Amazon S3

Amazon S3 Overview - Buckets

- Amazon S3 allows people to store objects (files) in “buckets” (directories)
- Buckets must have a globally unique name
- Buckets are defined at the region level
    - Naming convention
    - No uppercase
    - No underscore
    - 3-63 characters long
    - Not an IP
    - Must start with lowercase letter or number

Amazon S3 Overview - Objects

- Objects (files) have a Key
- The key is the FULL path:
- s3://my-bucket/my_file.txt
- s3://my-bucket/my_folder1/another_folder/my_file.txt
- The key is composed of prefix + object name
- s3://my-bucket/my_folder1/another_folder/my_file.txt
- There’s no concept of “directories” within buckets

(although the UI will trick you to think otherwise)

- Just keys with very long names that contain slashes (“/”)

Object values are the content of the body:

  - Max Object Size is 5TB (5000GB)
  - If uploading more than 5GB, must use “multi-part upload”

- Metadata (list of text key / value pairs – system or user metadata)
- Tags (Unicode key / value pair – up to 10) – useful for security / lifecycle
- Version ID (if versioning is enabled)

Amazon S3 - Versioning

- You can version your files in Amazon S3
- It is enabled at the bucket level
- Same key overwrite will increment the “version”: 1, 2, 3….
- It is best practice to version your buckets
    - Protect against unintended deletes (ability to restore a version)
    - Easy roll back to previous version
- Notes:
    - Any file that is not versioned prior to enabling versioning will have version “null”
    - Suspending versioning does not delete the previous versions

Create a S3 bucket and upload file -Presigned Url ,Versioning , Delete Marker

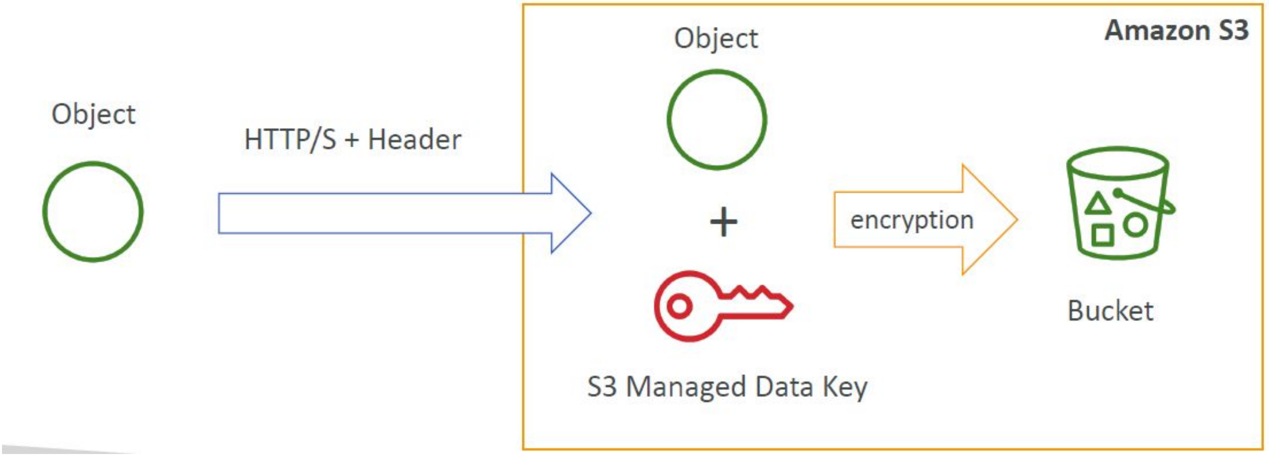
Pre signed Url : Signed with your credentials

S3 Encryption for Objects

- There are 4 methods of encrypting objects in S3
    - SSE-S3: encrypts S3 objects using keys handled & managed by AWS
    - SSE-KMS: leverage AWS Key Management Service to manage encryption keys
    - SSE-C: when you want to manage your own encryption keys
    - Client Side Encryption

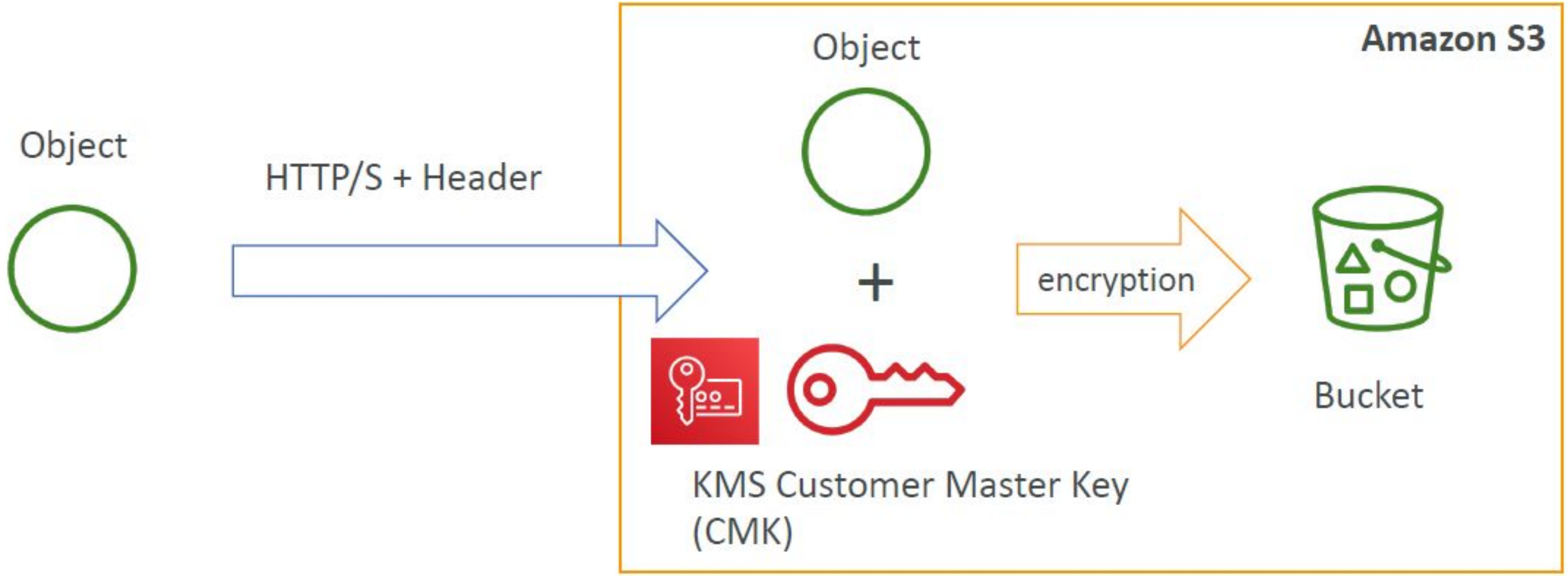
SSE-S3

- SSE-S3: encryption using keys handled & managed by Amazon S3
- Object is encrypted server side
- AES-256 encryption type
- Must set header: “x-amz-server-side-encryption": "AES256"



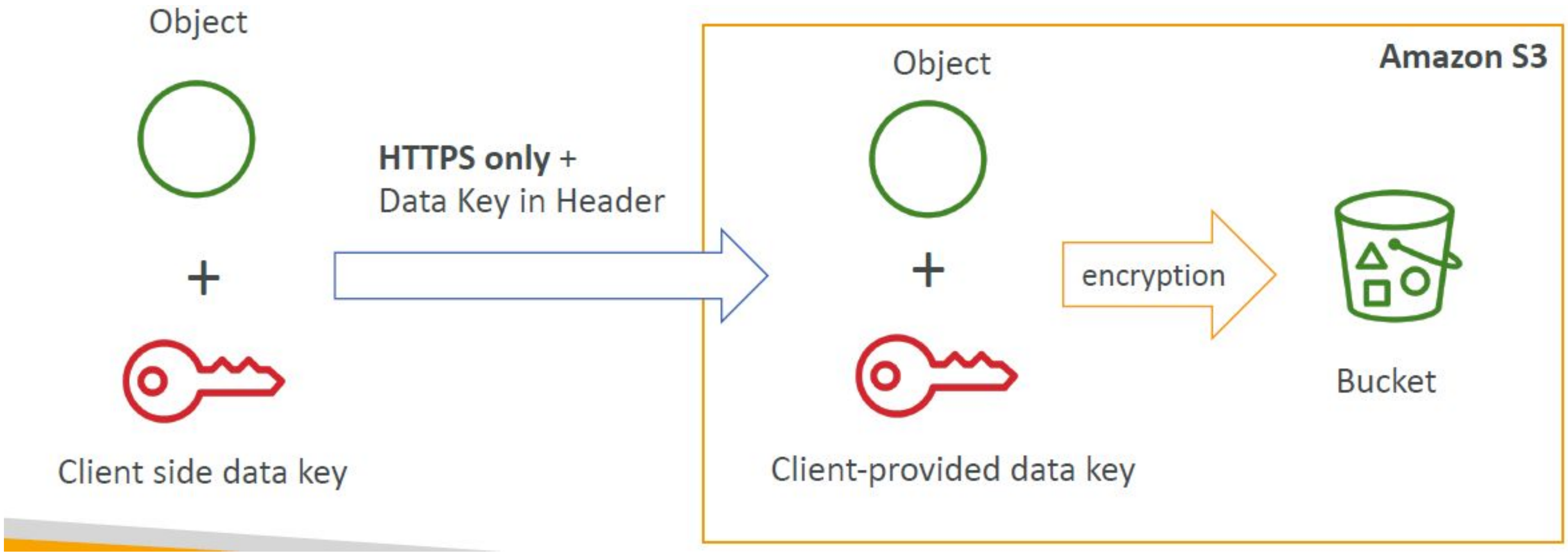
SSE-KMS

- SSE-KMS: encryption using keys handled & managed by KMS
- KMS Advantages: user control + audit trail
- Object is encrypted server side
- Must set header: “x-amz-server-side-encryption": ”aws:kms"



