencies straight from CodeArtifact

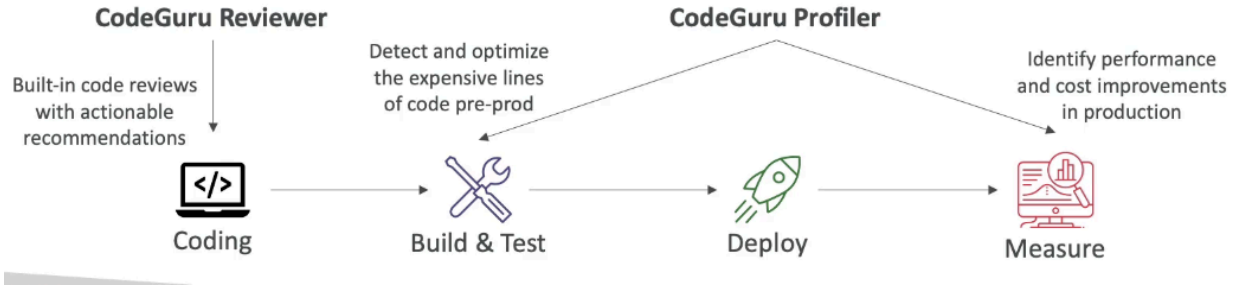
CodeArtifact & EventBridge Integration



CodeArtifact Resource Policy

- Can authorize another account access to packages in CodeArtifact

CodeGuru Overview



- ML powered service for automated code reviews and application performance recommendations
- 2 functionalities
    - Reviewer: automated code review for static code analysis (development)
        - Identify security vulnerabilities, etc…
    - Profiler: visibility/recommendations about application performance during runtime (production)
        - Helps understand runtime behavior of application
        - Features:
            - Identify/remove code inefficiencies, improve application performance, decrease compute costs, heap summary, anomaly detection
        - AWS or on premise with minimal overhead on application

CodeGuru Agent Configuration

Amazon CodeGuru – Agent Configuration

- MaxStackDepth – the maximum depth of the stacks in the code that is represented in the profile
    - Example: if CodeGuru Profiler finds a method A, which calls method B, which calls method C, which calls method D, then the depth is 4
    - If the MaxStackDepth is set to 2, then the profiler evaluates A and B
- MemoryUsageLimitPercent – the memory percentage used by the profiler
- MinimumTimeForReportingInMilliseconds – the minimum time between sending reports (milliseconds)
- ReportingIntervalInMilliseconds – the reporting interval used to report profiles (milliseconds)
- SamplingIntervalInMilliseconds – the sampling interval that is used to profile samples (milliseconds)
    - Reduce to have a higher sampling rate

AWS Cloud9

- Cloud / web based IDE, prepackaged with tools
- Deployed on EC2 instances

Section 25: AWS Serverless: SAM – Serverless Application Model

SAM Overview

- Serverless Application model that is a framework for developing and deploying serverless application code via YAML
    - Generates complex CF, supports CF, uses CodeDeploy to deploy lambda functions, can help run Lambda, API GW, dynamoDB locally

Recipe

- Transform header to indicate SAM template
- Write code via SAM constructs
- Package and deploy: sam deploy
    - SAM template + code → CF template + code → S3 and deploy via CF
- Quickly sync local changes w AWS Lambda with sam sync

SAM Accelerate

SAM Accelerate (sam sync) – Examples

- sam sync (no options)
    - Synchronize code and infrastructure
- sam sync --code
    - Synchronize code changes without updating infrastructure (bypass CloudFormation, update in seconds)
- sam sync --code --resource AWS::Serverless::Function
    - Synchronize only all Lambda functions and their dependencies
- sam sync --code --resource-id HelloWorldLambdaFunction
    - Synchronize only a specific resource by its ID
- sam sync --watch
    - Monitor for file changes and automatically synchronize when changes are detected
    - If changes include configuration, it uses sam sync
    - If changes are code only, it uses sam sync --code

- Reduce latency while deploying to AWS (sam sync)
    - Sync code changes to AWS without updating infrastructure

SAM Policy Templates

- List of templates to apply permissions to lambda functions
    - Basically an IAM policy attached to lambda functions in SAM framework
- Important examples:
    - S3ReadPolicy: read only permissions to objects in S3
    - SQSPollerPolicy: allows poll an SQS queue
    - DynamoDBCrudPolicy: CRUD = create read update delete

SAM + CodeDeploy

SAM and CodeDeploy

- AutoPublishAlias
    - Detects when new code is being deployed
    - Creates and publishes an updated version of that function with the latest code
    - Points the alias to the updated version of the Lambda function
- DeploymentPreference
    - Canary, Linear, AllAtOnce
- Alarms
    - Alarms that can trigger a rollback
- Hooks
    - Pre and post traffic shifting Lambda functions to test your deployment

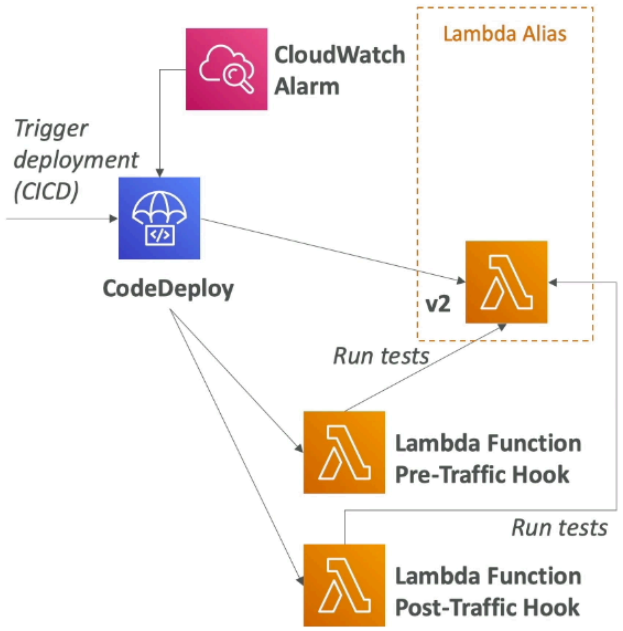
```yaml
Resources:
MyLambdaFunction:
  Type: AWS::Serverless::Function
  Properties:
    Handler: index.handler
    Runtime: nodejs12.x
    CodeUri: s3://bucket/code.zip

    AutoPublishAlias: live

    DeploymentPreference:
      Type: Canary10Percent10Minutes
      Alarms:
        # A list of alarms that you want to monitor
        - !Ref AliasErrorMetricGreaterThanZeroAlarm
        - !Ref LatestVersionErrorMetricGreaterThanZeroAlarm
      Hooks:
        # Validation Lambda functions that are run before & after traffic shifting
        PreTraffic: !Ref PreTrafficLambdaFunction
        PostTraffic: !Ref PostTrafficLambdaFunction
```

SAM and CodeDeploy

- SAM framework natively uses CodeDeploy to update Lambda functions
- Traffic Shifting feature
- Pre and Post traffic hooks features to validate deployment (before the traffic shift starts and after it ends)
- Easy & automated rollback using CloudWatch Alarms

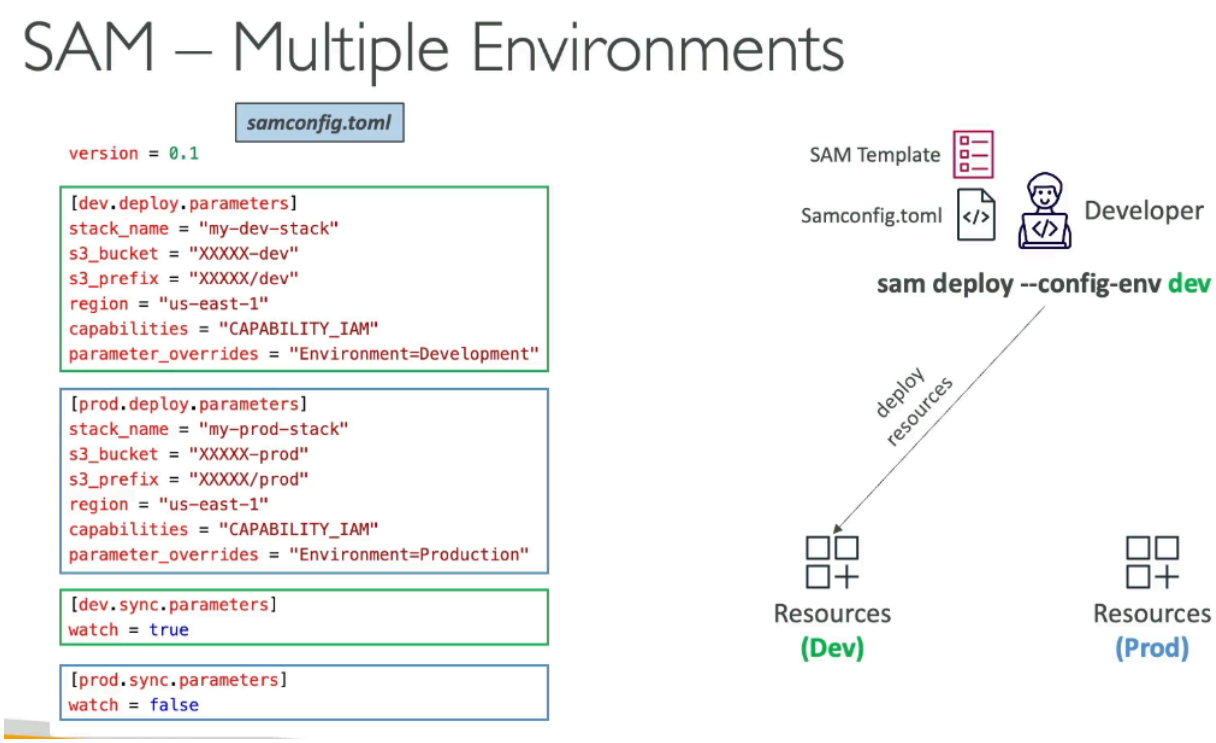


- Natively uses CodeDeploy to update lambda
- Traffic shifting feature, pre/post traffic hooks to validate deployment (before traffic shift starts and after it ends), automated rollback via CloudWatch Alarms

SAM – Local Capabilities

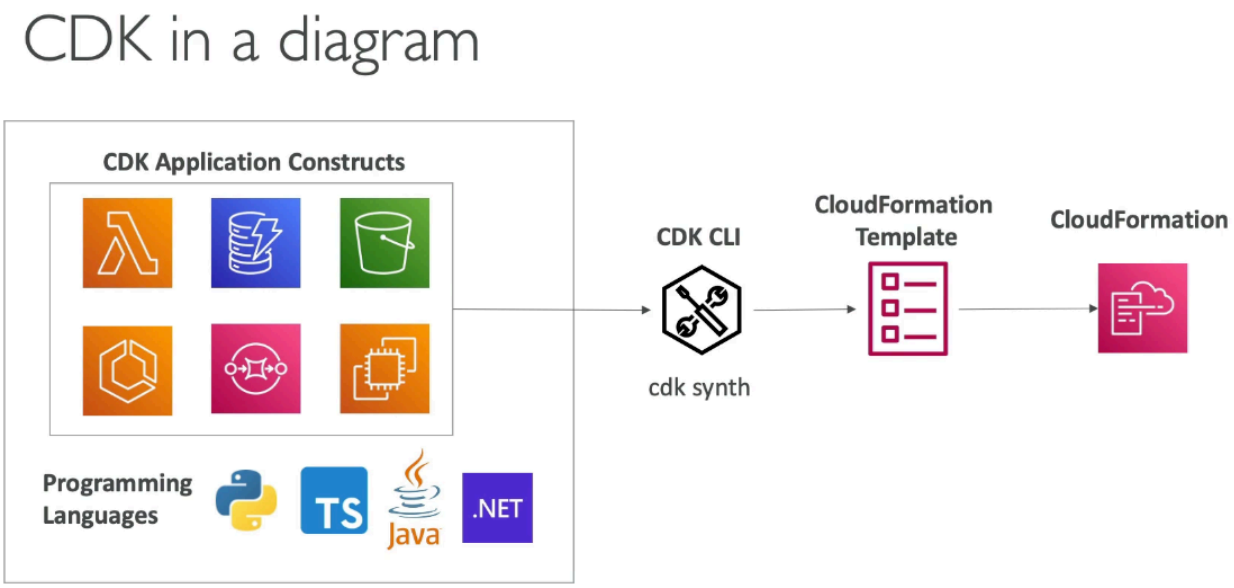
- Locally start lambda via sam local start-lambda
    - Starts a local endpoint that emulates lambda and can run automated tests against the local endpoint
- Locally invoke lambda via sam local invoke
    - Invoke lambda function with payload once and quit after invocation completes
    - Helps with generating test cases
- Locally start API GW endpoint via sam local start-api
    - Starts local HTTP server that hosts all functions, where changes to functions are automatically reloaded
- Generate AWS events for lambda via sam local generate-event
    - Generate sample payloads for event sources

SAM – Multiple Environments



Section 26: Cloud Development Kit (CDK)

CDK Overview

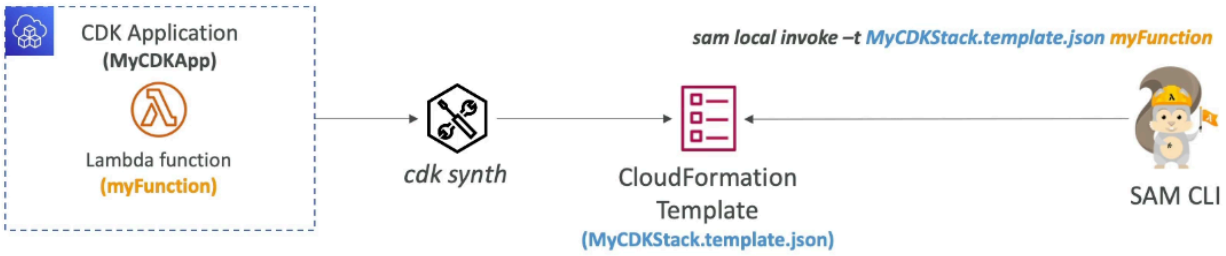


- Define infrastructure using code, compiled into CF template
    - Need to run cdk bootstrap once per region per account to create all necessary stuff to run cdk
- Contains high level components called constructs
    - Can deploy infrastructure and application code together, great for lambda, docker containers

CDK vs SAM

- SAM:
    - Serverless focused in JSON or YAML
    - Good for lambda and leverages CF
- CDK:
    - All services, many languages, leverages CF

CDK + SAM



- Used to locally test CDK apps via cdk synth

CDK Constructs

- Components that encapsulates everything CDK needs to create final CF stack
  - Can represent single AWS resource or multiple related resources
- AWS Construct Library
  - Collection of constructs included in CDK for every resource
  - 3 levels of constructs available
- Construct Hub contains additional constructs from 3rd parties

Layer 1 Constructs (L1)

- CFN resources, starts with “Cfn”, which represents all resources directly available in CF
  - Must explicitly configure all resource properties
- Periodically generated from CF resource specification

Layer 2 Constructs (L2)

- Higher level (intent based API) with similar functionality as L1 but has defaults and boilerplate (don’t need to know all the details about resource properties)
  - Does not start with Cfn
  - Provides methods that make it simpler to work with the resource

Layer 3 Constructs (L3)

aws-apigateway.LambdaRestApi represents an API Gateway backed by a Lambda function
aws-ecs-patterns.ApplicationLoadBalancerFargateService which represents an architecture that includes a Fargate cluster with Application Load Balancer

```
const api = new apigateway.LambdaRestApi(this, 'myapi', {
  handler: backend,
  proxy: false
});

const items = api.root.addResource('items');
items.addMethod('GET');  // GET /items
items.addMethod('POST'); // POST /items

const item = items.addResource('{item}');
item.addMethod('GET');   // GET /items/{item}

item.addMethod('DELETE', new apigateway.HttpIntegration('http://amazon.com'));
```

- Called patterns, represents multiple related resources to help with common tasks in AWS

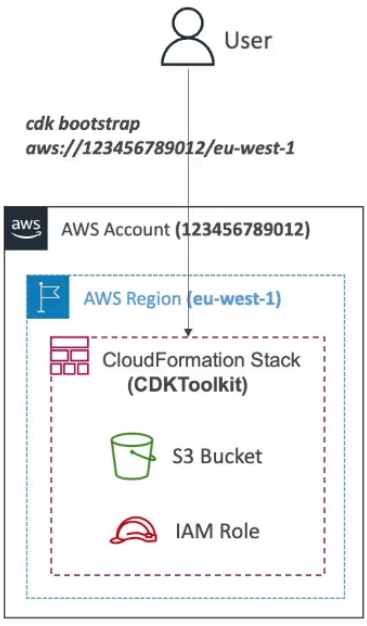
CDK Commands & Bootstrapping

CDK – Important Commands to know

Command	Description
npm install -g aws-cdk-lib	Install the CDK CLI and libraries
cdk init app	Create a new CDK project from a specified template
cdk synth	Synthesizes and prints the CloudFormation template
cdk bootstrap	Deploys the CDK Toolkit staging Stack
cdk deploy	Deploy the Stack(s)
cdk diff	View differences of local CDK and deployed Stack
cdk destroy	Destroy the Stack(s)