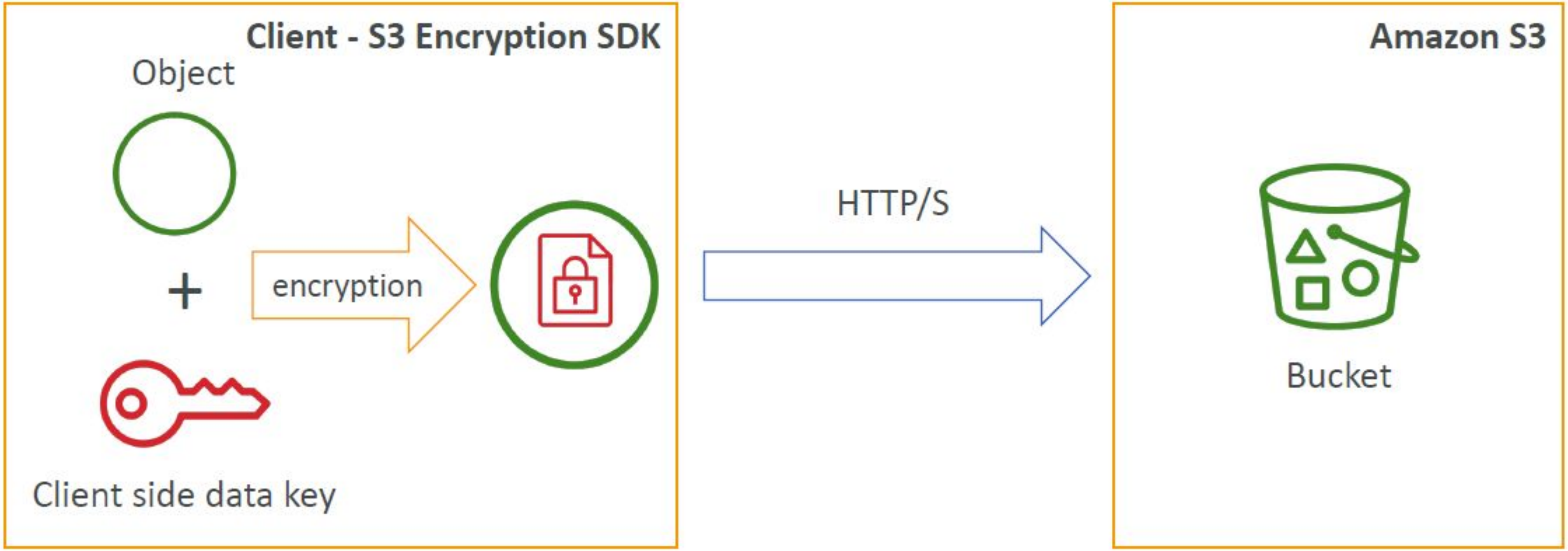
SSE-C

- SSE-C: server-side encryption using data keys fully managed by the customer outside of AWS
- Amazon S3 does not store the encryption key you provide
- HTTPS must be used
- Encryption key must provided in HTTP headers, for every HTTP request made



Client Side Encryption

- Client library such as the Amazon S3 Encryption Client
- Clients must encrypt data themselves before sending to S3
- Clients must decrypt data themselves when retrieving from S3
- Customer fully manages the keys and encryption cycle



Lab : Encryption on objects and Default encryption

SSE-C : CLI

Encryption in transit (SSL/TLS)

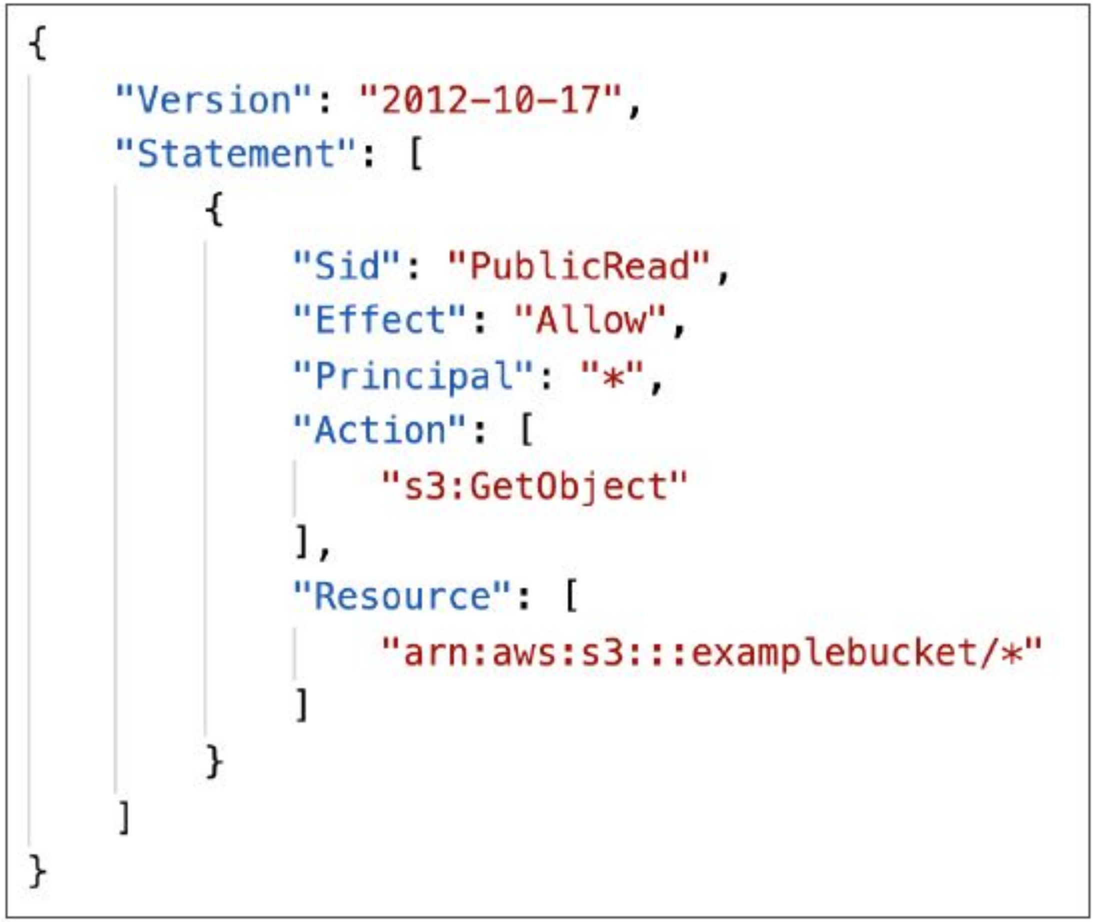
- Amazon S3 exposes:
  - HTTP endpoint: non encrypted
  - HTTPS endpoint: encryption in flight
- You’re free to use the endpoint you want, but HTTPS is recommended
- Most clients would use the HTTPS endpoint by default
- HTTPS is mandatory for SSE-C
- Encryption in flight is also called SSL / TLS

S3 Security

- User based
    - IAM policies - which API calls should be allowed for a specific user from IAM console
- Resource Based
    - Bucket Policies - bucket wide rules from the S3 console - allows cross account
    - Object Access Control List (ACL) – finer grain
    - Bucket Access Control List (ACL) – less common
- Note: an IAM principal can access an S3 object if
    - the user IAM permissions allow it OR the resource policy ALLOWS it
    - AND there’s no explicit DENY

S3 Bucket Policies

- JSON based policies
- Resources: buckets and objects
- Actions: Set of API to Allow or Deny
- Effect: Allow / Deny
- Principal: The account or user to apply the policy to
- Use S3 bucket for policy to:
    - Grant public access to the bucket
    - Force objects to be encrypted at upload
    - Grant access to another account (Cross Account)