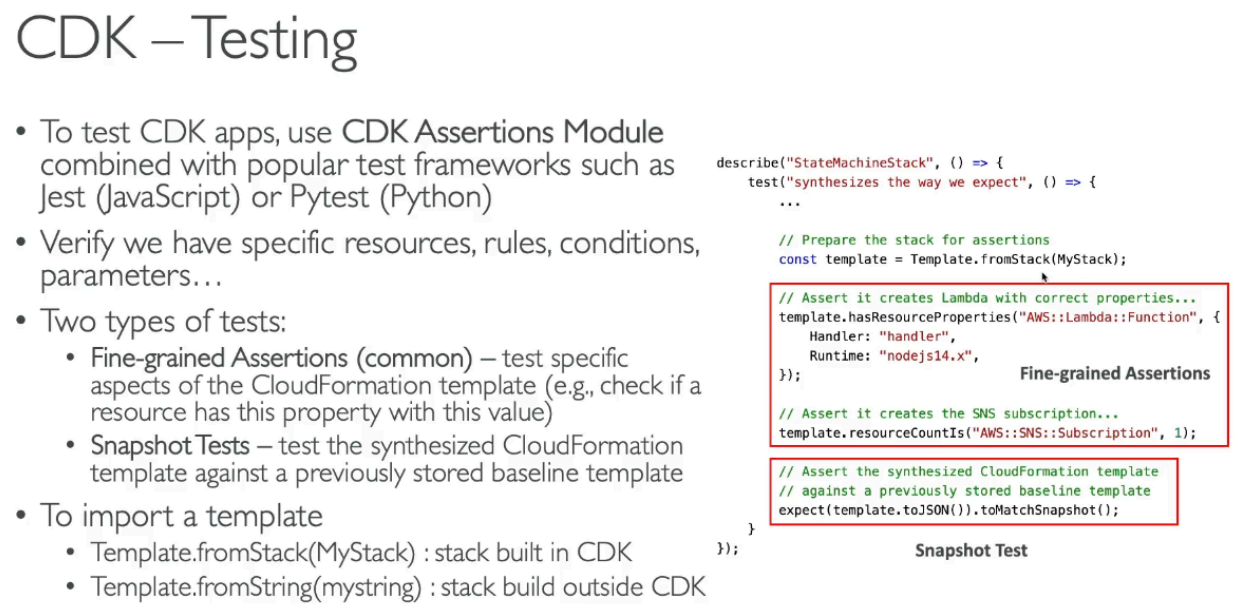
CDK – Bootstrapping

- The process of provisioning resources for CDK before you can deploy CDK apps into an AWS environment
- AWS Environment = account & region
- CloudFormation Stack called CDKToolkit is created and contains:
  - S3 Bucket – to store files
  - IAM Roles – to grant permissions to perform deployments
- You must run the following command for each new environment:
  - *cdk bootstrap aws://<aws_account>/<aws_region>*
- Otherwise, you will get an error "Policy contains a statement with one or more invalid principal"



CDK Unit Testing

- Template.fromStack and Template.fromtString important for exam

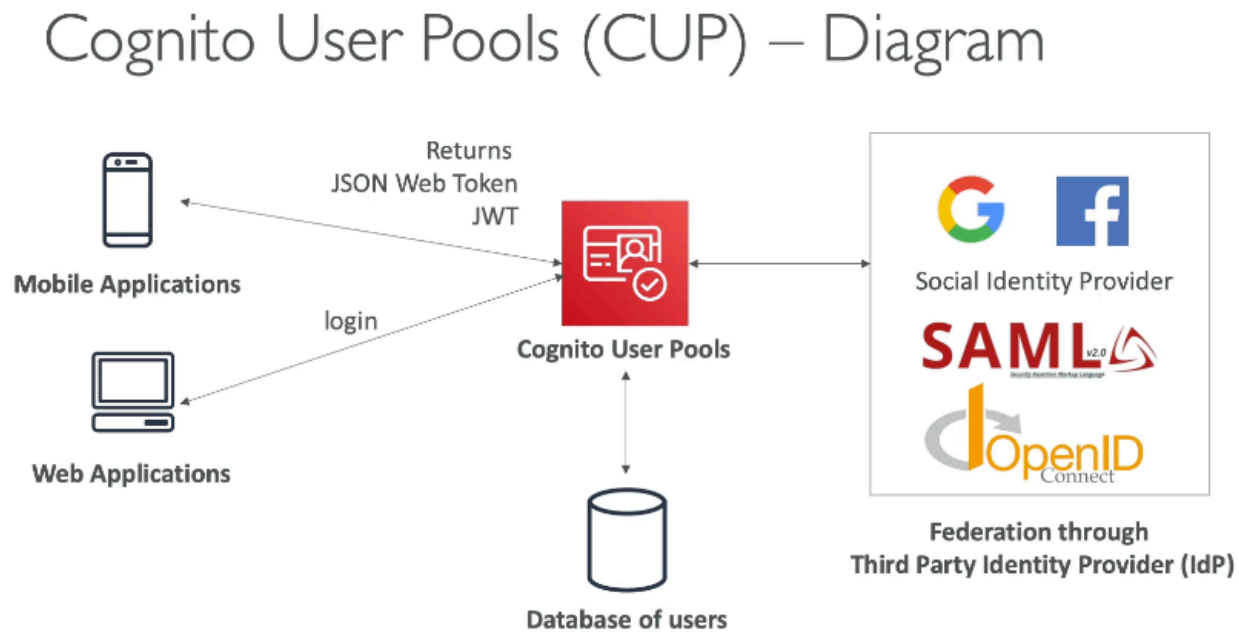


Section 27: Cognito

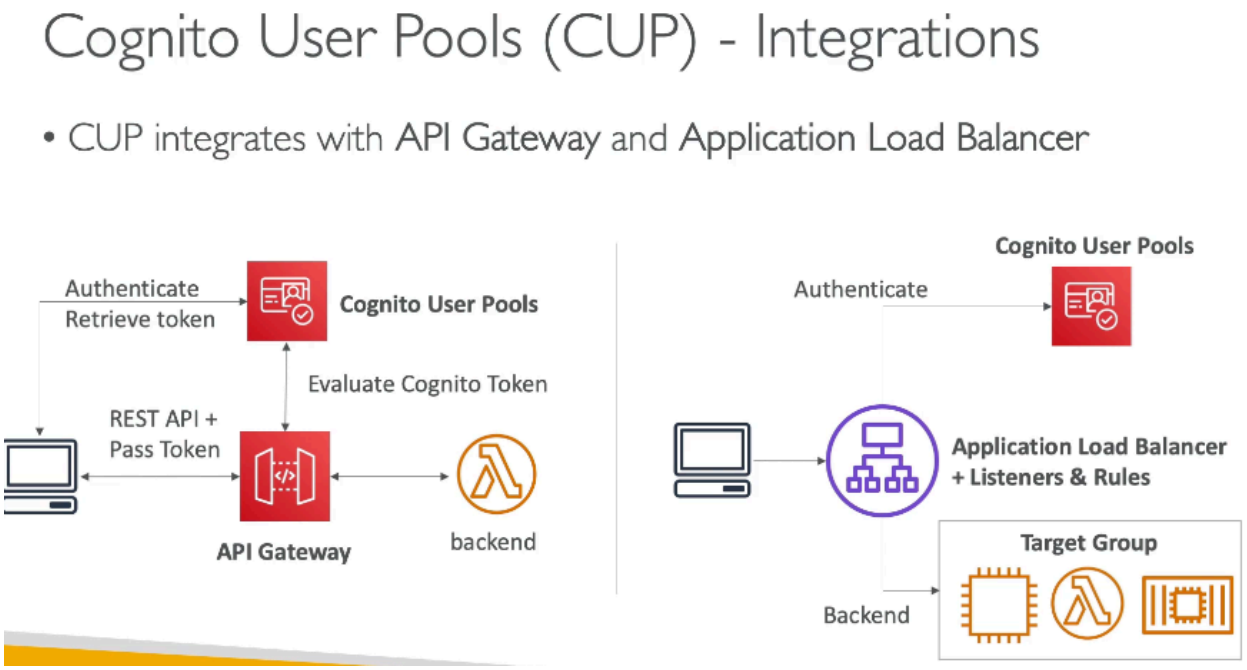
Cognito Overview

- Give users an identity to interact with application
  - Provides a Cognito Hosted UI for users to sign in and sign up
- Cognito User Pools:
  - Sign in functionality for users, integration with API GW & ALB
- Cognito Identity Pools
  - Temporary AWS credentials to access AWS resources directly
  - Integrate with Cognito User pools as identity provider
- Cognito vs IAM: “hundreds of users”, “mobile users”, authenticate with SAML

Cognito User Pools



- Serverless DB of users for web / mobile apps
- Simple login (username, password) with password reset, email/phone verification, MFA, 3rd party federated identities (Google, FB, etc…)
    - Can block users if credentials are compromised elsewhere
    - Login sends back as JWT



- Integrates with API GW and ALB

Lambda Triggers

Cognito User Pools – Lambda Triggers

- CUP can invoke a Lambda function synchronously on these triggers:

User Pool Flow	Operation	Description
Authentication Events	Pre Authentication Lambda Trigger	Custom validation to accept or deny the sign-in request
	Post Authentication Lambda Trigger	Event logging for custom analytics
	Pre Token Generation Lambda Trigger	Augment or suppress token claims
Sign-Up	Pre Sign-up Lambda Trigger	Custom validation to accept or deny the sign-up request
	Post Confirmation Lambda Trigger	Custom welcome messages or event logging for custom analytics
	Migrate User Lambda Trigger	Migrate a user from an existing user directory to user pools
Messages	Custom Message Lambda Trigger	Advanced customization and localization of messages
Token Creation	Pre Token Generation Lambda Trigger	Add or remove attributes in Id tokens

- CUP can invoke lambda function synchronously:
    - Authentication events (pre/post), sign up (pre/post), messages, token creation

Hosted Authentication UI

- Hosted Authentication UI to handle sign up/in with built in integration with logins
    - Can be customized with logo and CSS

Hosted UI Custom Domain

- Custom domains need ACM certificate in us east 1 and defined at “App Integration” section

Adaptive Authentication

- Block sign ins or require additional MFA for suspicious logins by examining each sign in attempt and giving a risk score (based on device, IP, etc…)
    - Integration with CloudWatch Logs (sign in attempts, risk score…)

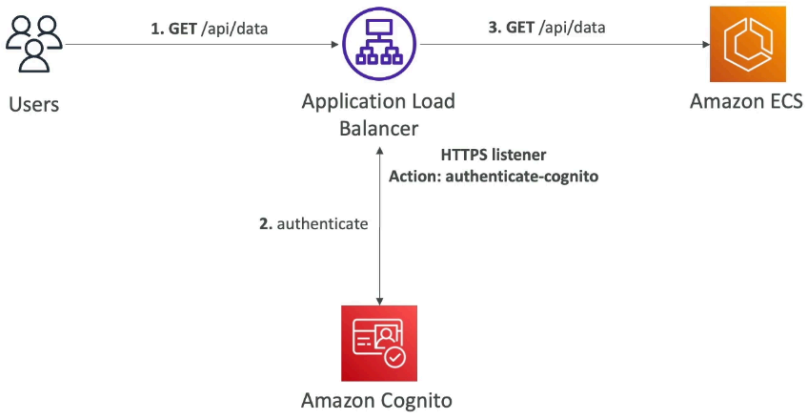
Decoding a ID Token; JWT – JSON Web Token

- CUP issues JWT tokens (Base64 encoded):
  - Header
  - Payload
  - Signature
- The signature must be verified to ensure the JWT can be trusted
- Libraries can help you verify the validity of JWT tokens issued by Cognito User Pools
- The Payload will contain the user information (sub UUID, given_name, email, phone_number, attributes…)
- From the sub UUID, you can retrieve all users details from Cognito / OIDC

ID JWT Token Payload

ALB User Authentication

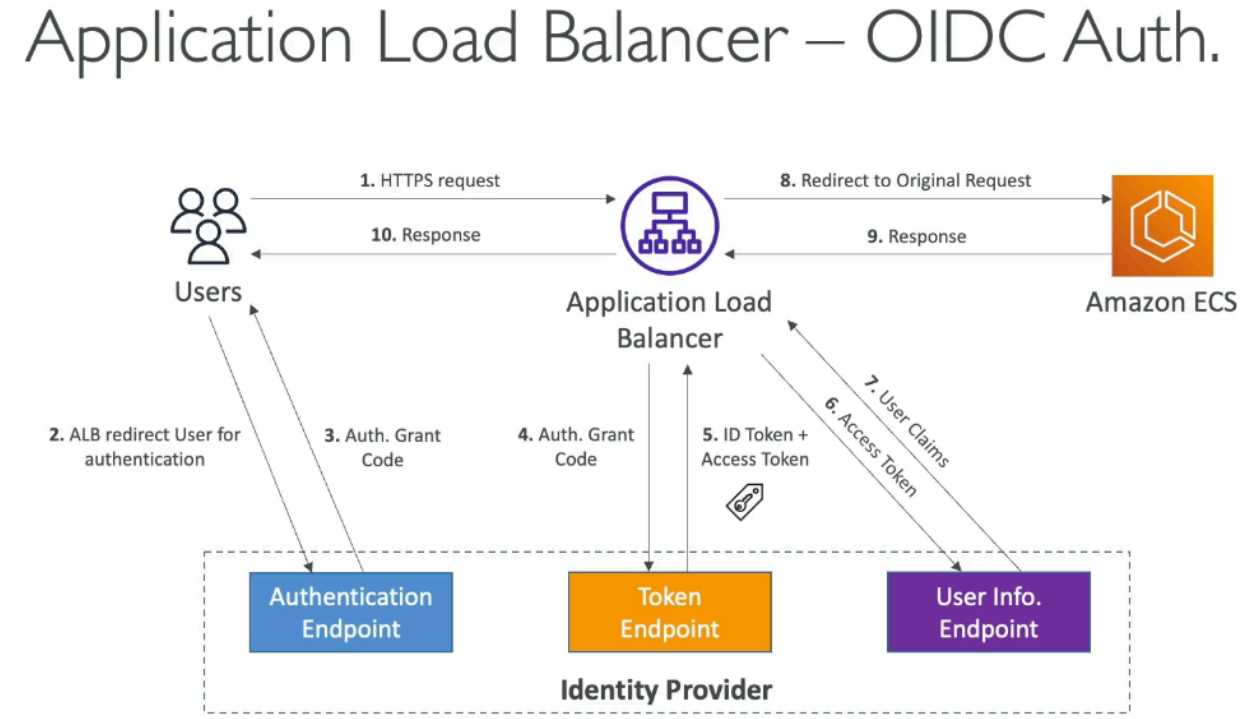
Application Load Balancer – Cognito Auth.



- ALB authenticates users instead of the application via:
  - Identity Provider (IdP): OpenID Connect (OIDC) compliant
  - Cognito User Pools: social IdPs like Google, Facebook, etc…
- Must use HTTPS listener to set OIDC or Cognito rules
  - First default action is to authenticate, and OnUnAuthenticatedRequest choose authenticate (default), deny or allow

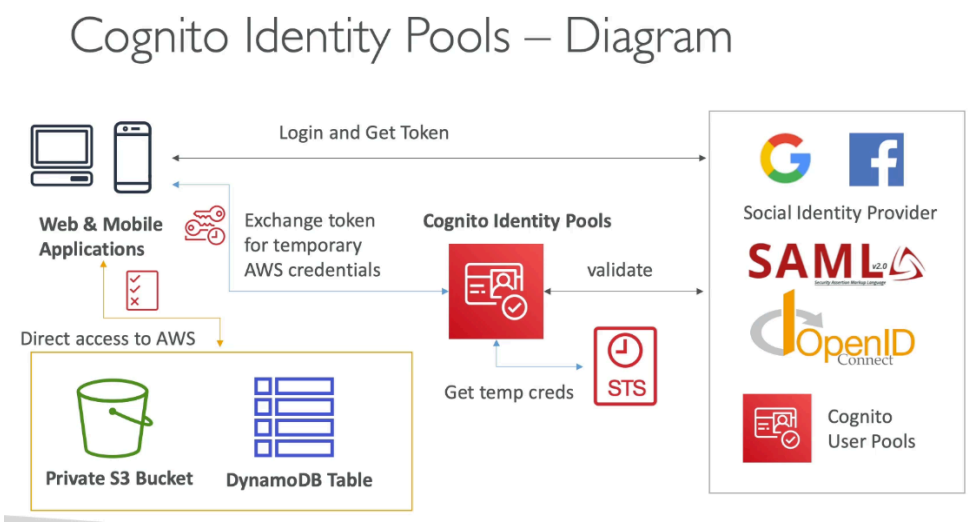
1. Create Cognito User Pool, client and domain. Make sure ID is returned, set URL redirections, allow Cognito User Pool Domain on IdP app's callback URL