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicRead",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::examplebucket/*"
            ]
        }
    ]
}
```

Lab : Hands on Bucket Policy

To deny the upload if encryption is not set

S3 Websites

- S3 can host static websites and have them accessible on the www
- The website URL will be:
- <bucket-name>.s3-website-<AWS-region>.amazonaws.com

OR

- <bucket-name>.s3-website.<AWS-region>.amazonaws.com
- If you get a 403 (Forbidden) error, make sure the bucket policy allows public reads!

Lab : HandsOn

S3 Security - Other

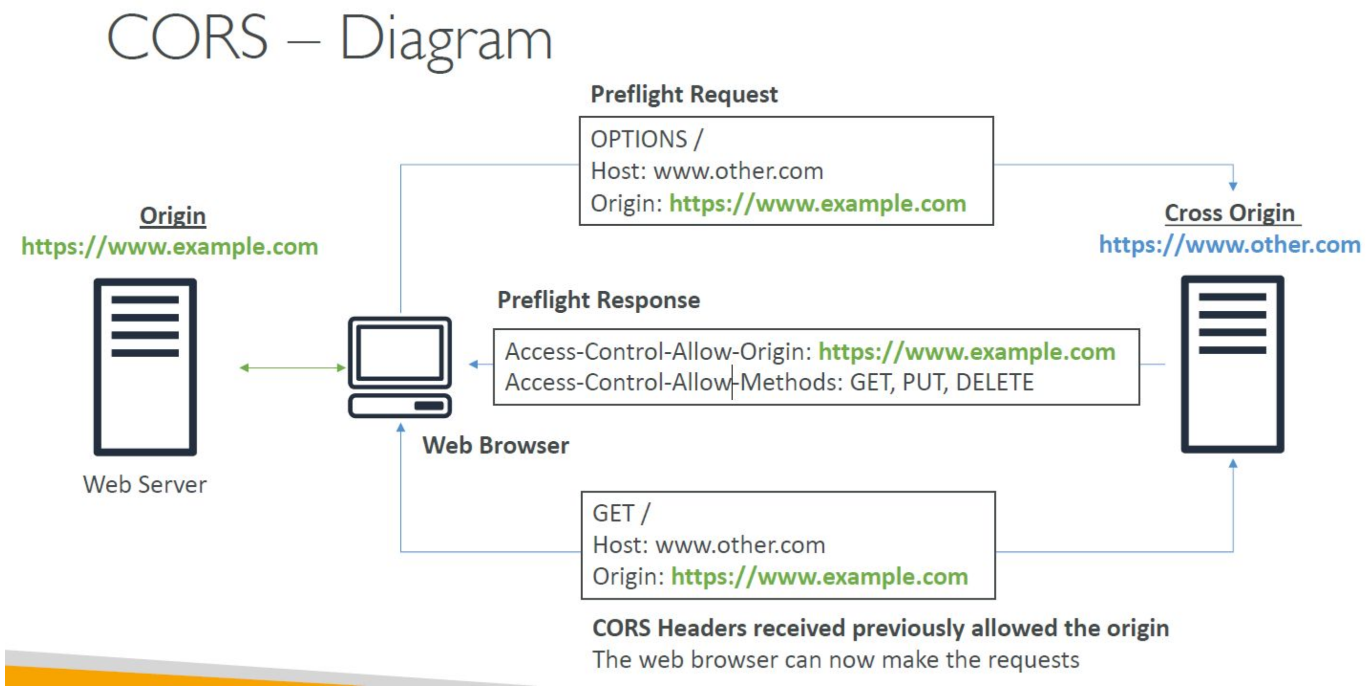
- Networking:
- Supports VPC Endpoints (for instances in VPC without www internet)
- Logging and Audit:
- S3 Access Logs can be stored in other S3 bucket
- API calls can be logged in AWS CloudTrail
- User Security:
- MFA Delete: MFA (multi factor authentication) can be required in versioned buckets to delete objects
- Pre-Signed URLs: URLs that are valid only for a limited time (ex: premium video service for logged in users)

Bucket settings for Block Public Access

- Block public access to buckets and objects granted through
    - new access control lists (ACLs)
    - any access control lists (ACLs)
    - new public bucket or access point policies
- Block public and cross-account access to buckets and objects through any public bucket or access point policies
- These settings were created to prevent company data leaks
- If you know your bucket should never be public, leave these on
- Can be set at the account level

CORS - Explained

- An origin is a scheme (protocol), host (domain) and port
- E.g.: https://www.example.com (implied port is 443 for HTTPS, 80 for HTTP)
- CORS means Cross-Origin Resource Sharing
- Web Browser based mechanism to allow requests to other origins while visiting the main origin
- Same origin: http://example.com/app1 & http://example.com/app2
- Different origins: http://www.example.com & http://other.example.com
- The requests won't be fulfilled unless the other origin allows for the requests, using CORS Headers (ex: Access-Control-Allow-Origin)



CORS – Diagram

Preflight Request

OPTIONS /
Host: www.other.com
Origin: https://www.example.com

Preflight Response

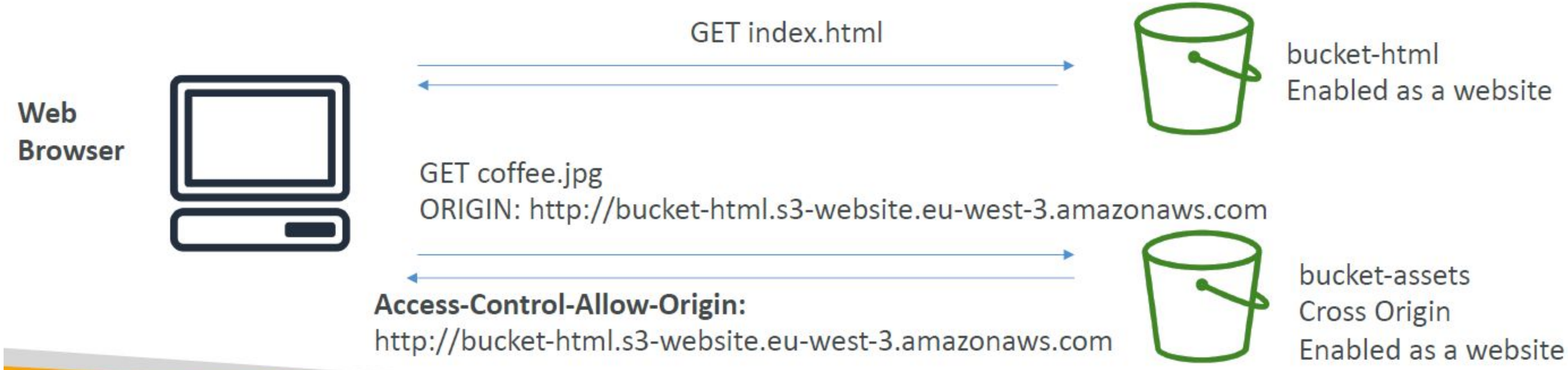
Access-Control-Allow-Origin: https://www.example.com
Access-Control-Allow-Methods: GET, PUT, DELETE

GET /
Host: www.other.com
Origin: https://www.example.com

CORS Headers received previously allowed the origin
The web browser can now make the requests

S3 CORS

- If a client does a cross-origin request on our S3 bucket, we need to enable the correct CORS headers
- It's a popular exam question
- You can allow for a specific origin or for * (all origins)



Web
Browser
GET index.html
bucket-html
Enabled as a website
GET coffee.jpg
ORIGIN: http://bucket-html.s3-website.eu-west-3.amazonaws.com
Access-Control-Allow-Origin:
http://bucket-html.s3-website.eu-west-3.amazonaws.com
bucket-assets
Cross Origin
Enabled as a website

CORS hands on : Same origin and Cross origin

Amazon S3 - Consistency Model

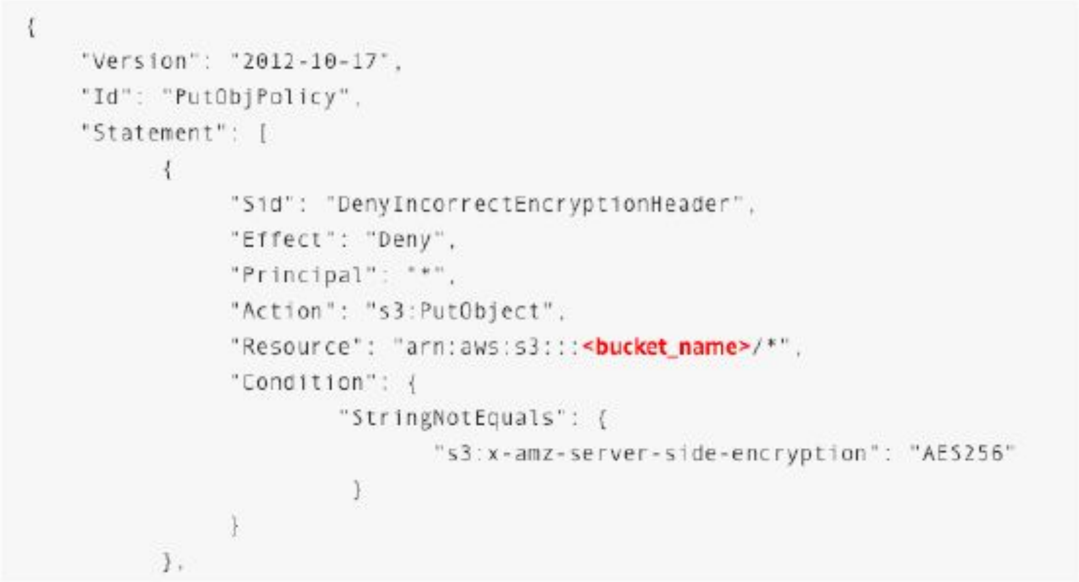
- Read after write consistency for PUTS of new objects
  - As soon as a new object is written, we can retrieve it ex: (PUT 200 => GET 200)
- This is true, except if we did a GET before to see if the object existed ex: (GET 404 => PUT 200 => GET 404) – eventually consistent
- Eventual Consistency for DELETES and PUTS of existing objects
  - If we read an object after updating, we might get the older version ex: (PUT 200 => PUT 200 => GET 200 (might be older version))
- If we delete an object, we might still be able to retrieve it for a short time ex: (DELETE 200 => GET 200)
- Note: there's no way to request "strong consistency"

S3 MFA-Delete

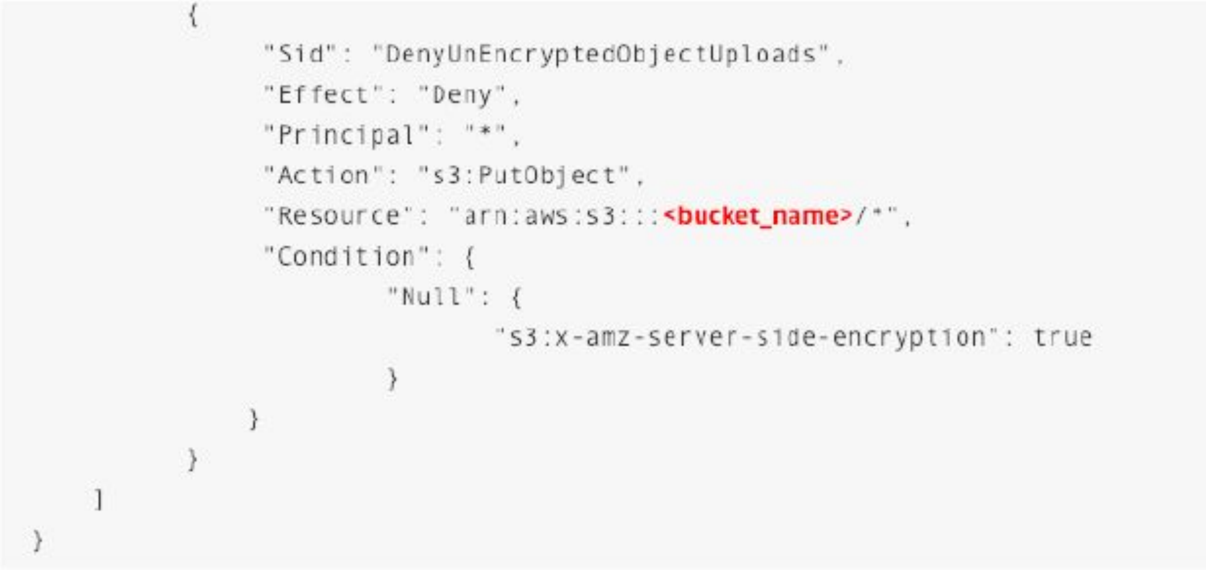
- MFA (multi factor authentication) forces user to generate a code on a device (usually a mobile phone or hardware) before doing important operations on S3
- To use MFA-Delete, enable Versioning on the S3 bucket
- You will need MFA to
  - permanently delete an object version
  - suspend versioning on the bucket
- You won’t need MFA for
  - enabling versioning
  - listing deleted versions
- Only the bucket owner (root account) can enable/disable MFA-Delete
- MFA-Delete currently can only be enabled using the CLI

S3 Default Encryption vs Bucket Policies

- The old way to enable default encryption was to use a bucket policy and refuse any HTTP command without the proper headers:



```
{
    "Version": "2012-10-17",
    "Id": "PutObjPolicy",
    "Statement": [
          {
                "Sid": "DenyIncorrectEncryptionHeader",
                "Effect": "Deny",
                "Principal": "*",
                "Action": "s3:PutObject",
                "Resource": "arn:aws:s3:::<bucket_name>/*",
                "Condition": {
                        "StringNotEquals": {
                               "s3:x-amz-server-side-encryption": "AES256"
                        }
                }
          },
```

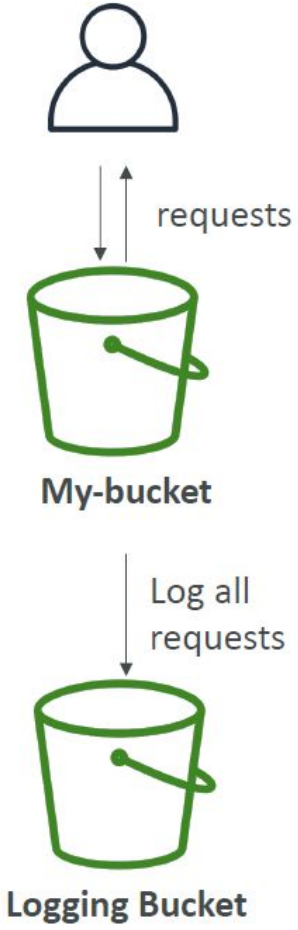