ALB OIDC Auth

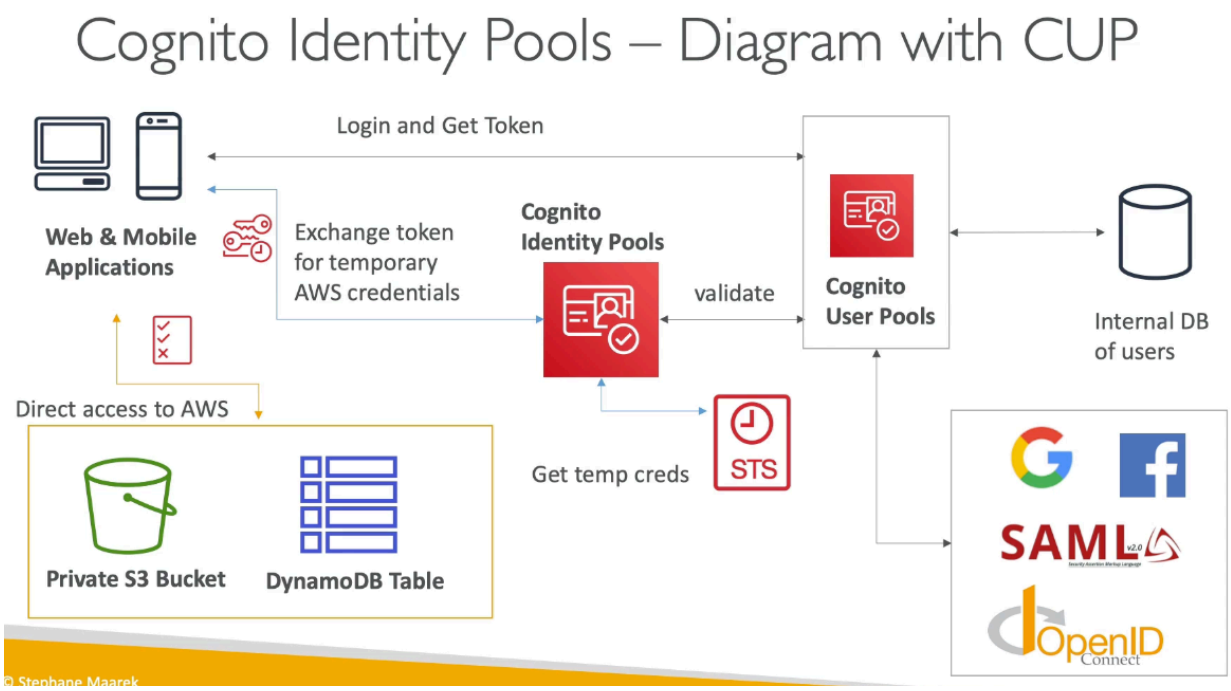


- Need to set Client ID & Client Secret and allow redirect from OIDC to ALB DNS name and CNAME (DNS alias of app)

Cognito Identity Pools



- Get identities for outside users for temporary AWS credentials
    - Identity pool can be public providers, Amazon Cognito, OpenID, custom login
        - Cognito Identity Pools allow for unauthenticated guest access
- Users can access AWS Services directly or via API GW
    - IAM policies applied to credentials are defined in Cognito and can be customized for fine grained control



IAM Roles

- Default IAM roles for authenticated and guest users where rules are defined based on user ID
    - Partition user access using policy variables
- IAM credentials are obtained by Cognito Identity Pools through STS and roles must have a trust policy of Cognito Identity Pools

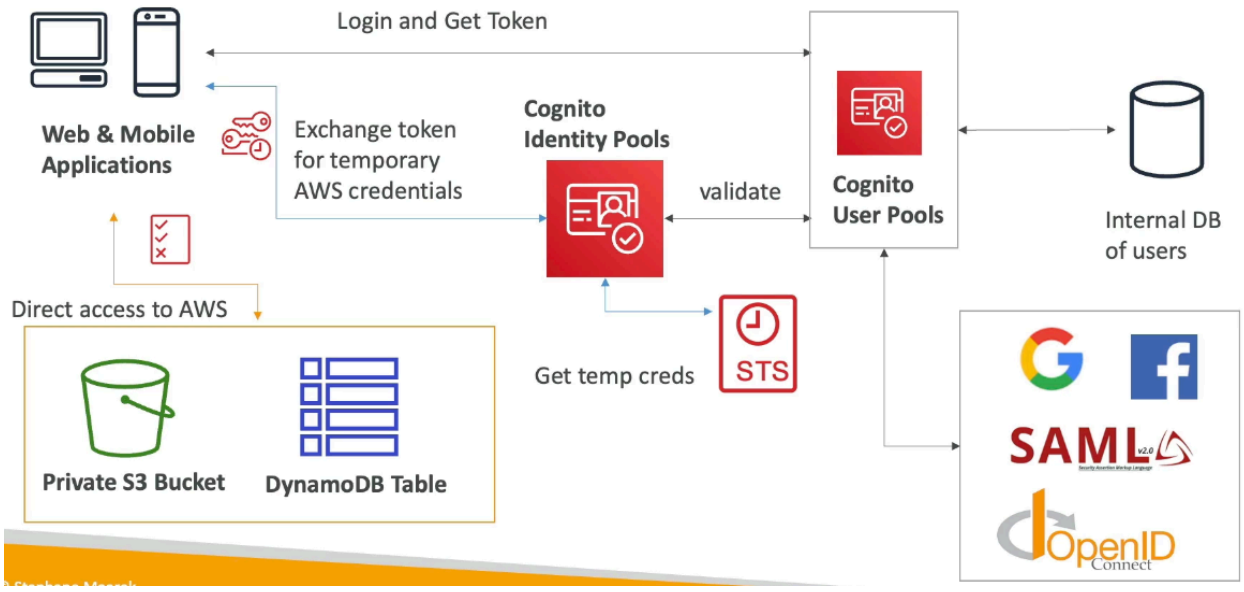
Cognito User Pools vs Identity Pools

- User pools are for applications, identity pools are for AWS resources

Cognito User Pools vs Identity Pools

- Cognito User Pools (for authentication = identity verification)
  - Database of users for your web and mobile application
  - Allows to federate logins through Public Social, OIDC, SAML…
  - Can customize the hosted UI for authentication (including the logo)
  - Has triggers with AWS Lambda during the authentication flow
  - Adapt the sign-in experience to different risk levels (MFA, adaptive authentication, etc…)
- Cognito Identity Pools (for authorization = access control)
  - Obtain AWS credentials for your users
  - Users can login through Public Social, OIDC, SAML & Cognito User Pools
  - Users can be unauthenticated (guests)
  - Users are mapped to IAM roles & policies, can leverage policy variables
- CUP + CIP = authentication + authorization

Cognito Identity Pools – Diagram with CUP



Section 28: Other Serverless: Step Functions & AppSync

Step Functions

- Model workflows in JSON as state machines (1 per workflow)
  - Order fulfillment, data processing, web apps, any workflow
  - Visualization of the workflow, execution, and history

- Start workflow with SDK, API GW, EventBridge

Task States

- Do work on the state machine → invoke AWS service (lambda, run ECS task, insert item to DynamoDB…) or run an activity (EC2, ECS, activities poll step functions for work and send back results to Step function)

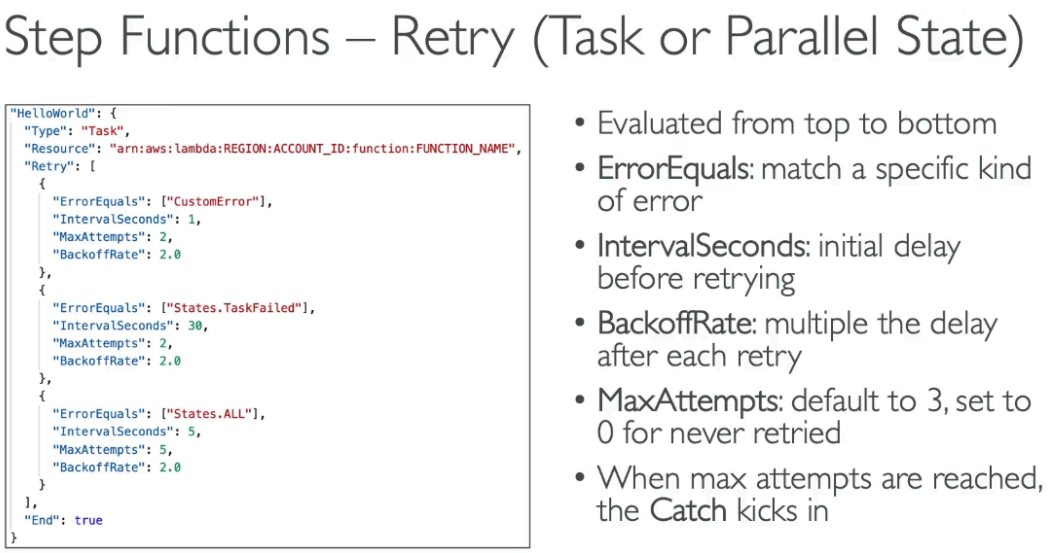
States:

- Choice State: test for a condition to send to a branch
- Fail or Succeed State: stop execution with fail or success
- Pass State: pass input to output without performing work
- Wait state: delay a certain amount of time
- Map State: dynamically iterate steps
- Parallel State: begin parallel branches of execution

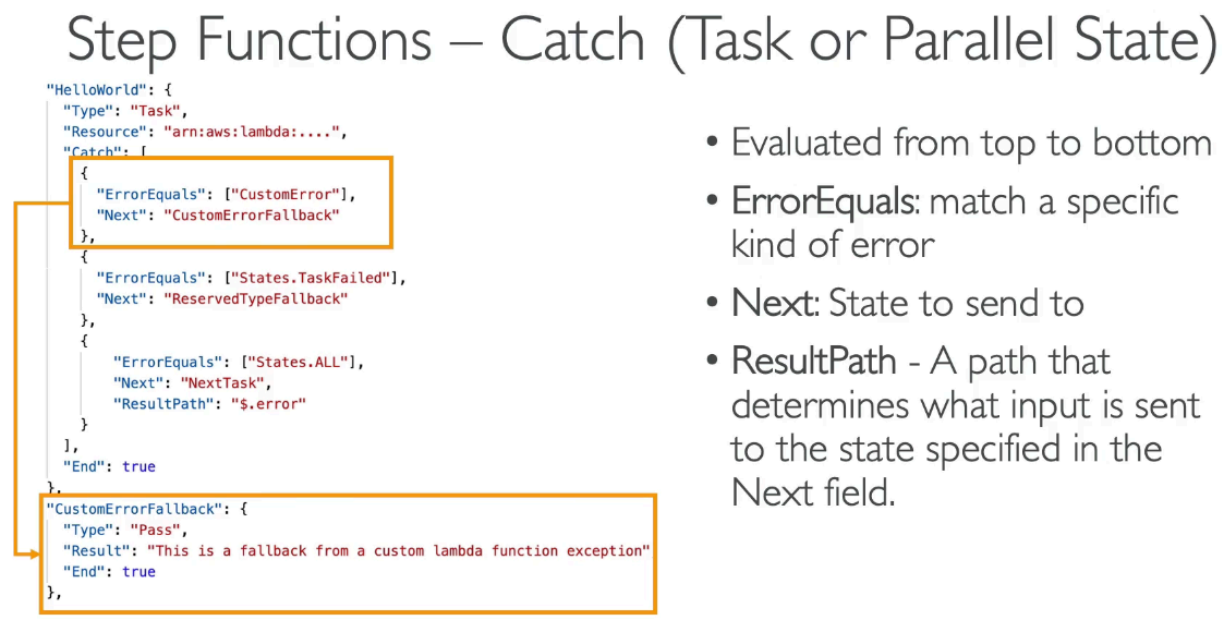
Error Handling

- Retry to retry the failed state or Catch to transition to failure path so State machine handles errors
- Predefined error codes:
    - States.ALL: matches any error name
    - States.Timeout:Task ran longer than timeout or no heartbeat received
    - States.TaskFailed execution failure
    - States.Permissions insufficient permissions to execute code
- State may report its own errors

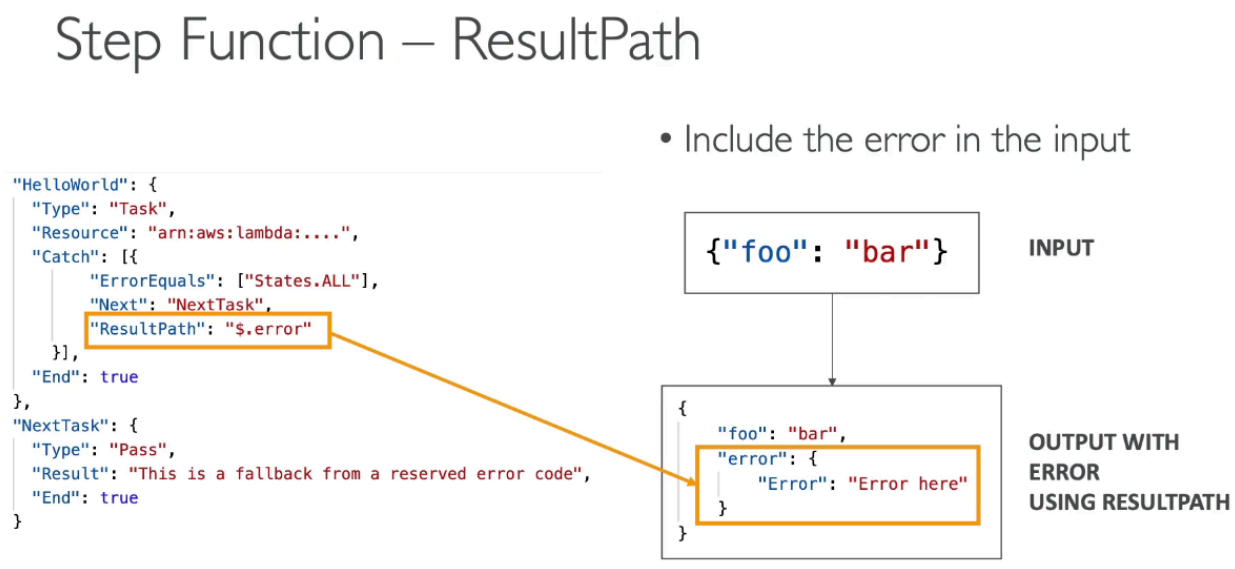
Retry (Task or Parallel State)



Catch (Task or Parallel State)

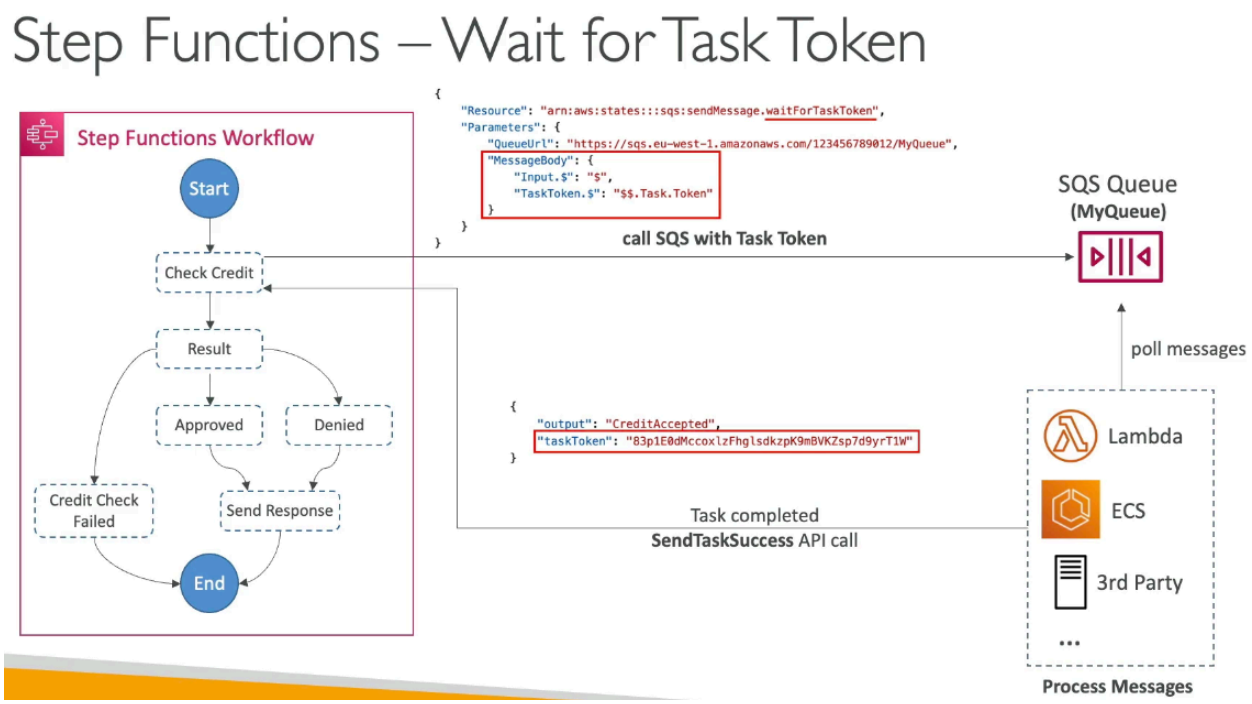


Result Path



- Is how you pass errors from the inputs to outputs into the next task

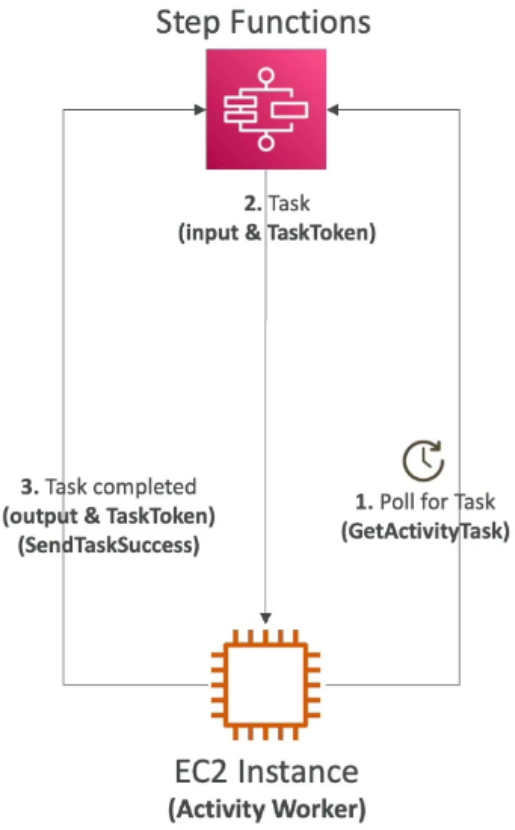
Wait for Task Token



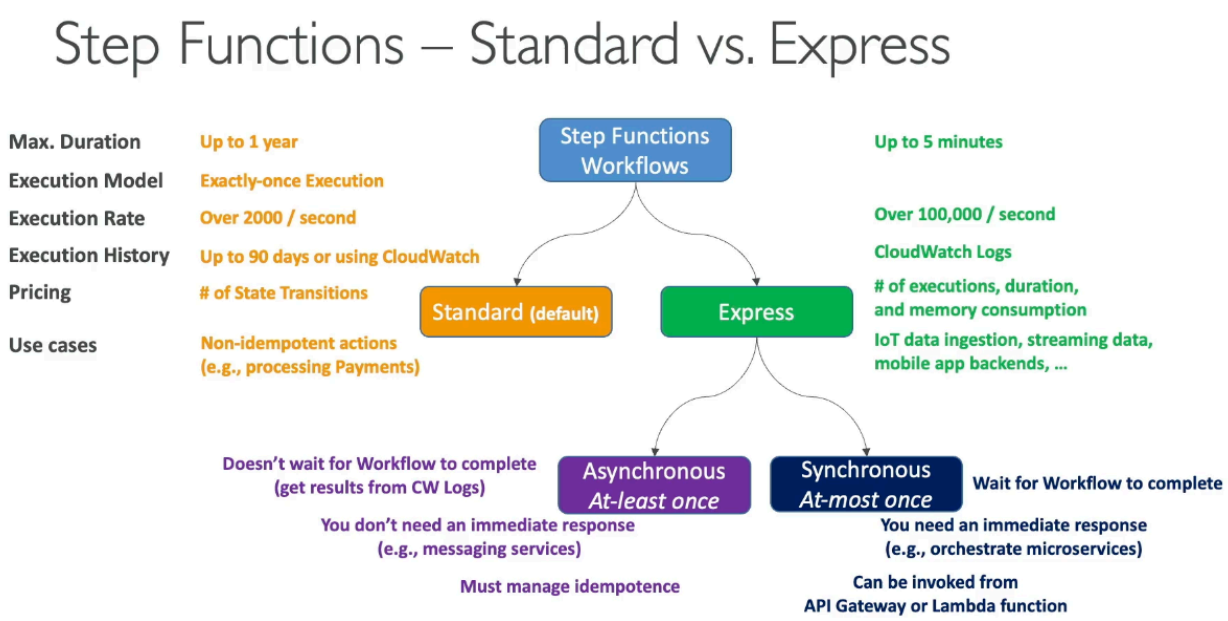
- Allows you to pause step function during a task until Task Token is returned
    - Wait for AWS services, 3rd party call…
    - Append .waitForTaskToken to resource field to tell step functions to wait for task token
    - Task paused until SendTaskSuccess or SendTaskFailure

Activity Tasks

- Enables task work to be performed by Activity Worker
- Activity Worker apps can be running on EC2, Lambda, etc… and poll for a task using GetActivityTask API
    - After worker completes, response of SendTaskSuccess or SendTaskFailure given
- To keep task active:
    - Timeout or heartbeat configurations
    - Configuring long timeout and actively sending heartbeat, activity task can wait up to 1 year



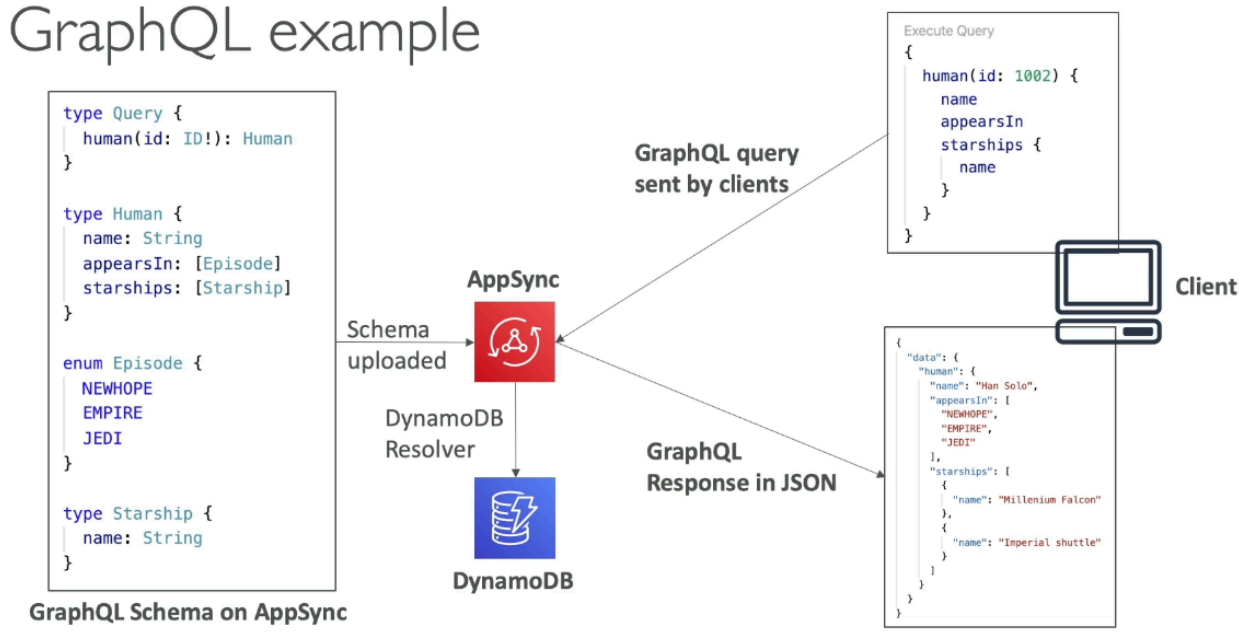
Standard vs Express

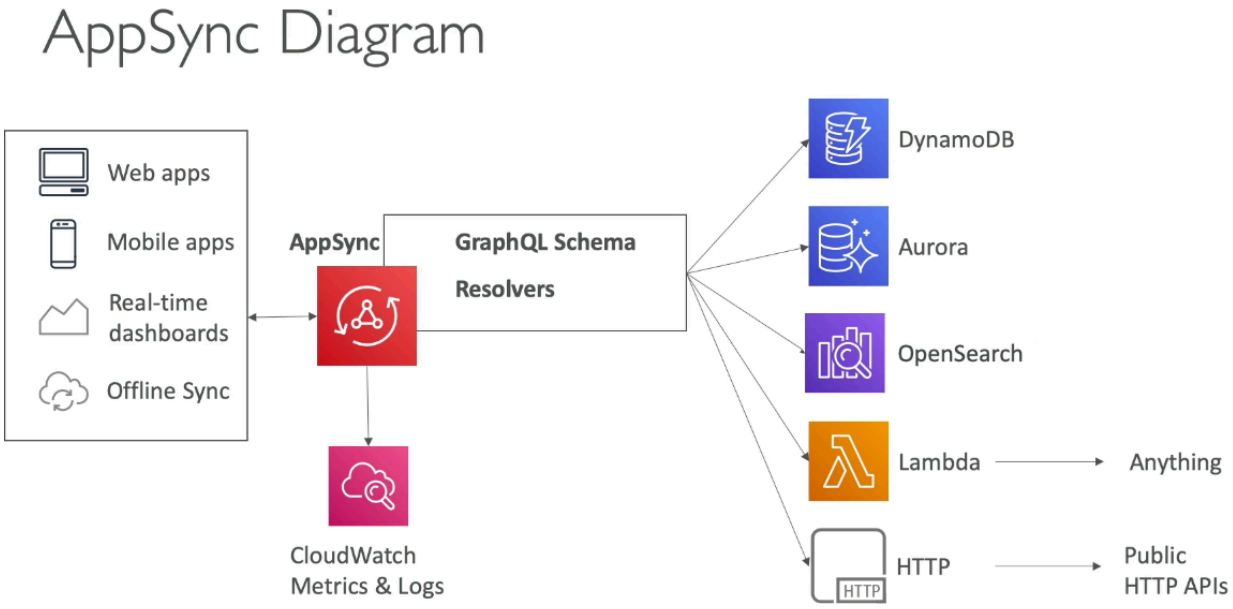


- Exam tests async vs sync where on failures async have at least once execution guarantee where errors will try again directly automatically from step function. Since the same action can be done 2x, must manage idempotence where the effects don't happen 2x. In sync with at most once execution, if it fails the step functions will not restart the workflow and must be set yourself

AppSync

- Managed GraphQL → easy for apps to get exactly the data needed
- Includes combining data from 1+ sources
    - NoSQL, relational DB, HTTP APIs…
    - Direct interaction with DynamoDB, Aurora, custom sources with Lambda
- Retrieve data in real time with WebSocket or MQTT on WebSocket
- For mobile apps: local data access & data sync
- Uploading 1 GraphQL schema





Security

- 4 methods of authorizing apps to interact with AppSync GraphQL API
    1. API KEY
    2. IAM → user / role / cross account
    3. OENID_CONNECT → OpenID or JWT
    4. AMAZON COGNITO USER POOLS
- Custom domain & HTTPS use CloudFront in front of AppSync

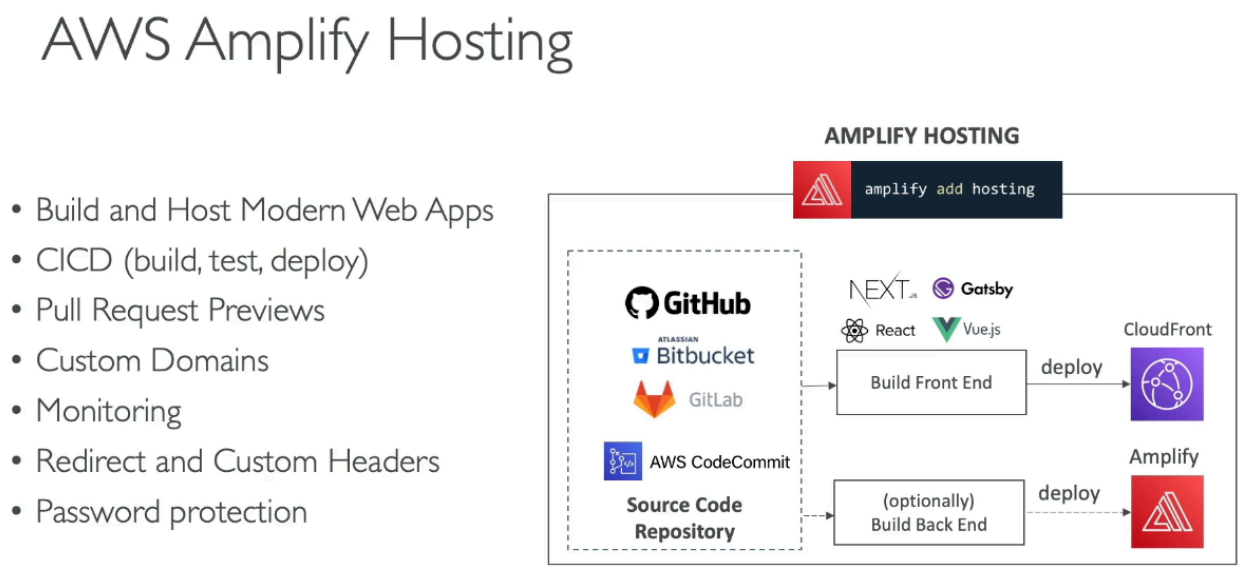
AWS Amplify

- Create mobile and web apps → Elastic beanstalk for mobile and web apps powered by CloudFormation
- Relies on DynamoDB, AppSync, Cognito, S3 as backend with any frontend library

Important Features

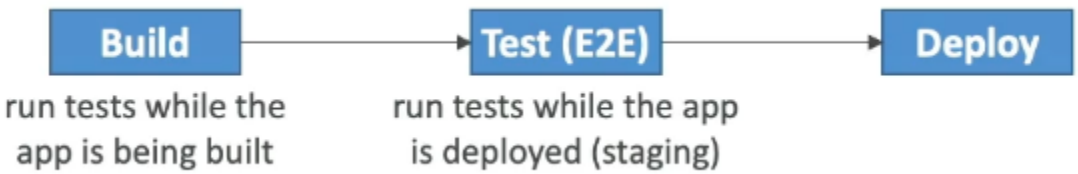
- Authentication (amplify add auth)
    - Uses Cognito for user registration, authentication, recovery with MFA…
    - Prebuilt UI components and fine grained authorization
- Datastore (amplify add api)
    - AppSync + DynamoDB to work with local data and auto sync to cloud
    - Powered with GraphQL with offline and real time capabilities

Amplify Hosting



- Build and host web apps with CICD, monitoring, etc…

E2E Testing



- Run tests in test phase at build time via amplify.yml with integration with Cypress

Section 29: Advanced Identity

STS

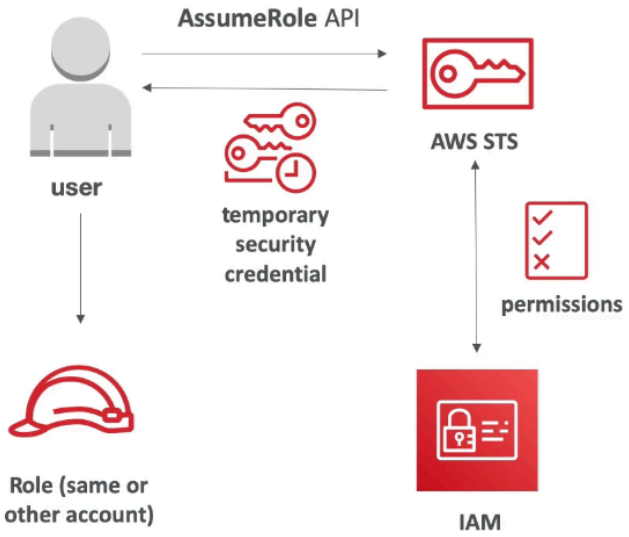
AWS STS – Security Token Service

STS

- Allows to grant limited and temporary access to AWS resources (up to 1 hour).
- AssumeRole: Assume roles within your account or cross account
- AssumeRoleWithSAML: return credentials for users logged with SAML
- AssumeRoleWithWebIdentity
  - return creds for users logged with an IdP (Facebook Login, Google Login, OIDC compatible…)
  - AWS recommends against using this, and using Cognito Identity Pools instead
- GetSessionToken: for MFA, from a user or AWS account root user
- GetFederationToken: obtain temporary creds for a federated user
- GetCallerIdentity: return details about the IAM user or role used in the API call
- DecodeAuthorizationMessage: decode error message when an AWS API is denied

- Allows to grant limited and temporary access to AWS resources (1 hour max)
- AssumeRole, GetSessionToken, GetCallerIdentity, DecodeAuthorizationMessage for exam

STS to Assume Role



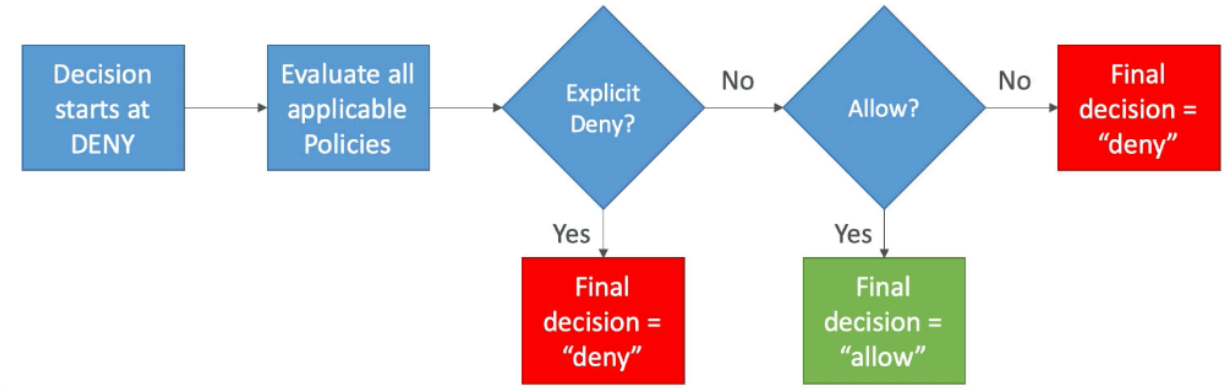
1. Define IAM role within account or cross account
2. Define which principals can access this IAM role
3. STS to retrieve creds and impersonate the IAM role you have access to (AssumeRole)
    a. Temp creds between 15 min to 1 hour

STS with MFA

1. GetSessionToken from STS
    a. Returns Access ID, Secret Key, Session token, expiration date
2. Appropriate IAM policy using IAM conditions
    a. Has aws:MultiFactorAuthPresent = true

Advanced IAM

Authorization Model



1. Explicit DENY, end decision and DENY
2. If ALLOW, end decision with ALLOW
3. Else DENY

IAM Policies & S3 Bucket Policies

- IAM policies are attached to users, roles, groups

- S3 bucket policies are attached to buckets
- Evaluating if IAM principal can perform an operation X on a bucket, the UNION of IAM and S3 is evaluated

Examples

Example 1

- IAM Role attached to EC2 instance, authorizes RW to "my_bucket"
- No S3 Bucket Policy attached

- => EC2 instance can read and write to "my_bucket"