```
          {
                "Sid": "DenyUnEncryptedObjectUploads",
                "Effect": "Deny",
                "Principal": "*",
                "Action": "s3:PutObject",
                "Resource": "arn:aws:s3:::<bucket_name>/*",
                "Condition": {
                        "Null": {
                               "s3:x-amz-server-side-encryption": true
                        }
                }
          }
     ]
}
```

- The new way is to use the "default encryption" option in S3
- Note: Bucket Policies are evaluated before "default encryption"

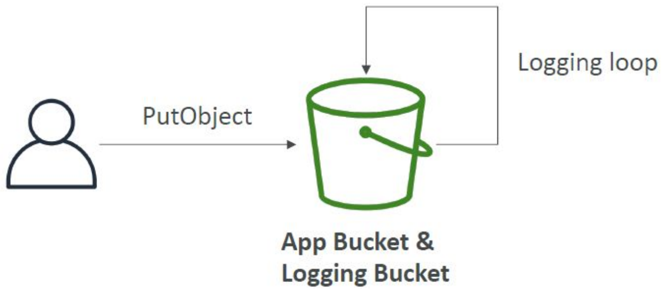
S3 Access Logs

- For audit purpose, you may want to log all access to S3 buckets
- Any request made to S3, from any account, authorized or denied, will be logged into another S3 bucket
- That data can be analyzed using data analysis tools…
- Or Amazon Athena as we'll see later in this section!



S3 Access Logs: Warning

- Do not set your logging bucket to be the monitored bucket
- It will create a logging loop, and your bucket will grow in size exponentially



S3 Replication (CRR & SRR)

- **Must enable versioning in source and destination**
- Cross Region Replication (CRR)
- Same Region Replication (SRR)
- Buckets can be in different accounts
- Copying is asynchronous
- Must give proper IAM permissions to S3
- CRR - Use cases: compliance, lower latency access, replication across accounts
- SRR – Use cases: log aggregation, live replication between production and test accounts

S3 Replication – Notes

- After activating, only new objects are replicated (not retroactive)
- For DELETE operations:
- If you delete without a version ID, it adds a delete marker, not replicated
- If you delete with a version ID, it deletes in the source, not replicated
- There is 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 Pre-Signed URLs

- Can generate pre-signed URLs using SDK or CLI
- For downloads (easy, can use the CLI)
- For uploads (harder, must use the SDK)
- Valid for a default of 3600 seconds, can change timeout with --expires-in [TIME_BY_SECONDS] argument
- Users given a pre-signed URL inherit the permissions of the person who generated the URL for GET / PUT
- Examples :
- Allow only logged-in users to download a premium video on your S3 bucket
- Allow an ever changing list of users to download files by generating URLs dynamically
- Allow temporarily a user to upload a file to a precise location in our bucket

S3 Storage Classes

- Amazon S3 Standard - General Purpose
- Amazon S3 Standard-Infrequent Access (IA)
- Amazon S3 One Zone-Infrequent Access
- Amazon S3 Intelligent Tiering
- Amazon Glacier
- Amazon Glacier Deep Archive
- Amazon S3 Reduced Redundancy Storage (deprecated - omitted)

S3 Standard – General Purpose

- High durability (99.999999999%) 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
- 99.99% Availability over a given year
- Sustain 2 concurrent facility failures
- Use Cases: Big Data analytics, mobile & gaming applications, content distribution…

S3 Standard – Infrequent Access (IA)

- Suitable for data that is less frequently accessed, but requires rapid access when needed
- High durability (99.999999999%) of objects across multiple AZs
- 99.9% Availability
- Low cost compared to Amazon S3 Standard
- Sustain 2 concurrent facility failures
- Use Cases: As a data store for disaster recovery, backups…

S3 One Zone - Infrequent Access (IA)

- Same as IA but data is stored in a single AZ
- High durability (99.999999999%) of objects in a single AZ; data lost when AZ is destroyed
- 99.5% Availability
- Low latency and high throughput performance
- Supports SSL for data at transit and encryption at rest
- Low cost compared to IA (by 20%)
- Use Cases: Storing secondary backup copies of on-premise data, or storing data you can recreate

S3 Intelligent Tiering

- Same low latency and high throughput performance of S3 Standard
- Small monthly monitoring and auto-tiering fee
- Automatically moves objects between two access tiers based on changing access patterns
- Designed for durability of 99.999999999% of objects across multiple Availability Zones
- Resilient against events that impact an entire Availability Zone
- Designed for 99.9% availability over a given year

Amazon Glacier

- **Low cost object storage meant for archiving / backup**
- Data is retained for the longer term (10s of years)
- Alternative to on-premise magnetic tape storage
- Average annual durability is 99.999999999%
- Cost per storage per month ($0.004 / GB) + retrieval cost
- Each item in Glacier is called “Archive” (up to 40TB)
- Archives are stored in ”Vaults”

Amazon Glacier & Glacier Deep Archive

- Amazon Glacier – 3 retrieval options:
- Expedited (1 to 5 minutes)
- Standard (3 to 5 hours)
- Bulk (5 to 12 hours)
- Minimum storage duration of 90 days

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

S3 Storage Classes Comparison

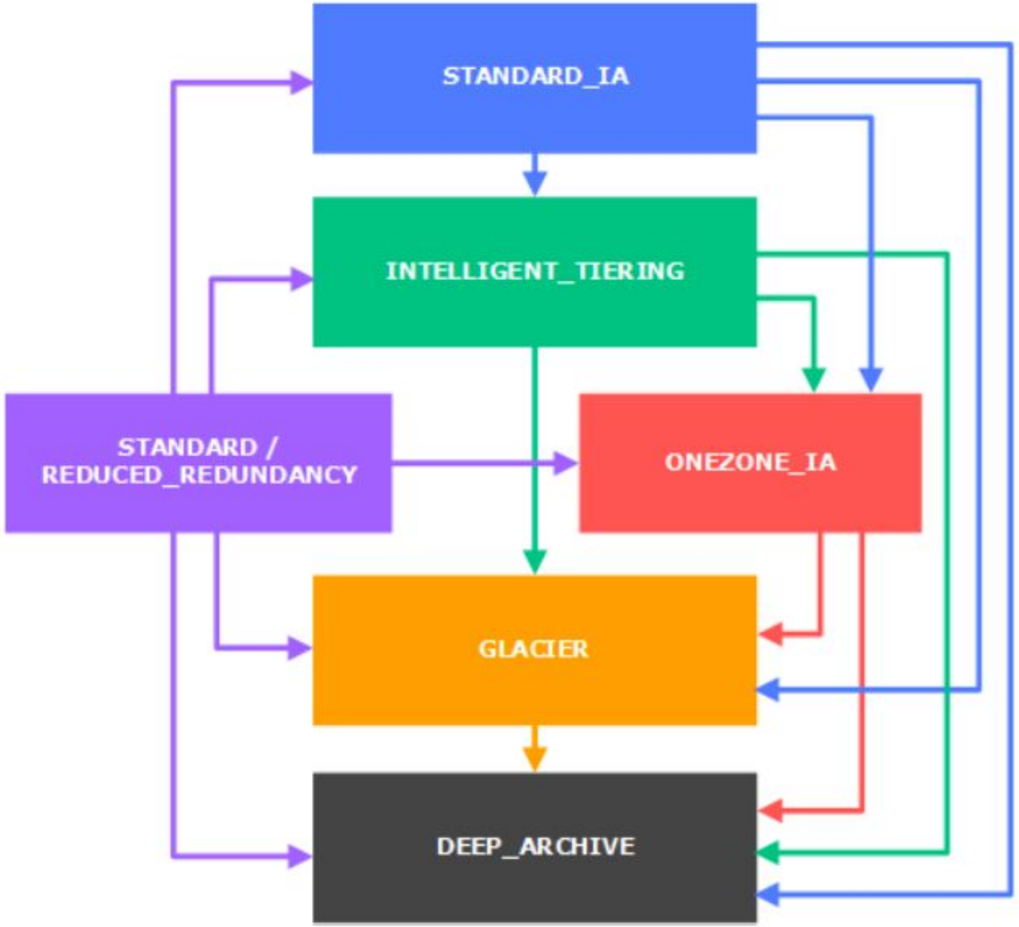
	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
Designed for durability	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)	99.999999999% (11 9's)
Designed for availability	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Availability SLA	99.9%	99%	99%	99%	99.9%	99.9%
Availability Zones	≥3	≥3	≥3	1	≥3	≥3
Minimum capacity charge per object	N/A	N/A	128KB	128KB	40KB	40KB
Minimum storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved

S3 – Moving between storage classes

- You can transition objects between storage classes

- For infrequently accessed object, move them to STANDARD_IA
- For archive objects you don't need in real-time, GLACIER or DEEP_ARCHIVE

- Moving objects can be automated using a **lifecycle configuration**



S3 Lifecycle Rules

- Transition actions: It defines when objects are transitioned to another storage class.
    - Move objects to Standard IA class 60 days after creation
    - Move to Glacier for archiving after 6 months
- Expiration actions: configure objects to expire (delete) after some time
    - Access log files can be set to delete after a 365 days
    - Can be used to delete old versions of files (if versioning is enabled)
    - Can be used to delete incomplete multi-part uploads
- Rules can be created for a certain prefix (ex - s3://mybucket/mp3/*)
- Rules can be created for certain objects tags (ex - Department: Finance)

S3 Lifecycle Rules – Scenario 1

Your application on EC2 creates images thumbnails after profile photos are uploaded to Amazon S3. These thumbnails can be easily recreated, and only need to be kept for 45 days. The source images should be able to be immediately retrieved for these 45 days, and afterwards, the user can wait up to 6 hours. How would you design this?

- S3 source images can be on STANDARD, with a lifecycle configuration to transition them to GLACIER after 45 days.
- S3 thumbnails can be on ONEZONE_IA, with a lifecycle configuration to expire them (delete them) after 45 days.

S3 Lifecycle Rules – Scenario 2

- A rule in your company states that you should be able to recover your deleted S3 objects immediately for 15 days, although this may happen rarely. After this time, and for up to 365 days, deleted objects should be recoverable within 48 hours.

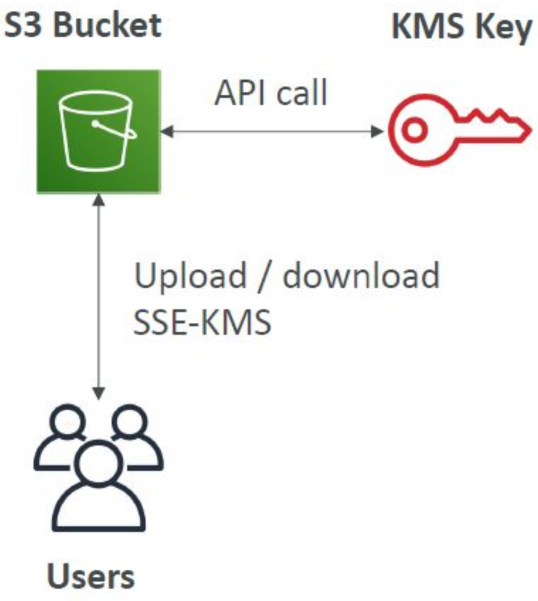
- You need to enable S3 versioning in order to have object versions, so that "deleted objects" are in fact hidden by a "delete marker" and can be recovered
- You can transition these "noncurrent versions" of the object to S3_IA
- You can transition afterwards these "noncurrent versions" to DEEP_ARCHIVE

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 and 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

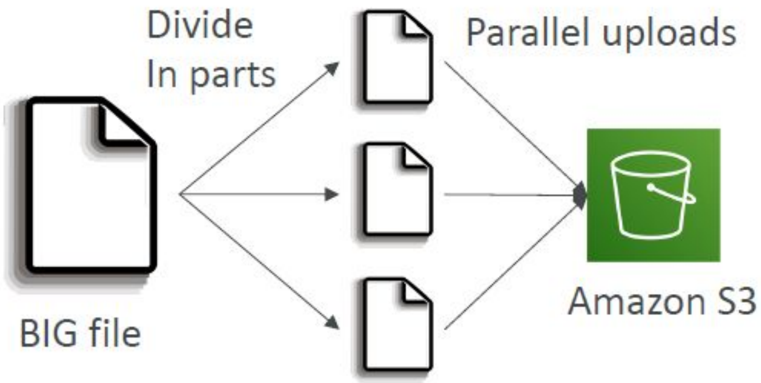
S3 – KMS Limitation

- If you use SSE-KMS, you may be impacted by the KMS limits
- When you upload, it calls the GenerateDataKey KMS API
- When you download, it calls the Decrypt KMS API
- Count towards the KMS quota per second (5500, 10000, 30000 req/s based on region)
- As of today, you cannot request a quota increase for KMS

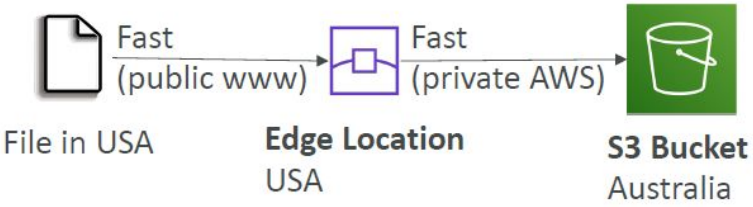


S3 Performance

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



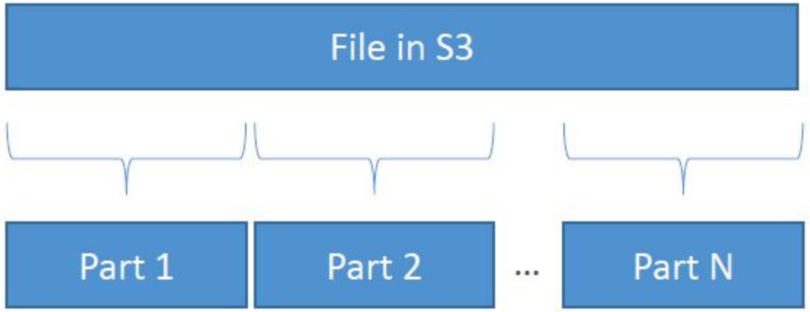
- S3 Transfer Acceleration (upload only)
- Increase transfer speed by transferring file to an AWS edge location which will forward the data to the S3 bucket in the target region
- Compatible with multi-part upload



S3 Performance – S3 Byte-Range Fetches

- Parallelize GETs by requesting specific byte ranges
- Better resilience in case of failures

Can be used to speed up downloads



File in S3

Part 1

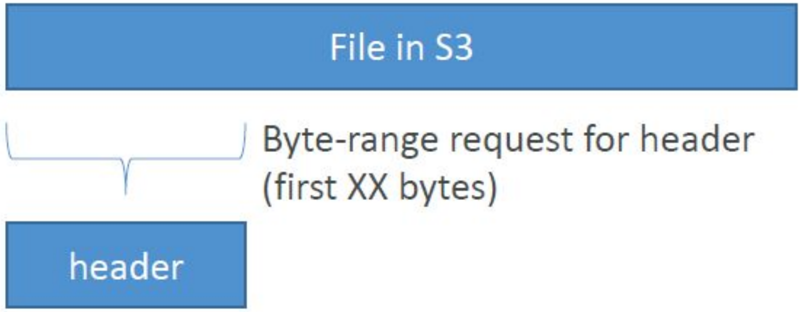
Part 2

...

Part N

Requests in parallel

Can be used to retrieve only partial data (for example the head of a file)



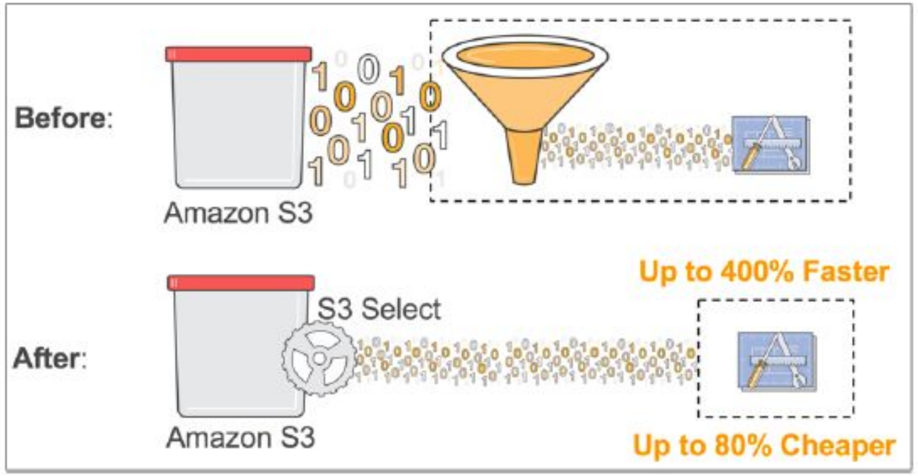
File in S3

Byte-range request for header (first XX bytes)

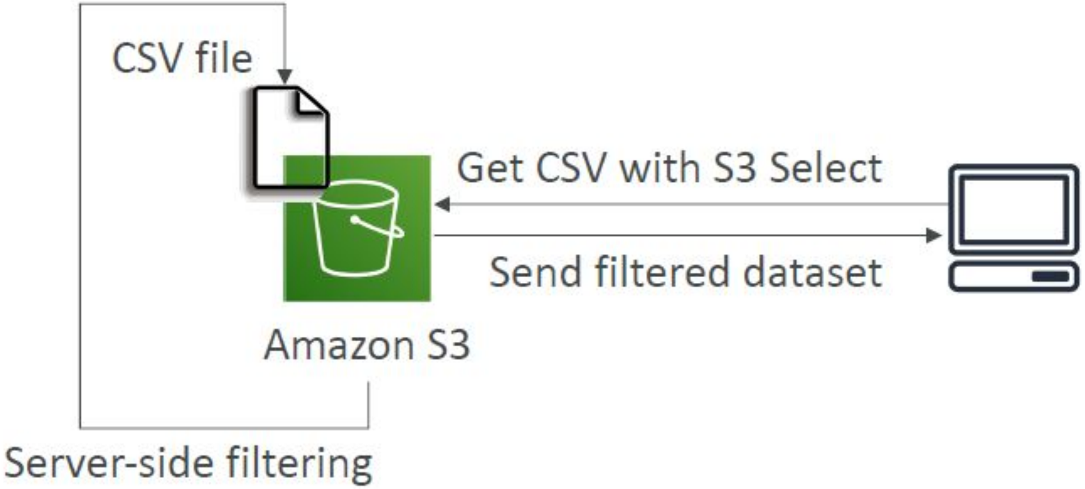
header

S3 Select & Glacier Select

- Retrieve less data using SQL by performing server side filtering
- Can filter by rows & columns (simple SQL statements)
- Less network transfer, less CPU cost client-side

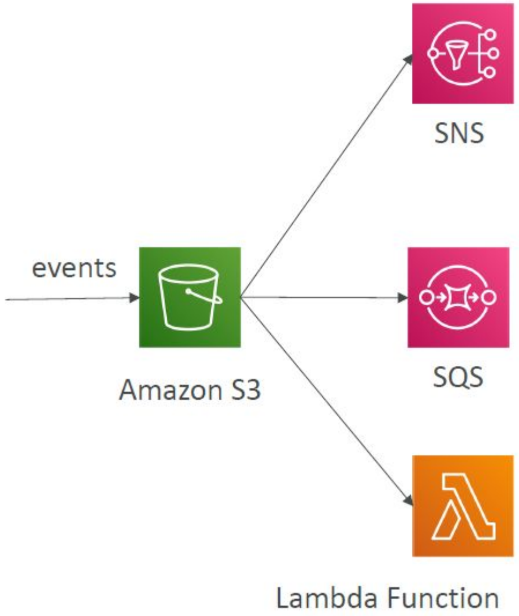


https://aws.amazon.com/blogs/aws/s3-glacier-select/



S3 Event Notifications

- S3:ObjectCreated, S3:ObjectRemoved, S3:ObjectRestore, S3:ReplicatioN
- Object name filtering possible (*.jpg)
- Use case: generate thumbnails of images uploaded to S3
- Can create as many “S3 events” as desired
- S3 event notifications typically deliver events in seconds but can sometimes take a minute or longer
- If two writes are made to a single non-versioned object at the same time, it is possible that 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.



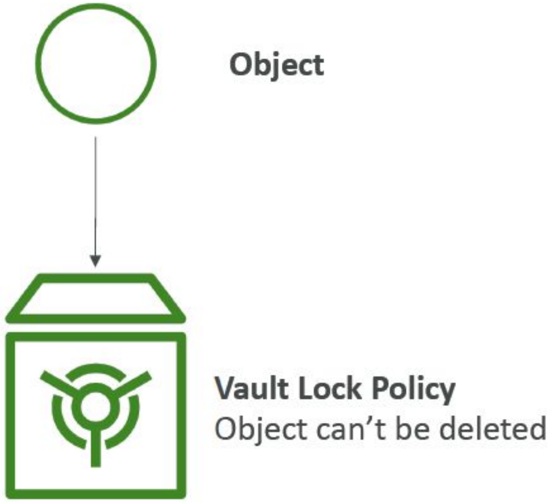
Lab S3 with event notification

AWS Athena

- Serverless service to perform analytics directly against S3 files
- Uses SQL language to query the files
- Has a JDBC / ODBC driver
- Charged per query and amount of data scanned
- Supports CSV, JSON, ORC, Avro, and Parquet (built on Presto)
- Use cases: Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, CloudTrail
- Exam Tip: Analyze data directly on S3 => use Athena

S3 Object Lock & Glacier Vault Lock

- S3 Object Lock
    - Adopt a WORM (Write Once Read Many) model
    - Block an object version deletion for a specified amount of time
- Glacier Vault Lock
    - Adopt a WORM (Write Once Read Many) model
    - Lock the policy for future edits (can no longer be changed)
    - Helpful for compliance and data retention



CloudFront

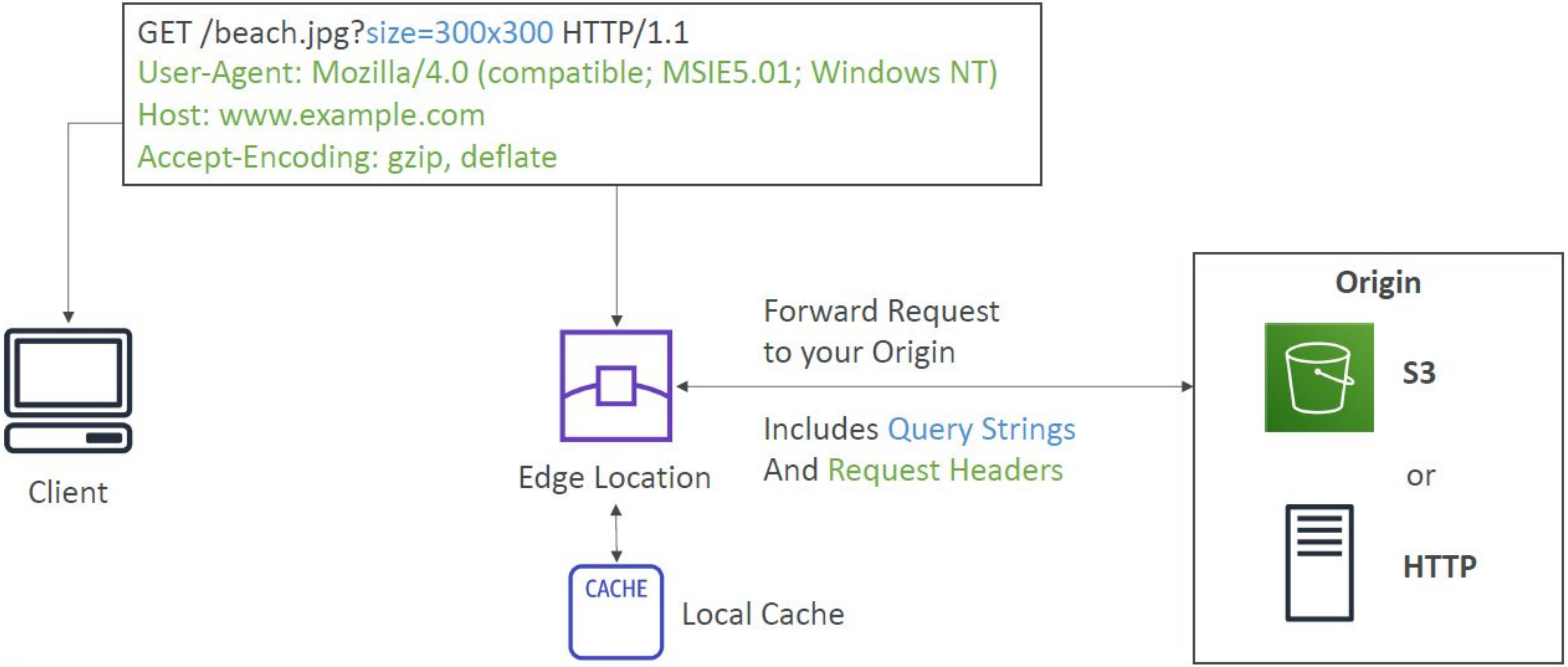
AWS CloudFront

- Content Delivery Network (CDN)
- Improves read performance, content is cached at the edge
- 216 Point of Presence globally (edge locations)
- DDoS protection, integration with Shield, AWS Web Application Firewall
- Can expose external HTTPS and can talk to internal HTTPS backends

CloudFront – Origins

- S3 bucket
  - For distributing files and caching them at the edge
  - Enhanced security with CloudFront Origin Access Identity (OAI)
  - CloudFront can be used as an ingress (to upload files to S3)
- Custom Origin (HTTP)
  - Application Load Balancer
  - EC2 instance
  - S3 website (must first enable the bucket as a static S3 website)
  - Any HTTP backend you want

CloudFront at a high level