Example 2

- IAM Role attached to EC2 instance, authorizes RW to "my_bucket"
- S3 Bucket Policy attached, explicit deny to the IAM Role

- => EC2 instance cannot read and write to "my_bucket"

Example 3

- IAM Role attached to EC2 instance, no S3 bucket permissions
- S3 Bucket Policy attached, explicit RW allow to the IAM Role

- => EC2 instance can read and write to "my_bucket"

Example 4

- IAM Role attached to EC2 instance, explicit deny S3 bucket permissions
- S3 Bucket Policy attached, explicit RW allow to the IAM Role

- => EC2 instance cannot read and write to "my_bucket"

IAM Dynamic Policies

- How to assign each user a folder in S3 bucket?
  - Option 1:
    - Create IAM policy for each user to have access to each folder → doesn't scale
  - Option 2:
    - Dynamic policies leveraging ${aws:username}

```
{
   "Sid": "AllowAllS3ActionsInUserFolder",
   "Action":["s3:*"],
   "Effect":"Allow",
   "Resource": ["arn:aws:s3:::my-company/home/${aws:username}/*"]
}
```

Inline vs Managed Policies

- AWS Managed Policy
  - Maintained and updated by AWS, good for power users and amin
- Customer Managed Policy
  - Best practice, re-usable, version control + rollback
- Inline
  - Strict 1-1 relationship between policy and principal → only for IAM USERS
  - Not reusable, not version controlled, no rollback, policy deleted if IAM principal deleted

Granting User Permissions to pass a role to AWS Service

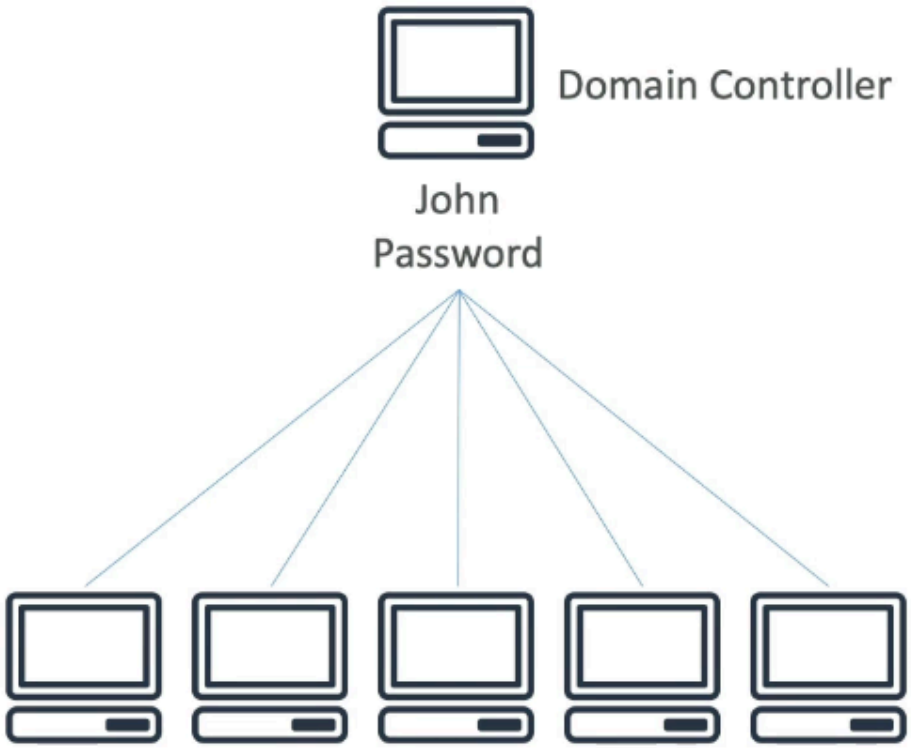
- Must pass IAM role to service to assume role and perform actions on your behalf
  - Need iam:PassRole and iam:GetRole to view role being passed

Can a role be passed to any service?