```
GET /beach.jpg?size=300x300 HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.example.com
Accept-Encoding: gzip, deflate
```

Client

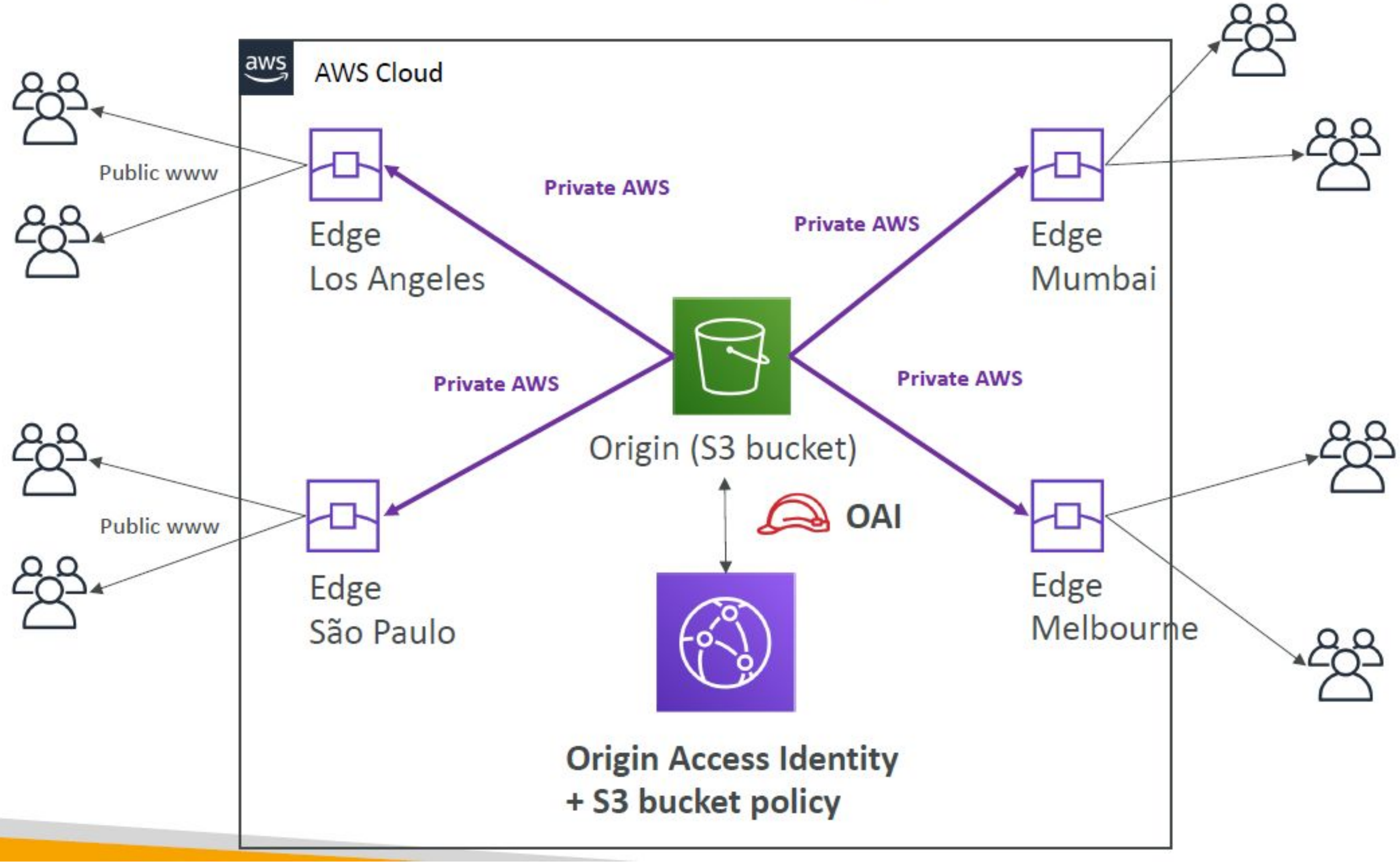
Edge Location

Forward Request
to your Origin

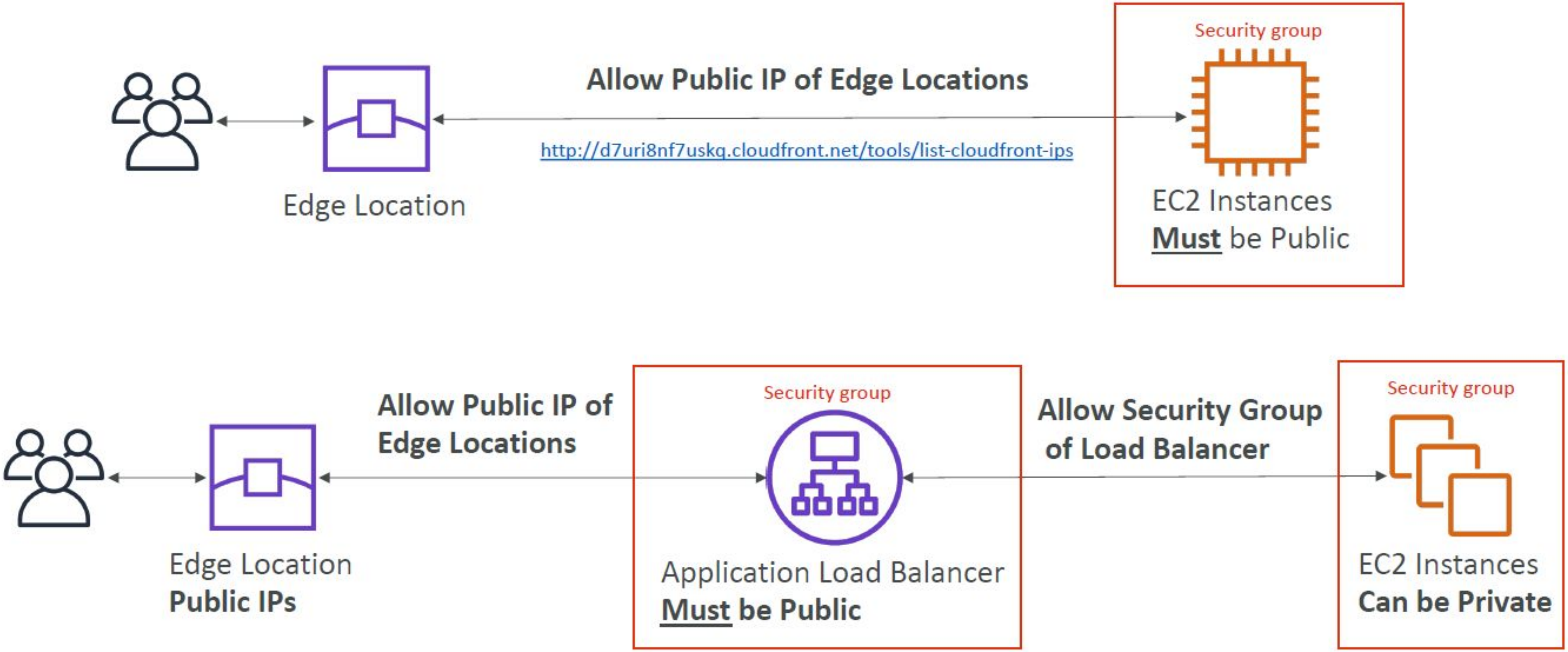
Includes Query Strings
And Request Headers

Local Cache

CloudFront – S3 as an Origin



CloudFront – ALB or EC2 as an origin



Allow Public IP of Edge Locations

http://d7uri8nf7uskq.cloudfront.net/tools/list-cloudfront-ips

CloudFront Geo Restriction

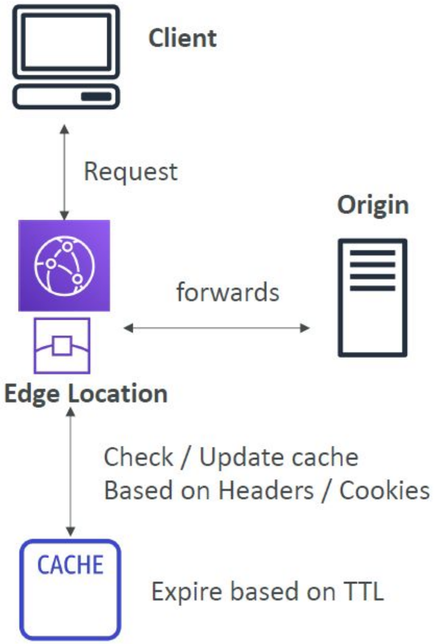
- You can restrict who can access your distribution
  - Whitelist: Allow your users to access your content only if they're in one of the countries on a list of approved countries.
  - Blacklist: Prevent your users from accessing your content if they're in one of the countries on a blacklist of banned countries.
- The “country” is determined using a 3rd party Geo-IP database
- **Use case: Copyright Laws to control access to content**

CloudFront vs S3 Cross Region Replication

- CloudFront:
  - Global Edge network
  - Files are cached for a TTL (maybe a day)
  - Great for static content that must be available everywhere
- S3 Cross Region Replication:
  - Must be setup for each region you want replication to happen
  - Files are updated in near real-time
  - Read only
  - Great for dynamic content that needs to be available at low-latency in few regions

CloudFront Caching

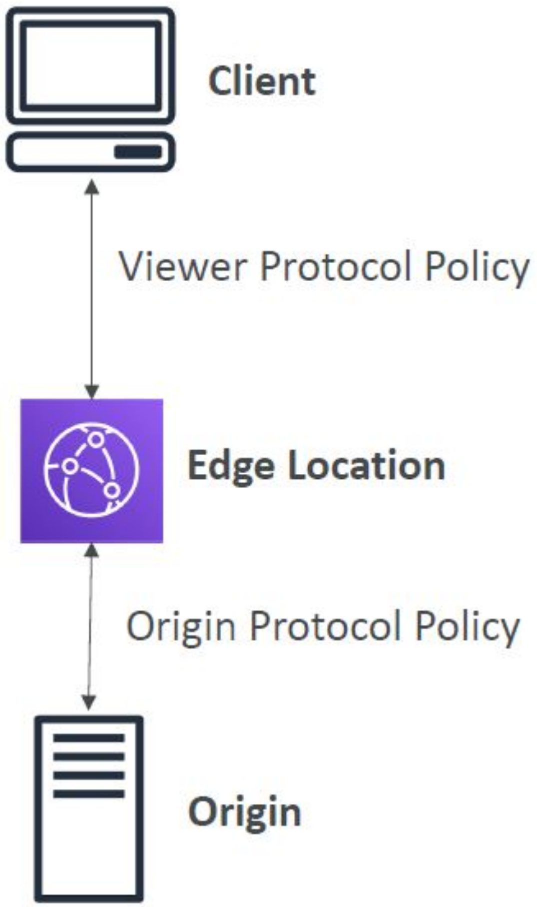
- Cache based on
  - Headers
  - Session Cookies
  - Query String Parameters
- The cache lives at each CloudFront Edge Location
- You want to maximize the cache hit rate to minimize requests on the origin
- Control the TTL (0 seconds to 1 year), can be set by the origin using the Cache-Control header, Expires header…
- You can invalidate part of the cache using the CreateInvalidation API



CloudFront and HTTPS

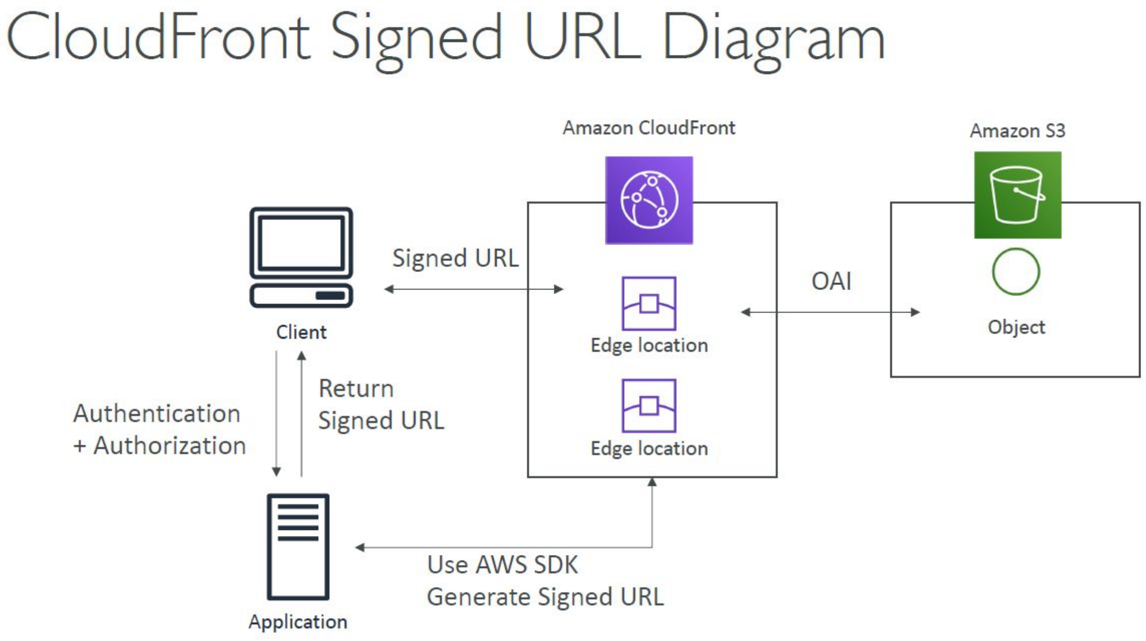
- Viewer Protocol Policy:
  - Redirect HTTP to HTTPS
  - Or use HTTPS only
- Origin Protocol Policy (HTTP or S3):
  - HTTPS only
  - Or Match Viewer

  (HTTP => HTTP & HTTPS => HTTPS)
- Note:• S3 bucket “websites” don’t support HTTPS



CloudFront Signed URL / Signed Cookies