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Sid": "TrustPolicyStatementThatAllowsEC2ServiceToAssumeTheAttachedRole",
        "Effect": "Allow",
        "Principal": {
            "Service": "ec2.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
}
```

- No, roles can only be passed to what their trust allows.
- A trust policy for the role that allows the service to assume the role

AWS Directory Services

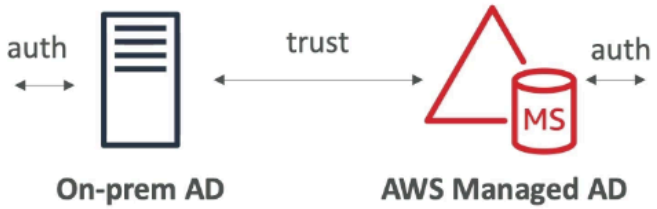


Microsoft Active Directory

- DB of objects: user accounts, computers, printers, etc… with centralized security management
    - Objects are organized in trees and a group of trees is a forest

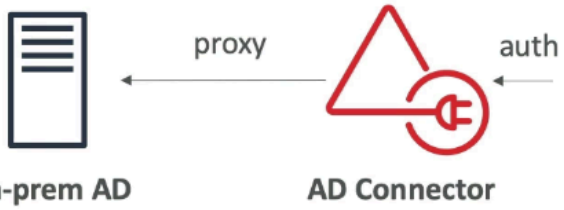
AWS Managed Microsoft AD

- Create own AD in AWS, manage users locally, supports MFA
- Establish trust connections with on premise AD



AD Connector

- Directory Gateway (proxy) to redirect to on premise AD, supports MFA
  - Users are managed on on premise AD



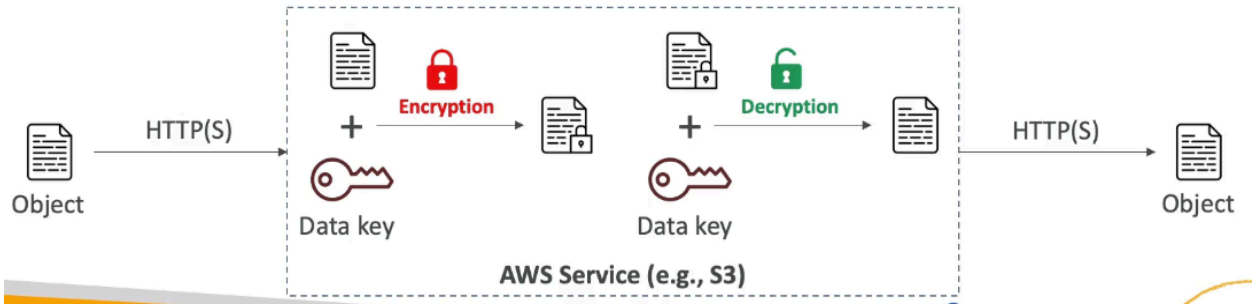
Simple AD

- AD compatible managed directory on AWS, cannot be joined with on premise AD

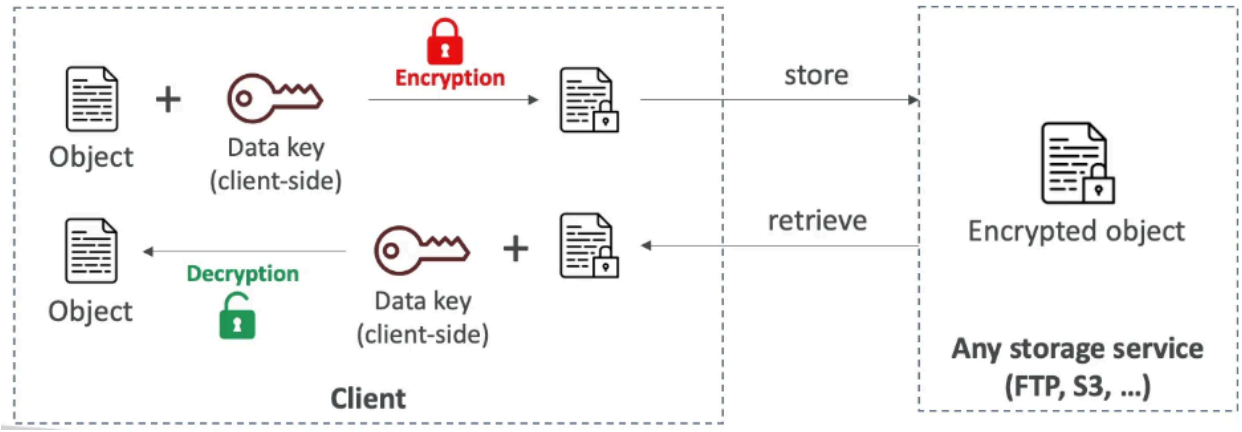
Section 30: AWS Security & Encryption

Encryption 101

- Encryption in flight (TLS / SSL for HTTPS encryption)
  - Data encrypted before sending and decrypted after receiving → only target server can receive it; ensures no middle man attack occurs



- Server Side encryption at rest
  - Data is encrypted after received by the server and decrypted before being sent
  - Stored in encrypted form via (data) key where the encryption / decryption keys must be managed somewhere and the server needs access to it



- Client side encryption
    - Data is encrypted by the client and never decrypted by the server (server not trusted and cannot decrypt data)
    - Data will be decrypted by a receiving client
    - Could leverage envelope encryption

KMS

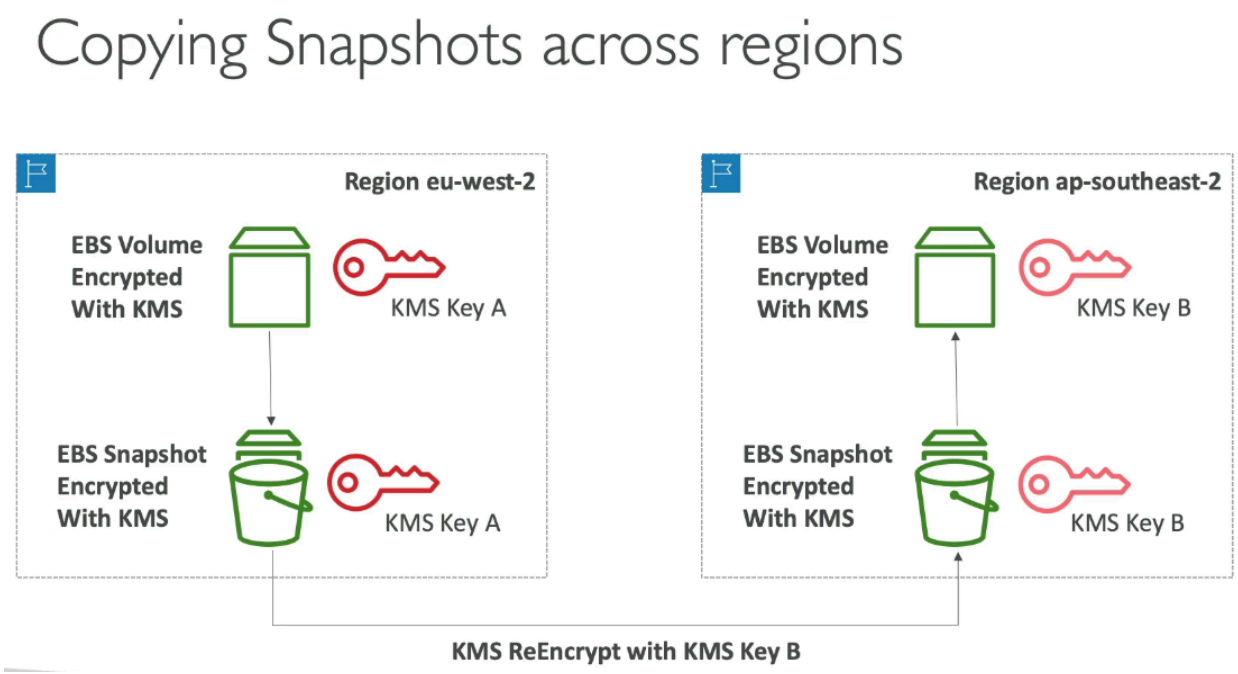
- AWS managed encryption keys, fully integrated with IAM for authorization and easy ways to control access to data; can audit with CloudTrail and integration with AWS services
    - Scoped per region, the same KMS key cannot be in the same region
- KMS key types:
    - AWS owned → default free key for SSE-S3, SSE-SQS, SSE-DDB
    - AWS managed key → free for aws/service name
    - Customer managed key created in KMS → $1 /month
    - Customer managed imported → $1 /month
    - + pay for API call to KMS
- Automatic Key rotation
    - AWS managed: auto rotate every 1 year
    - Customer managed KMS: enabled feature for automatic rotation 1 year
    - Imported: only manual rotation using alias

Key Types

- Symmetric (AES-256)
    - Single encryption key for encryption / decryption that all AWS services integrate with
    - Never get access to KMS key unencrypted
- Asymmetric (RSA & ECC Key Pairs)
    - Public (encrypt) and private (decrypt) key pair for sign / verify

    - Public key is downloadable, but can't access private key unencrypted
- Used for encryption outside of AWS by users who can't use KMS

Copying Snapshots across Regions



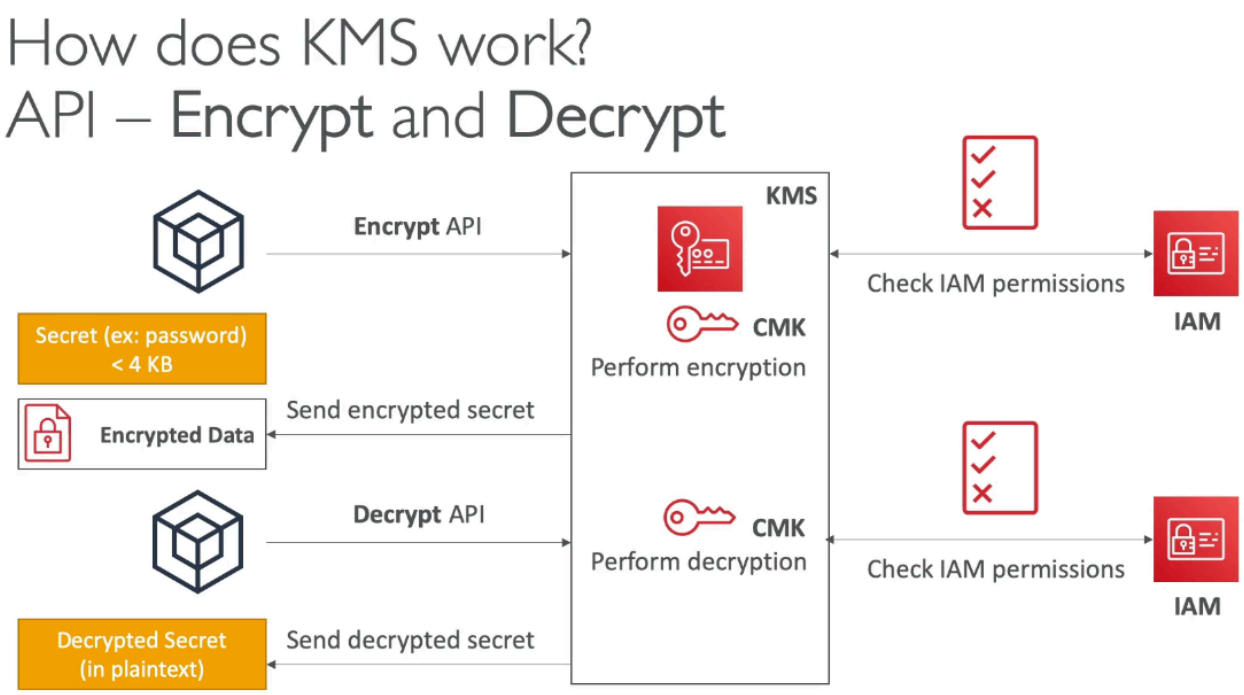
Key Policies

- Control access to KMS keys similar to S3 bucket policy
    - Difference: cannot control access without them
        - If there is no policy, no one can access the KMS key
- Default KMS policy:
    - Created if you don't provide a key policy
    - Complete access to the key to root user = entire AWS account
- Custom Key Policy:
    - Define users, roles access to key and define who can administer the key
    - Useful for cross account access of KMS key

Copying Snapshots across Accounts

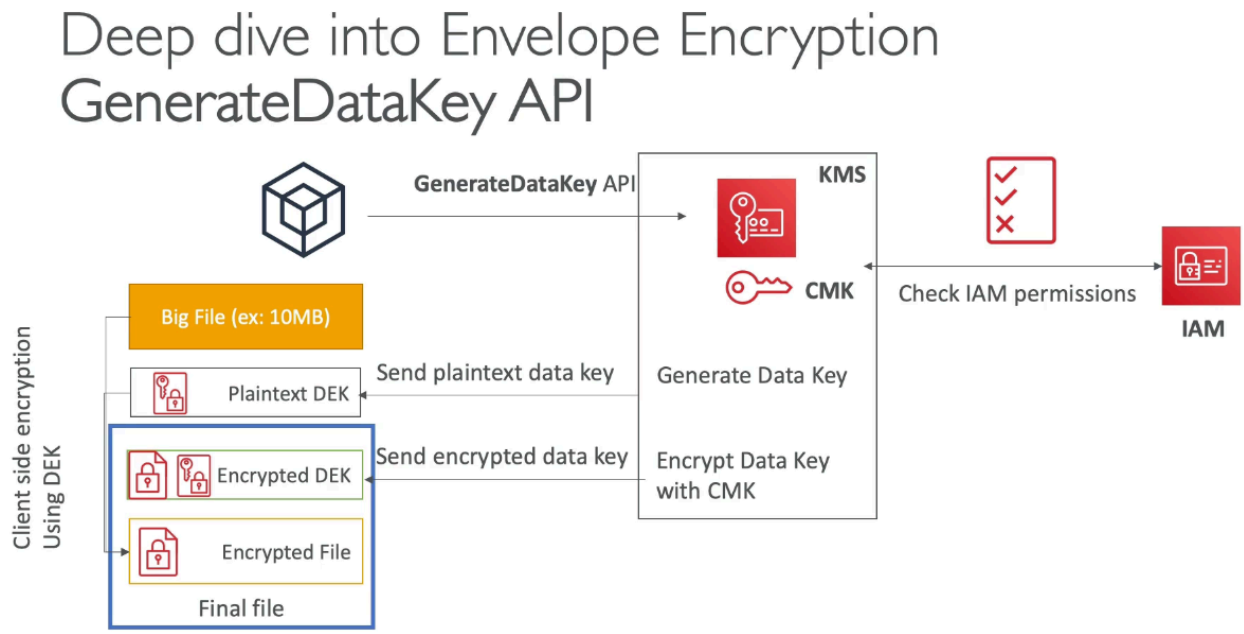
1. Create snapshot, encrypted with own KMS key (customer managed)
2. Attach KMS key policy to authorize cross account access
3. Share encrypted snapshot
4. (in target) create a copy of the snapshot, encrypt it with a different customer managed key in account
5. Create a volume from snapshot

KMS Encryption Patterns and Envelope Encryption



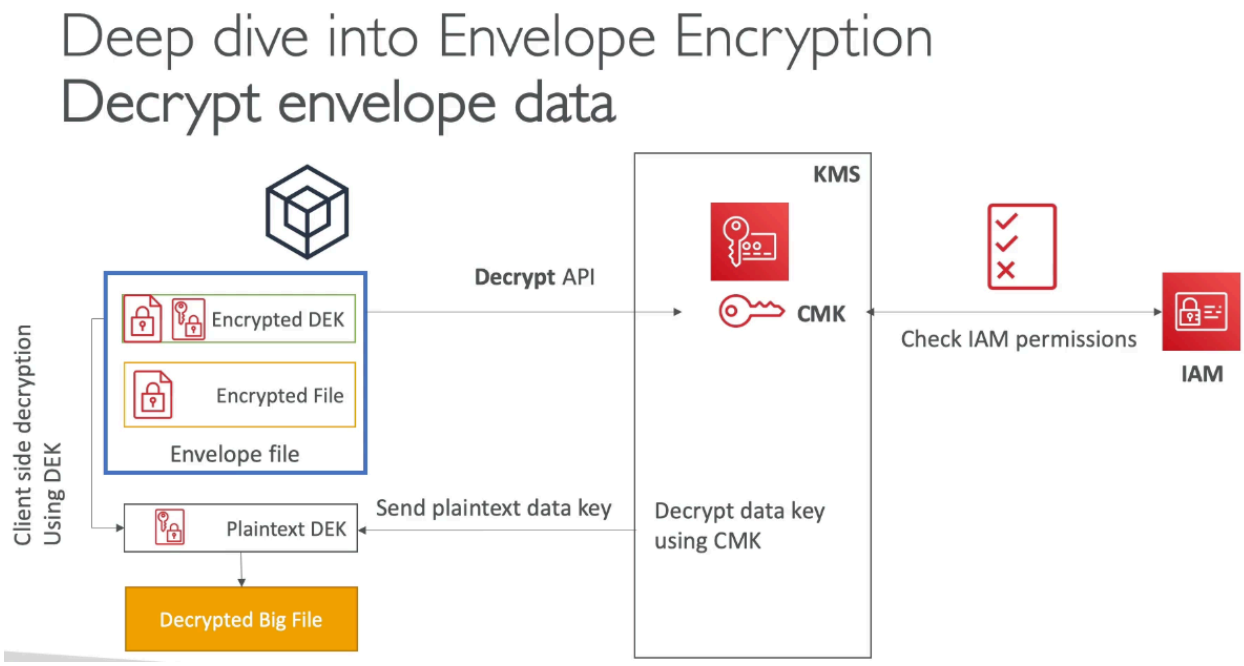
- Size of the secret is limited to 4 KB

Envelope Encryption



- To encrypt > 4 KB using GenerateDataKey API

- API call sends plain text & encrypted data key for client side to encrypt the file, but wraps encrypted file + received encrypted DEK together into 1 file, thus envelope



- To decrypt, call DecryptAPI, which only has max 4 KB data sent. This API sends plain text data key for client side decryption

Encryption SDK

- Implements envelope encryption for programming languages + CLI
- Feature → data key caching
  - Reuse data keys instead of creating new ones for each encryption to help reduce KMS calls with security trade off → basically all encryption is done with the same key
  - Use LocalCryptoMaterialsCache (max age, max bytes, max # of messages) to set cache limits

KMS Symmetric – API Summary

- Encrypt up to 4 KB of data through KMS
- GenerateDataKey: generates unique symmetric data key (DEK)
  - Returns plaintext and encrypted version (under customer managed key you specify) of the data key
- GenerateDataKeyWithoutPlaintext → generate DEK at some point (not immediate)
  - DEK encrypted under CMK you specify (must decrypt later)
- Decrypt: decrypt up to 4 KB data (including data encryption keys)

- GenerateRandom: returns a random byte string

KMS Limits

KMS Request Quotas

API operation	Request quotas (per second)

Decrypt
Encrypt
GenerateDataKey (symmetric)
GenerateDataKeyWithoutPlaintext (symmetric)
GenerateRandom
ReEncrypt
Sign (asymmetric)
Verify (asymmetric)
	These shared quotas vary with the AWS Region and the type of CMK used in the request. Each quota is calculated separately.
	
	Symmetric CMK quota:
	• 5,500 (shared)
	• 10,000 (shared) in the following Regions:
	  • us-east-2, ap-southeast-1, ap-southeast-2, ap-northeast-1, eu-central-1, eu-west-2
	• 30,000 (shared) in the following Regions:
	  • us-east-1, us-west-2, eu-west-1
	
	Asymmetric CMK quota:
	• 500 (shared) for RSA CMKs
	• 300 (shared) for Elliptic curve (ECC) CMKs

- ThrottlingException if exceeds request quota
    - Use exponential backoff or request quota via Support
- For each cryptographic operations, they share a quota → this includes requests made by AWS services on your behalf (ex: SSE-KMS)
    - For GenerateDataKey API, consider using DEK caching to reduce calls

S3 Bucket Key

- New setting for SSE-KMS encryption to reduce S3 calls to KMS by 99%
- Leverages data keys
    - S3 bucket key is generated from KMS, which the bucket key can create many data keys via envelope encryption to encrypt S3 buckets → S3 bucket key doesn't count as KMS call
- See less KMS CloudTrail events

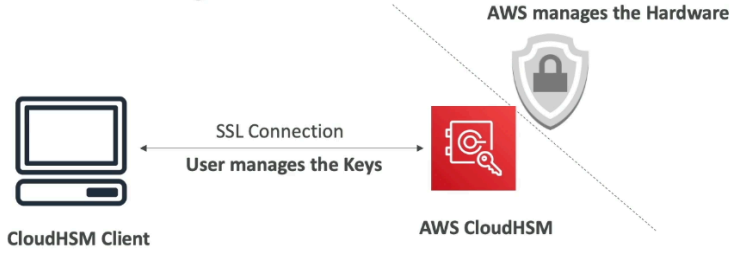
KMS Key Policies & IAM Principals

- Principal options in IAM policies
    - Account root user, IAM roles / role sessions / users., federated user sessions, AWS services, all principals via *

CloudHSM

- In KMS, AWS manages software for encryption, but CloudHSM (hardware security module) AWS provisions encryption hardware → dedicated hardware to manage own encryption keys
    - HSM device is tamper resistant FIPS 140-2 Level 3 compliance
        - KMS is multi-tenant while CloudHSM is single tenant meaning hardware used is only for you, not shared with others
    - Supports both symmetric and asymmetric encryption (SSL / TLS)
    - No free tier and must use CloudHSM client software
- Redshift supports CloudHSM for DB encryption and key management, good option to use with SSE-C encryption
- CloudHSM clusters are Multi AZ for high availability and durability

CloudHSM Diagram

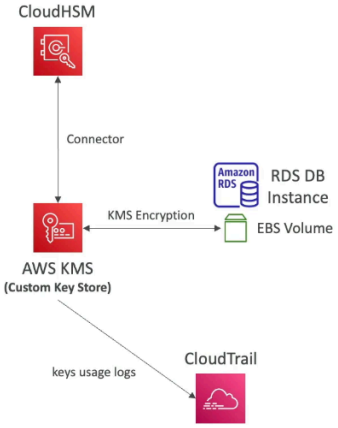


IAM permissions:
- CRUD an HSM Cluster

CloudHSM Software:
- Manage the Keys
- Manage the Users

CloudHSM – Integration with AWS Services

- Through integration with AWS KMS
- Configure KMS Custom Key Store with CloudHSM
- Example: EBS, S3, RDS …

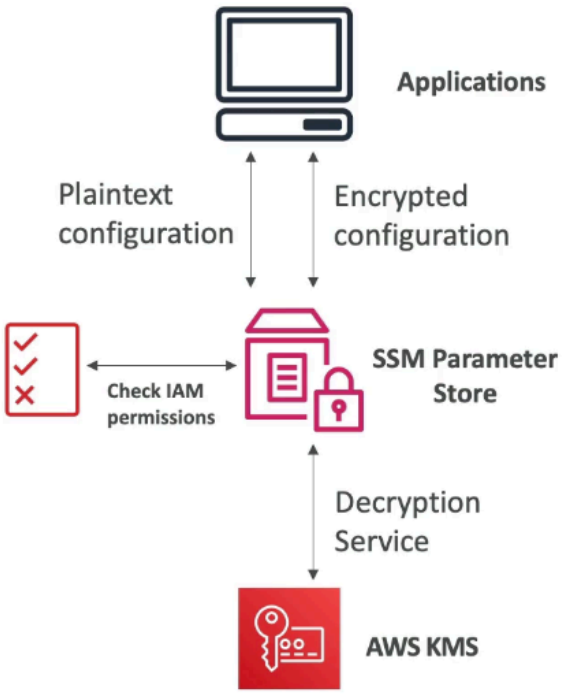


CloudHSM vs. KMS

Feature	AWS KMS	AWS CloudHSM
Tenancy	Multi-Tenant	Single-Tenant
Standard	FIPS 140-2 Level 3	FIPS 140-2 Level 3
Master Keys	• AWS Owned CMK • AWS Managed CMK • Customer Managed CMK	Customer Managed CMK
Key Types	• Symmetric • Asymmetric • Digital Signing	• Symmetric • Asymmetric • Digital Signing & Hashing
Key Accessibility	Accessible in multiple AWS regions (can't access keys outside the region it's created in)	• Deployed and managed in a VPC • Can be shared across VPCs (VPC Peering)
Cryptographic Acceleration	None	• SSL/TLS Acceleration • Oracle TDE Acceleration
Access & Authentication	AWS IAM	You create users and manage their permissions

SSM Parameter Store

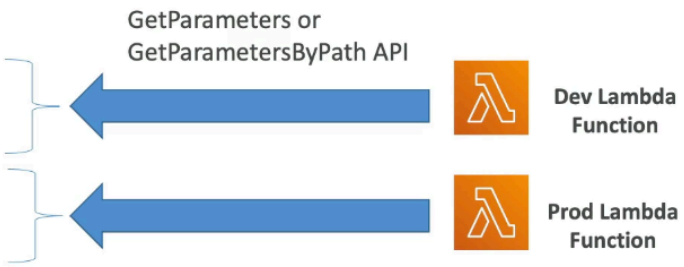
- Secure storage for configuration and secrets with optional encryption with KMS
- Serverless, scalable, durable, easy SDK with version tracking of configurations / secrets
- Security via IAM and EventBridge notifications; integration with CloudFormation



Parameter Store Hierarchy

SSM Parameter Store Hierarchy

- /my-department/
  - my-app/
    - dev/
      - db-url
      - db-password
    - prod/
      - db-url
      - db-password
  - other-app/
- /other-department/
- /aws/reference/secretsmanager/secret_ID_in_Secrets_Manager
- /aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2 (public)



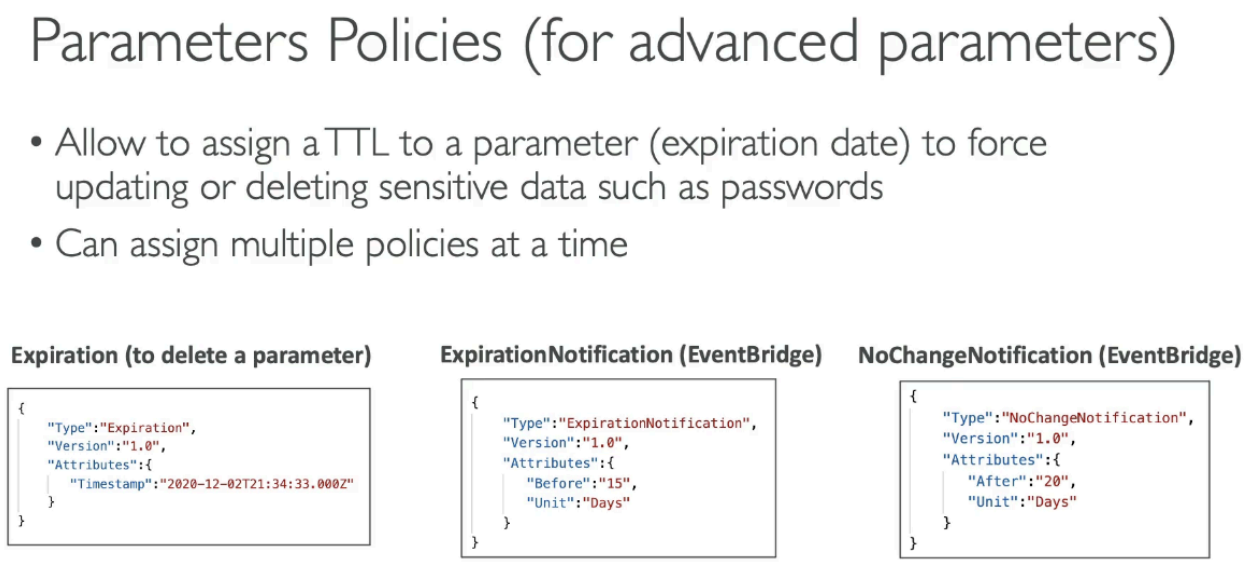
- Can have folders for how far down passwords are and can get secrets manager values

Standard and advanced parameter tiers

	Standard	Advanced
Total number of parameters allowed (per AWS account and Region)	10,000	100,000
Maximum size of a parameter value	4 KB	8 KB
Parameter policies available	No	Yes
Cost	No additional charge	Charges apply
Storage Pricing	Free	$0.05 per advanced parameter per month

- Biggest difference between standard and advanced parameter tiers is parameter policies

Parameters Policies (advanced parameters only)



- Allow to assign TTL to a parameter to force update or delete sensitive data
- Can assign multiple policies at a time

Secrets Manager

- Capability to force rotation of secrets every X days and automate generation of secrets on rotation (uses lambda)
    - Can encrypt with RDS
- Integration with RDS
    - Mostly meant for RDS

Multi Region Secrets

- Replicate secrets across multiple regions and secrets manager keeps read replicas in sync with primary secret
    - Can promote read replica secret to standalone secret
    - For: multi region apps, disaster recovery, multi region DB…

SSM Parameter Store vs Secrets Manager

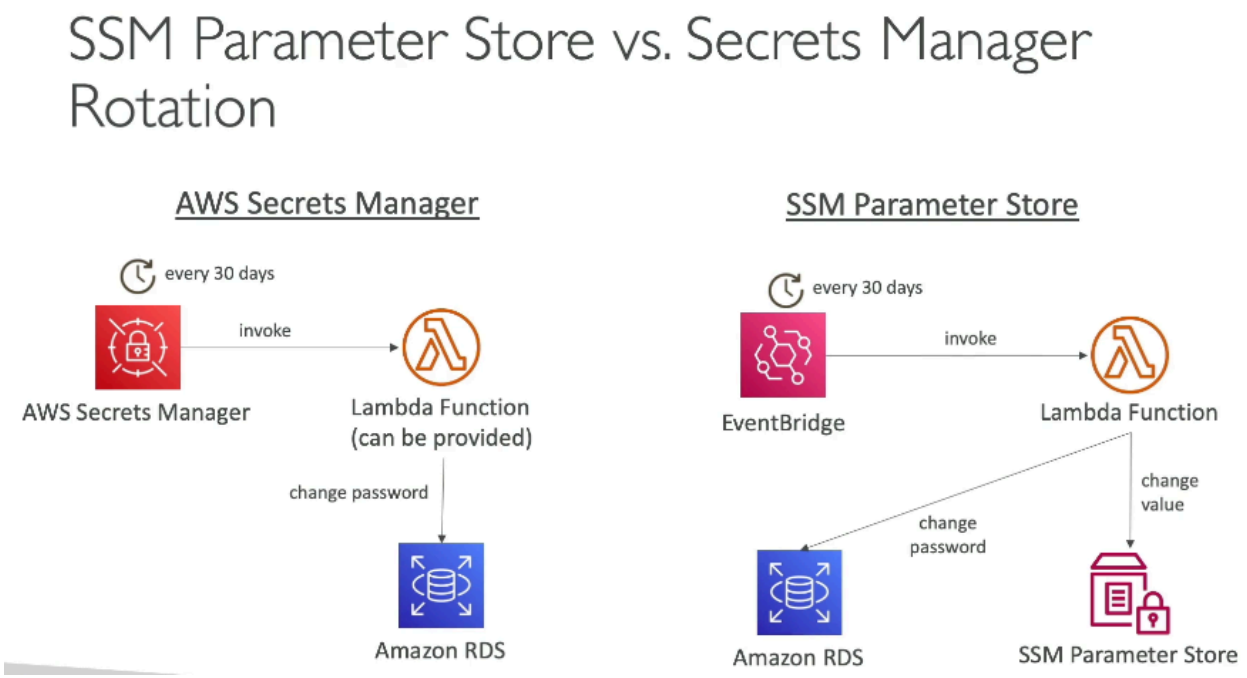
Secrets Manager ($$$)

- Auto rotation of secrets with Lambda (lambda provided for RDS, Redshift, DocumentDB)
- KMS encryption is mandatory and has CloudFormation integration

SSM Parameter Store ($)

- Simple API with no secret rotation (can enable using lambda triggered by EventBridge)
- KMS encryption optional, has integration with CF
- Can pull Secrets Manager secret using API

Rotation difference

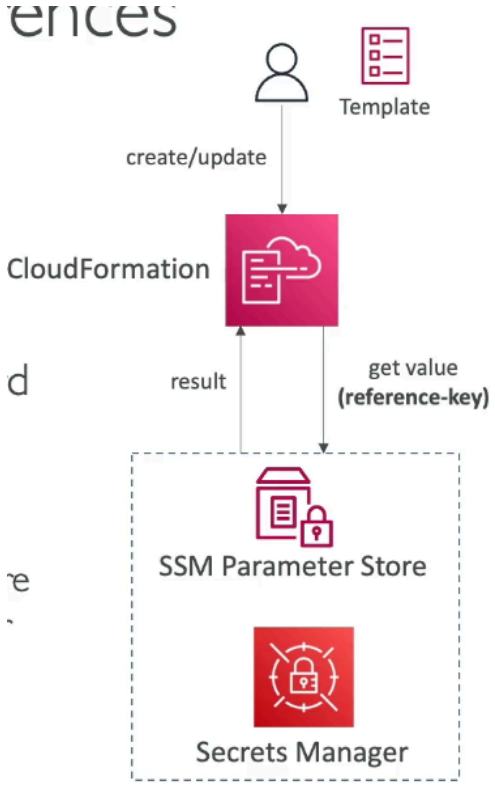


CloudFormation – Secrets Manager & SSM Integration

Dynamic References

- Reference external values stored in Parameter Store or Secrets Manager within CF templates
  - CF retrieves value of specified reference during create / update / delete operations
- Supports: ssm for plaintext values in parameter store, ssm-secure for secure strings stored in SSM parameter store, and secretsmanager for secret values stored in secrets manager

'{{resolve:*service-name*:*reference-key*}}'



CloudFormation – Dynamic References

SSM

{{resolve:**ssm**:*parameter-name*:*version*}}

```
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      AccessControl: '{{resolve:ssm:S3AccessControl:2}}'
```

SSM Secure

{{resolve:**ssm-secure**:*parameter-name*:*version*}}

```
Resources:
  IAMUser:
    Type: AWS::IAM::User
    Properties:
      UserName: john
      LoginProfile:
        Password: '{{resolve:ssm-secure:IAMUserPassword:10}}'
```

Secrets Manager

{{resolve:**secretsmanager**:*secret-id*:*secret-string*:*json-key*:*version-stage*:*version-id*}}

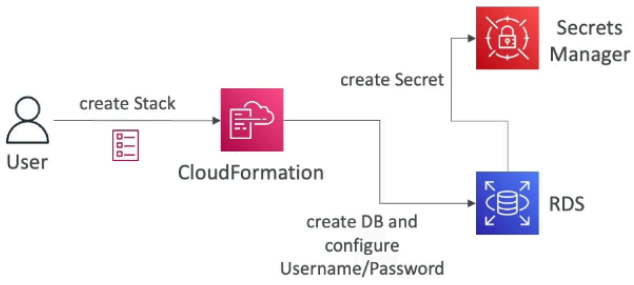
```
Resources:
  DBInstance:
    Type: AWS::RDS::DBInstance
    Properties:
      DBName: MyRDSInstance
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword: '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

CloudFormation, Secrets Manager & RDS

Option 1 – ManagerMasterUserPassword

CloudFormation, Secrets Manager & RDS Option 1 – ManageMasterUserPassword

- ManageMasterUserPassword – creates admin secret implicitly
- RDS, Aurora will manage the secret in Secrets Manager and its rotation

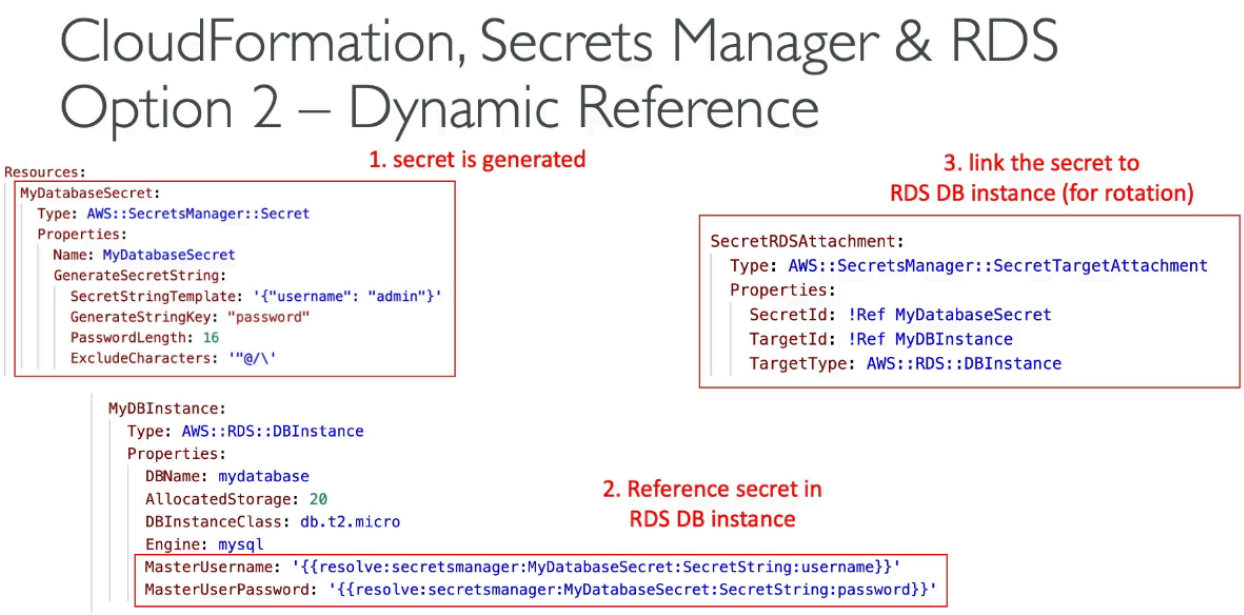



```
Resources:
  MyCluster:
    Type: AWS::RDS::DBCluster
    Properties:
      Engine: aurora-mysql
      MasterUsername: masteruser
      ManageMasterUserPassword: true

Outputs:
  Secret:
    Value: !GetAtt MyCluster.MasterUserSecret.SecretArn
```

- Secrets manager will create admin secret implicitly and RDS / Aurora will manage the secret in Secrets Manager

Option 2 – Dynamic Reference



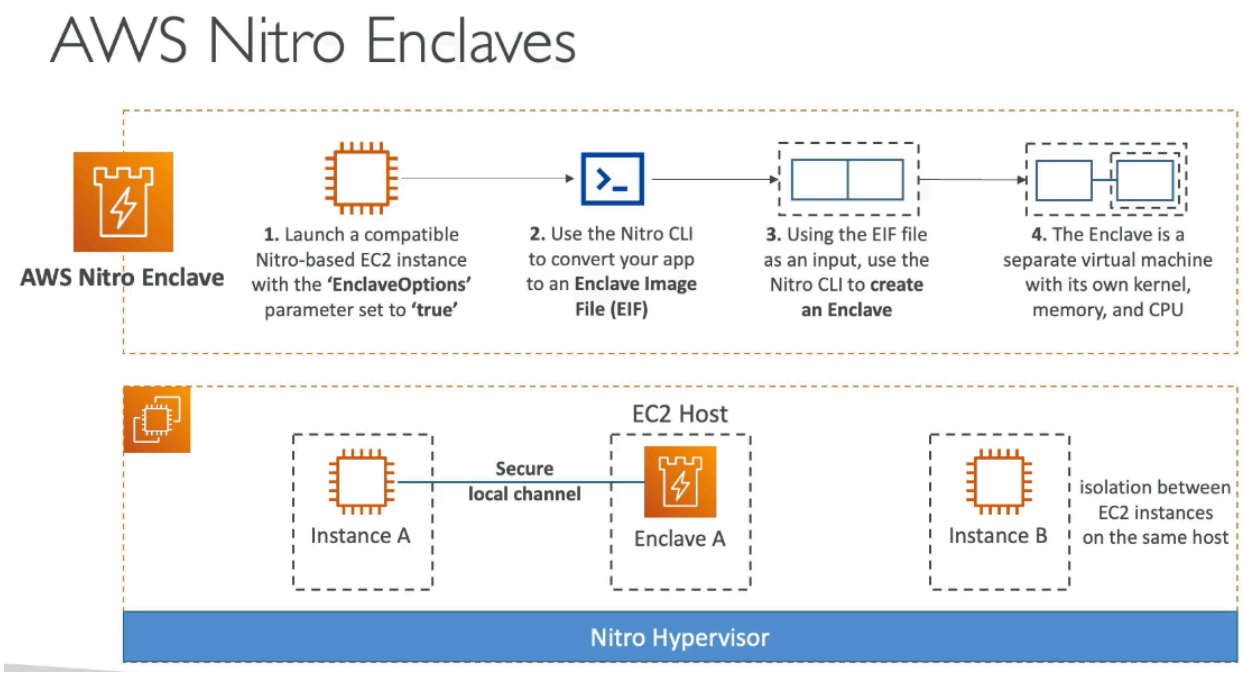
CloudWatch Logs Encryption

- Can encrypt with KMS and occurs at log group level by associating a CMK with a log group during or after creation
    - Cannot associate a CMK with a log group within console
- API calls:
    - associate-kms-key → if the log group already exists
    - create-log-group → if the log group doesn't exist yet

CodeBuild Security

- To access resources in VPC, make sure to specify a VPC configuration for CodeBuild
- Secrets in CodeBuild should be environment variables referenced as parameter store or secret manager values

AWS Nitro Enclaves



- Process highly sensitive data in isolated compute environment
- Fully isolated VMs, hardened and highly constrained
    - Not a container, persistent storage, interactive access, external networking
    - Helps reduce attack surface for sensitive data processing apps
        - Cryptographic Attestation → only authorized code can be running on Enclave
        - Only Enclaves can access sensitive data (KMS integration)

Section 31: AWS Other Services

AWS Simple Email Service (SES)

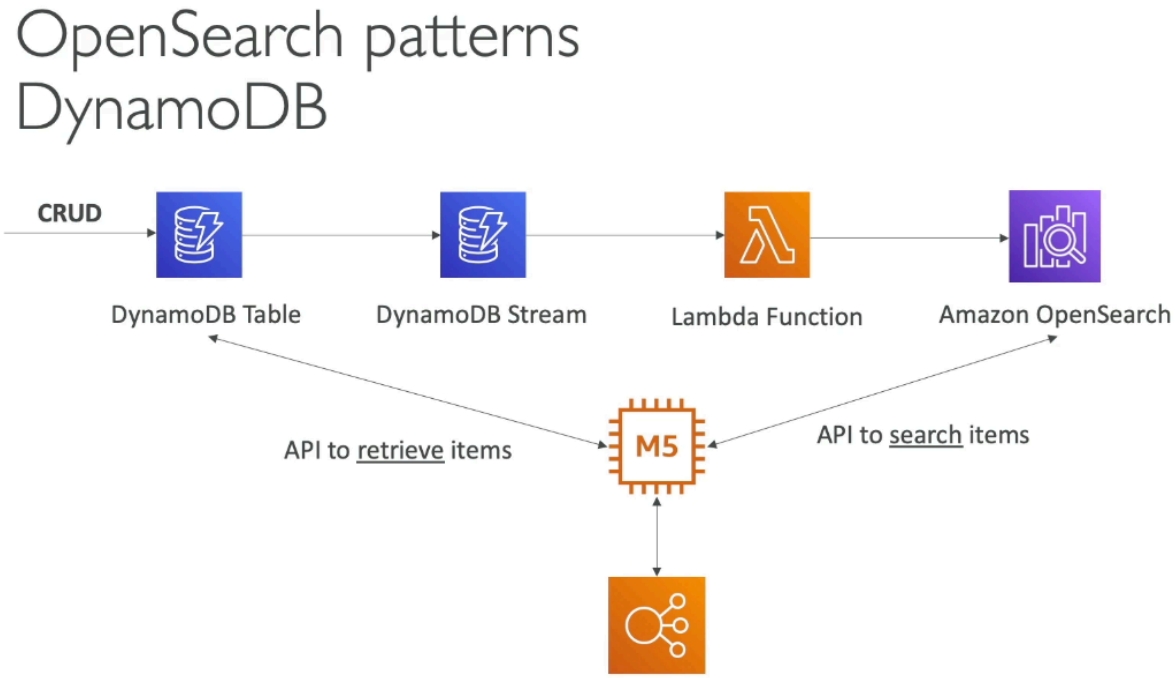
- Send emails and ability receive email
    - Integrates with S3, SNS, Lambda and IAM for allowing to send emails

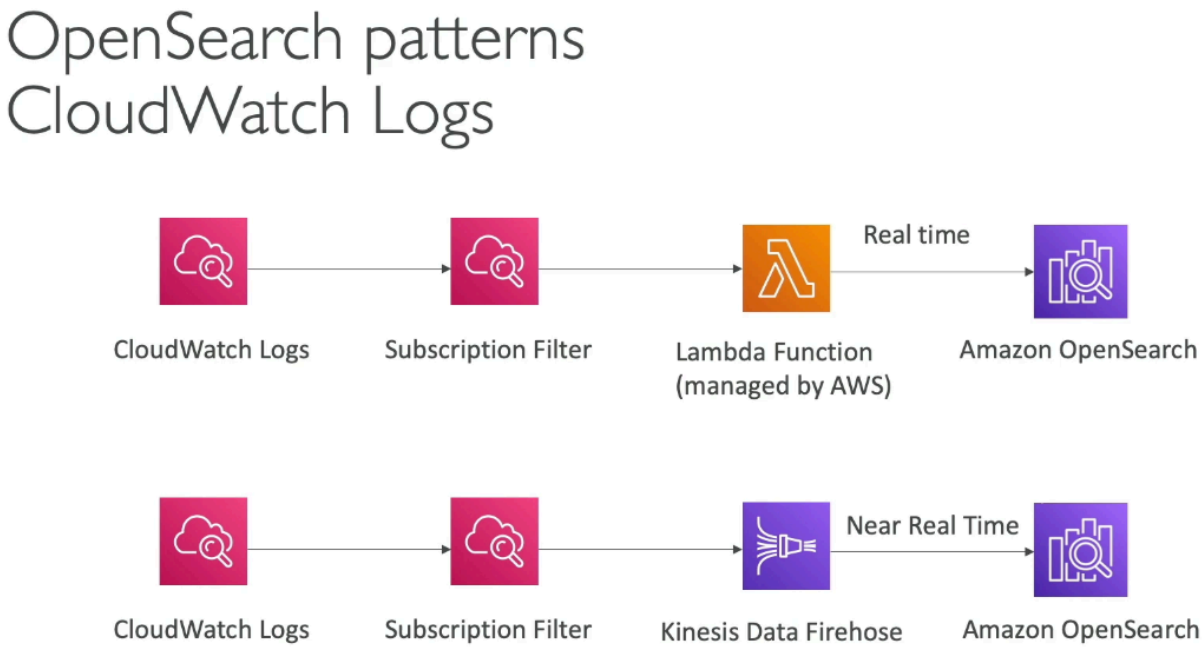
Amazon OpenSearch

- In DynamoDB, queries only exist by primary key or indexes, but with OpenSearch you can search any field, even partial matches and comes with OpenSearch Dashboards for visualization
    - Common to use OpenSearch as a complement to another DB
    - Ingestion of data from Kinesis Firehose, AWS IoT, CloudWatch logs
- 2 modes: managed cluster or serverless
- No native support for SQL

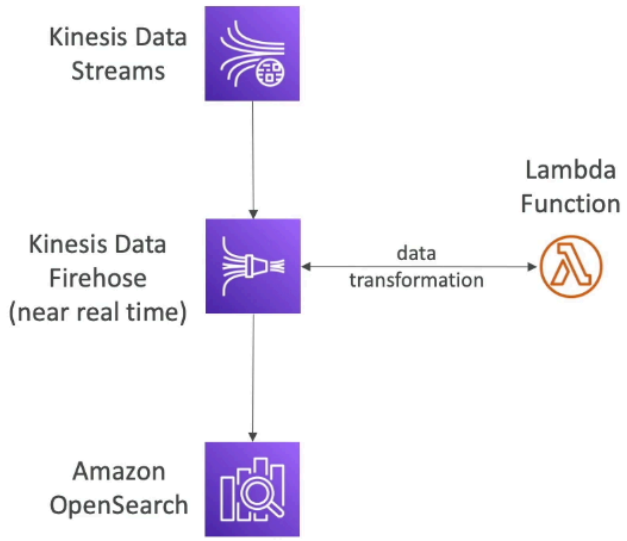
- Security through Cognito, IAM, KMS, TLS

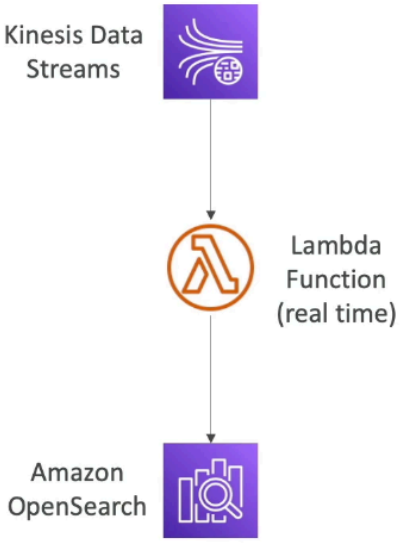
Architecture Patterns





OpenSearch patterns
Kinesis Data Streams & Kinesis Data Firehose





Amazon Athena

- Serverless query service to analyze data stored in S3 via SQL to query files (business analytics, reporting, etc…)
  - Supports CSV, JSON…
  - Pricing: $5 per TB of data scanned
- Commonly used with QuickSight for reporting / dashboards
- Analyze data in S3 using serverless SQL, use Athena

Performance Improvement

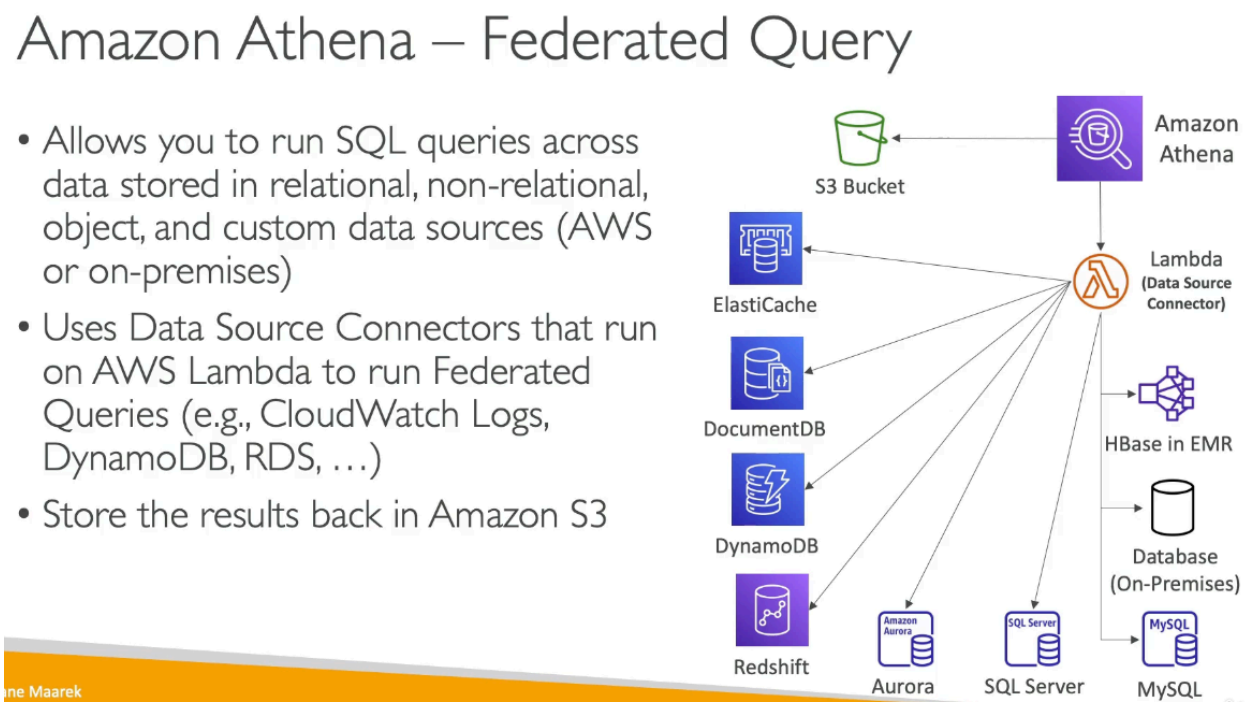
- Columnar data for cost savings (less scan) to only scan the columns you need
  - Apache Parquet or OCR is recommended
  - Huge performance improvement
  - Use Glue to convert data to Parquet or ORC
- Compress data for smaller retrievals
- Partition datasets in S3 for easy querying on virtual columns
  - s3://bucket//pathToTable for direct access

s3://yourBucket/pathToTable
/<PARTITION_COLUMN_NAME>=<VALUE>
/<PARTITION_COLUMN_NAME>=<VALUE>
/<PARTITION_COLUMN_NAME>=<VALUE>
/etc…

Example: s3://athena-examples/flight/parquet/year=1991/month=1/day=1/

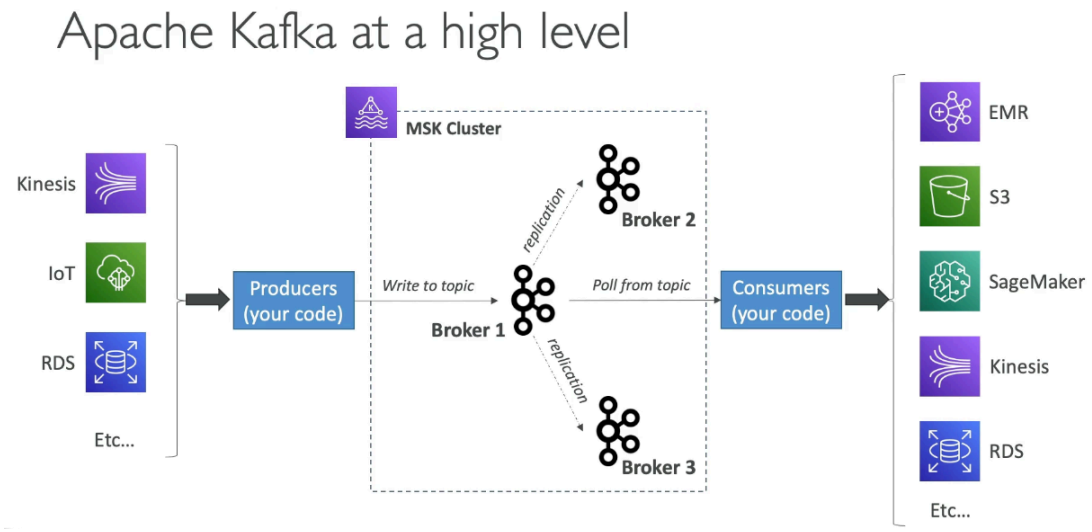
- Use larger files (> 128 MB) to minimize overhead

Federated Query



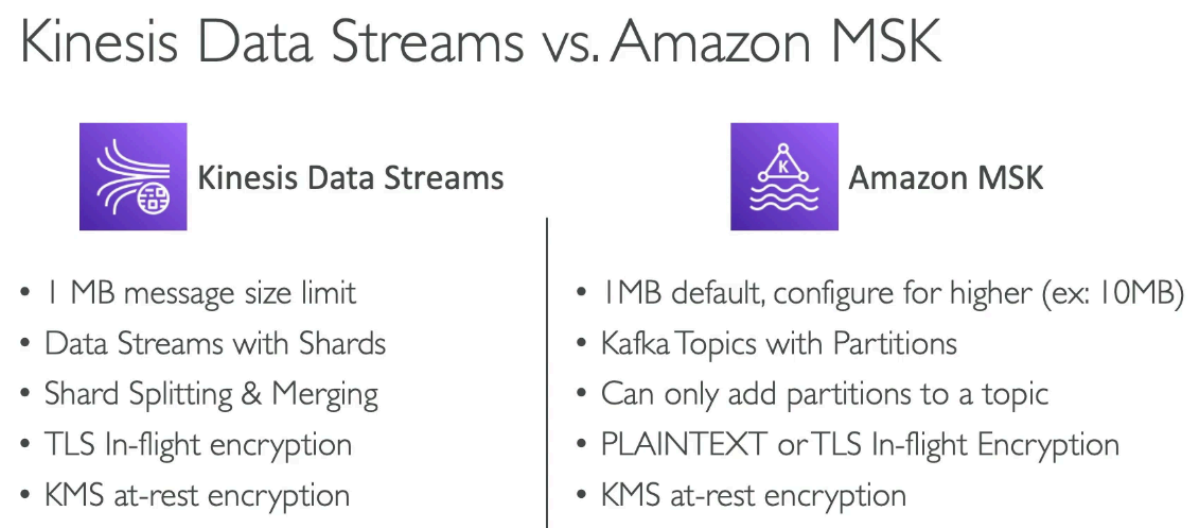
- Allows to run SQL queries across data stored in relational, non relational, object, and custom data sources (AWS or on premise)
- Uses data source connectors that run on Lambda to run Federated Queries in other services (CW logs, DynamoDB, RDS…) and stores results back to S3

Amazon MSK (Managed Streaming for Apache Kafka)

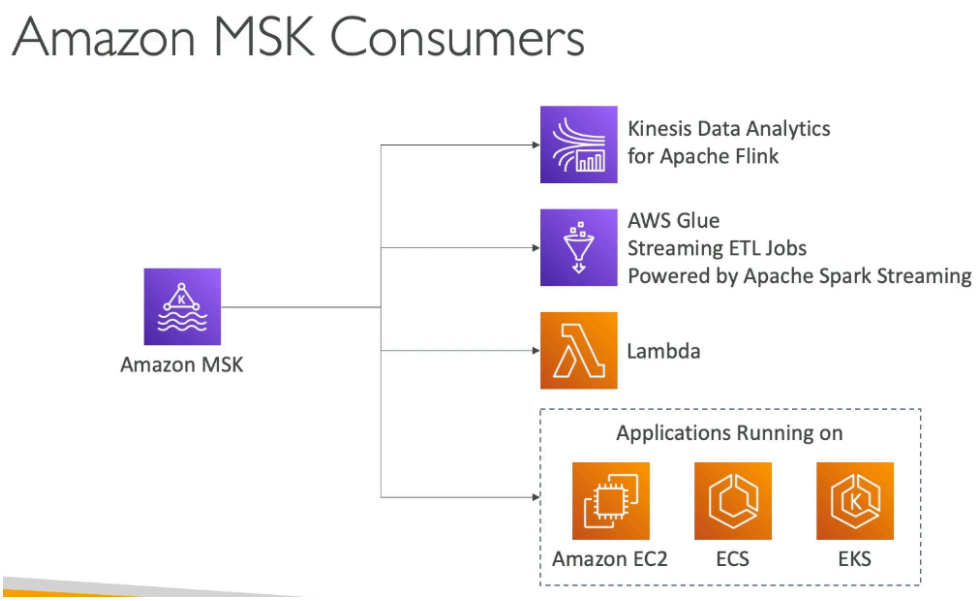


- Alternative to Kinesis, fully managed Apache Kafka on AWS
  - Create, update, delete clusters
  - MSK creates / manages Kafka brokers nodes & Zookeeper nodes for you
  - Deploy MSK cluster in VPC, multi AZ; auto recovery from common failures
  - Data stored on EBS volumes for as long as you want
- MSK Serverless
  - Run Kafka on MSK without managing capacity, where MSK auto provisions resources and scales compute / storage

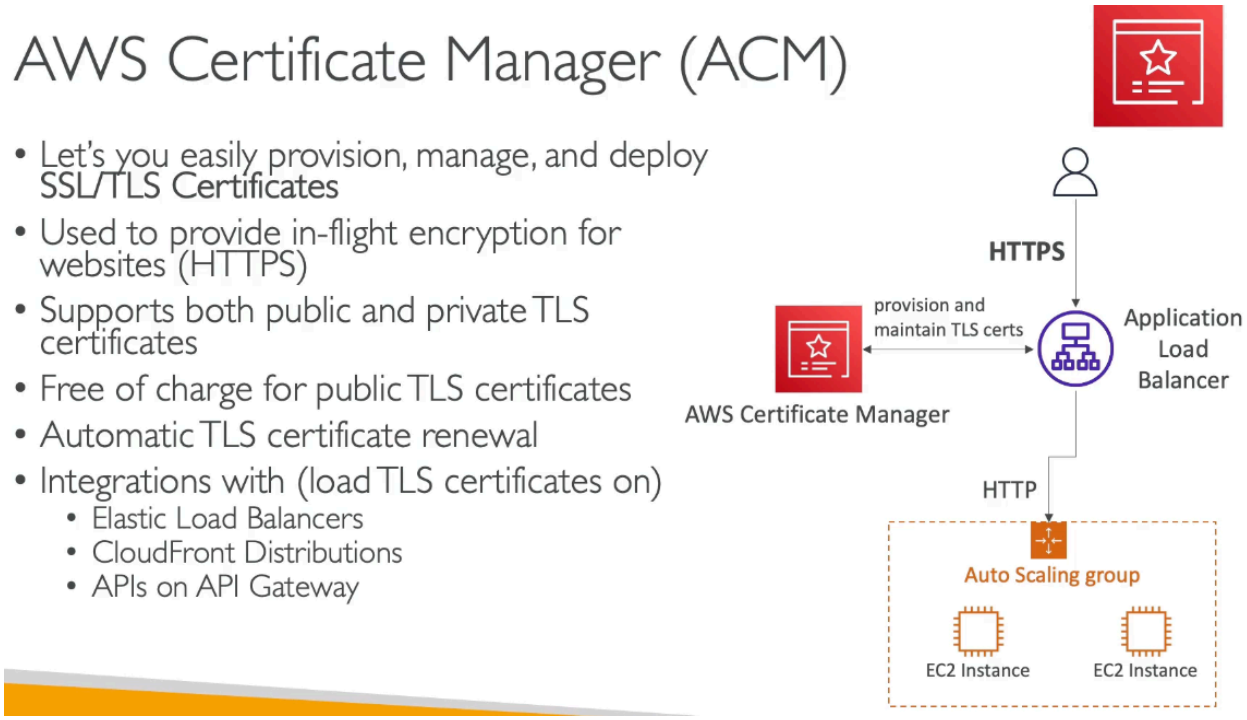
Kinesis Data Stream vs MSK



MSK Consumers



AWS Certificate Manager (ACM)



- Easily provision, manage, deploy SSL / TLS Certificates for in flight encryption for websites (HTTPS)
- Supports both public and private TLS certificates, free of charge for public TLS certificates and automatic TLS certificate renewal
- Integrations with (load TLS certificates on) ELB, CloudFront distributions, APIs on API GW
- Anytime for in flight encryption and generate certificates think ACM

ACM Private CA (Certificate Authority)

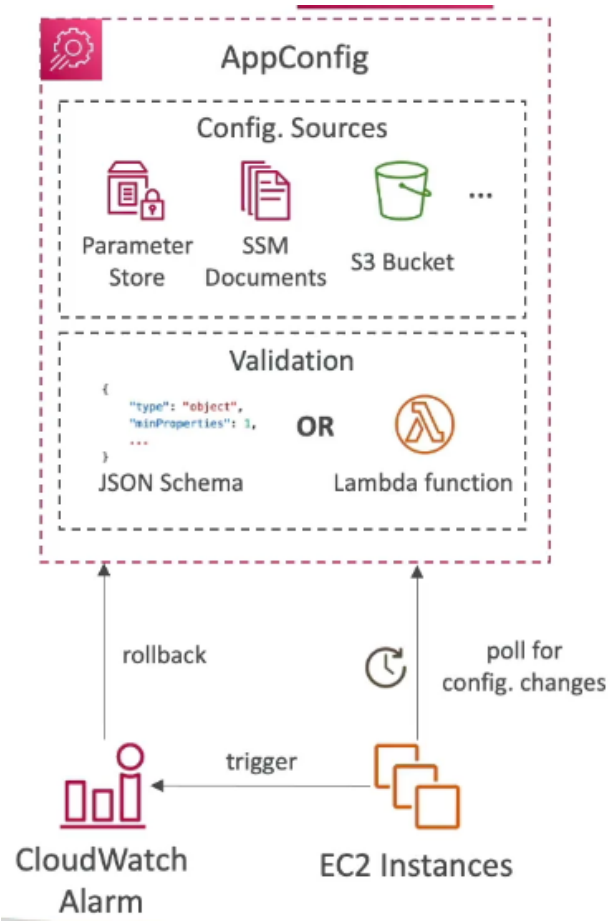
- Managed service to create private Certificate Authority including root and sub ordinaries CAs
- Can issue and deploy end entity x.509 certificates
- Certificates are trusted only by organization not for public internet
- Works for AWS services that are integrated with ACM
- Uses cases:
    - Encrypted TLS communication, authenticate users, etc…

Amazon Macie

- Fully managed data security that uses ML and pattern matching to discover and protect sensitive data in AWS → alerts

AWS AppConfig

- Configure, validate, and deploy dynamic configuration
  - Validate config changes before deployment with:
    - JSON schema check or Lambda function
- Deploy dynamic configuration changes to applications independently of code deployments
- Feature flags, application tuning, allow / block IP…
  - Gradually deploy config changes and rollback if needed
- Use with EC2 instance, lambda, ECS, EKS…



CloudWatch Evidently

- Safely validate new features by serving to % of users to reduce risk, collect experiment data, monitoring performance…
- Launches (feature flags)
  - Enable and disable features for subset of users
- Experiments (AB testing)
- Overrides: pre-define a variation for specific user
- Store evaluation events in CloudWatch logs or S3