- You want to distribute paid shared content to premium users over the world
- To Restrict Viewer Access, we can create a CloudFront Signed URL / Cookie
- How long should the URL be valid for?
    - Shared content (movie, music): make it short (a few minutes)
    - Private content (private to the user): you can make it last for years
- Signed URL = access to individual files (one signed URL per file)
- Signed Cookies = access to multiple files (one signed cookie for many files)

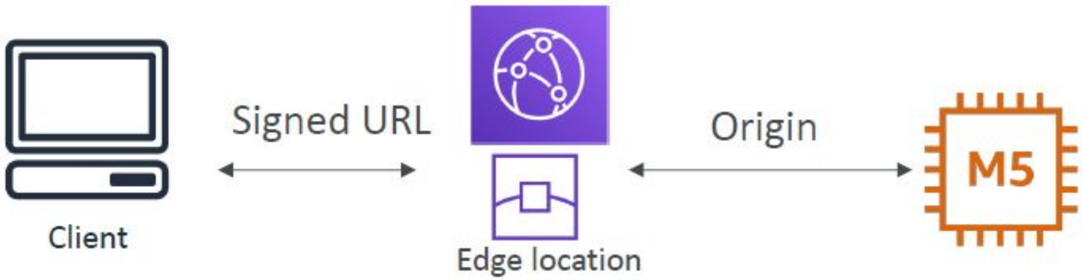


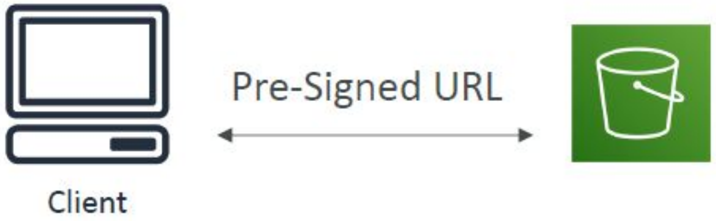
CloudFront Signed URL vs S3 Pre-Signed URL

- CloudFront Signed URL:
  - Allow access to a path, no matter the origin
  - Account wide key-pair, only the root can manage it
  - Can filter by IP, path, date, expiration
  - Can leverage caching features

S3 Pre-Signed URL:

- Issue a request as the person who pre-signed the URL
- Uses the IAM key of the signing IAM principal
- Limited lifetime





RDS, Aurora & ElastiCache

AWS RDS Overview

- RDS stands for Relational Database Service
- It’s a managed DB service for DB use SQL as a query language.
- It allows you to create databases in the cloud that are managed by AWS
    - Postgres
    - MySQL
    - MariaDB
    - Oracle
    - Microsoft SQL Server
    - Aurora (AWS Proprietary database)

Advantage over using RDS versus deploying
DB on EC2

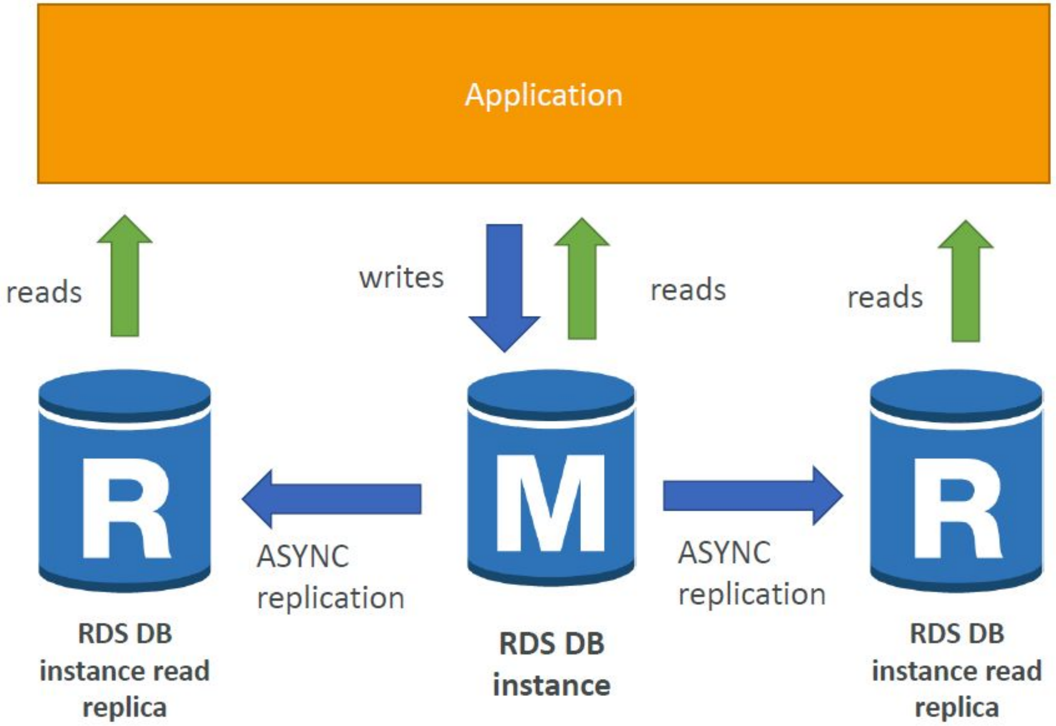
- RDS is a managed service:
- Automated provisioning, OS patching
- Continuous backups and restore to specific timestamp (Point in Time Restore)!
- Monitoring dashboards
- Read replicas for improved read performance
- Multi AZ setup for DR (Disaster Recovery)
- Maintenance windows for upgrades
- Scaling capability (vertical and horizontal)
- Storage backed by EBS (gp2 or io1)
- BUT you can’t SSH into your instances

RDS Backups

- Backups are automatically enabled in RDS
- Automated backups:
    - Daily full backup of the database (during the maintenance window)
    - Transaction logs are backed-up by RDS every 5 minutes
    - => ability to restore to any point in time (from oldest backup to 5 minutes ago)
    - 7 days retention (can be increased to 35 days)
- DB Snapshots:
    - Manually triggered by the user
    - Retention of backup for as long as you want

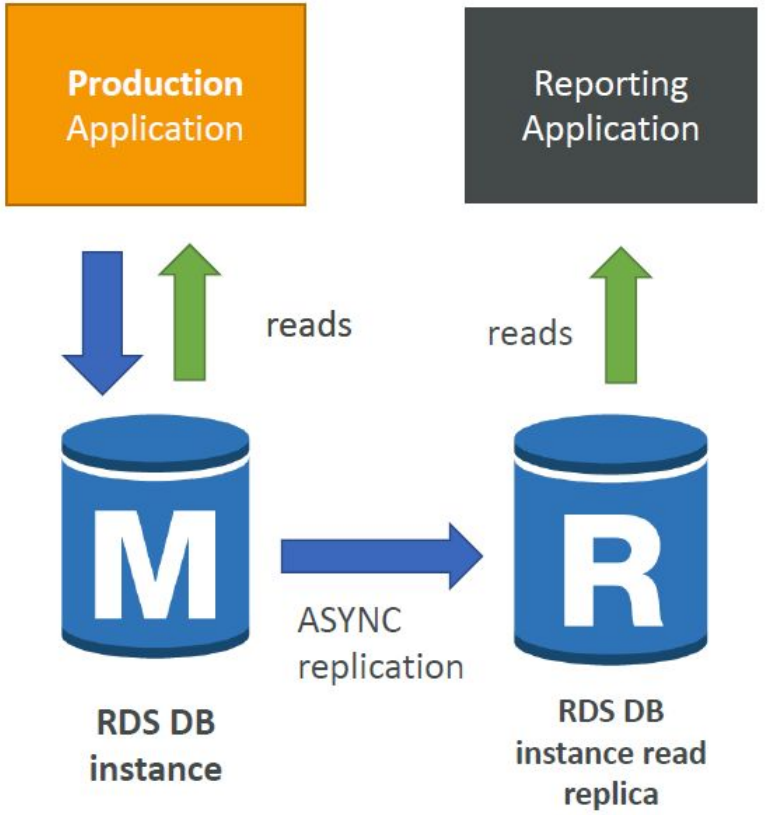
RDS Read Replicas for read scalability

- Up to 5 Read Replicas
- Within AZ, Cross AZ or Cross Region
- Replication is **ASYNC**, so reads are eventually consistent
- Replicas can be promoted to their own DB
- Applications must update the connection string to leverage read replicas



RDS Read Replicas – Use Cases

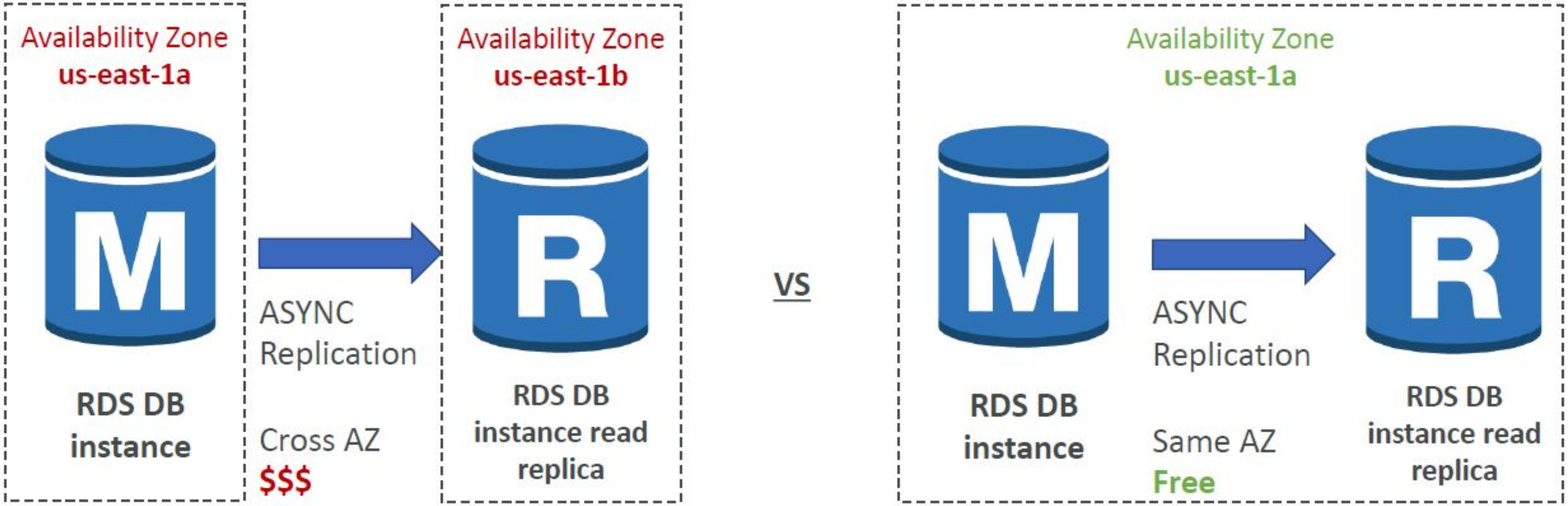
- You have a production database that is taking on normal load
- You want to run a reporting application to run some analytics
- You create a Read Replica to run the new workload there
- The production application is unaffected
- Read replicas are used for SELECT (=read) only kind of statements (not INSERT, UPDATE, DELETE)



RDS Read Replicas – Network Cost

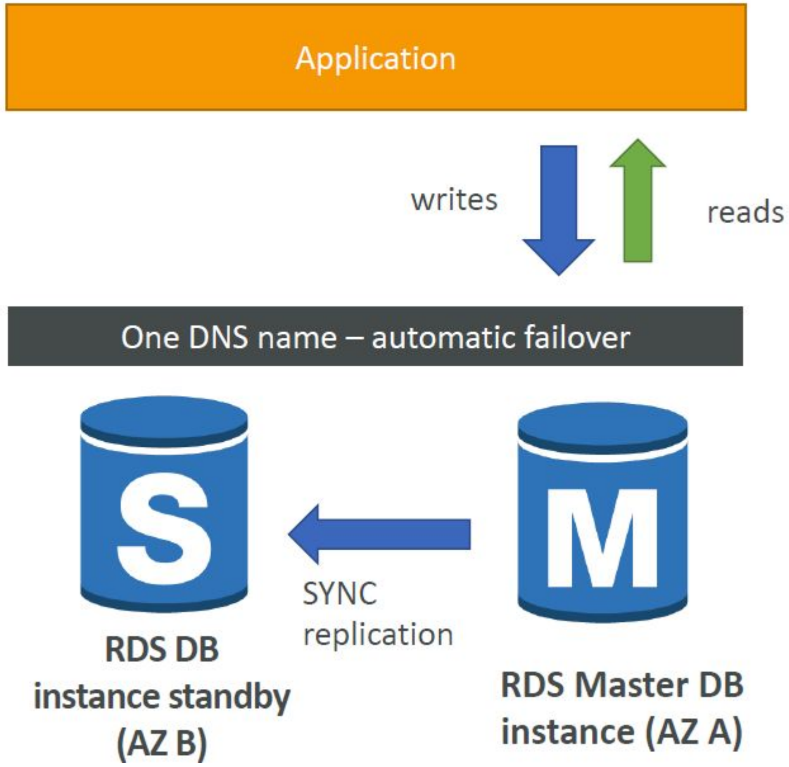
In AWS there's a network cost when data goes from one AZ to another

- To reduce the cost, you can have your Read Replicas in the same AZ



RDS Multi AZ (Disaster Recovery)

- SYNC replication
- One DNS name – automatic app failover to standby
- Increase availability
- Failover in case of loss of AZ, loss of network, instance or storage failure
- No manual intervention in apps
- Not used for scaling
- **Note: The Read Replicas be setup as Multi AZ for Disaster Recovery (DR)**


Application
writes
reads
One DNS name – automatic failover
S
M
SYNC replication
RDS DB instance standby (AZ B)
RDS Master DB instance (AZ A)

RDS Security - Encryption

- At rest encryption
  - Possibility to encrypt the master & read replicas with AWS KMS - AES-256 encryption
  - Encryption has to be defined at launch time
  - If the master is not encrypted, the read replicas cannot be encrypted
  - Transparent Data Encryption (TDE) available for Oracle and SQL Server
- In-flight encryption
  - SSL certificates to encrypt data to RDS in flight
  - Provide SSL options with trust certificate when connecting to database
- To enforce SSL:
  - PostgreSQL: rds.force_ssl=1 in the AWS RDS Console (Parameter Groups)
  - MySQL: Within the DB:

```
GRANT USAGE ON *.* TO 'mysqluser'@'%' REQUIRE SSL;
```

RDS Encryption Operations

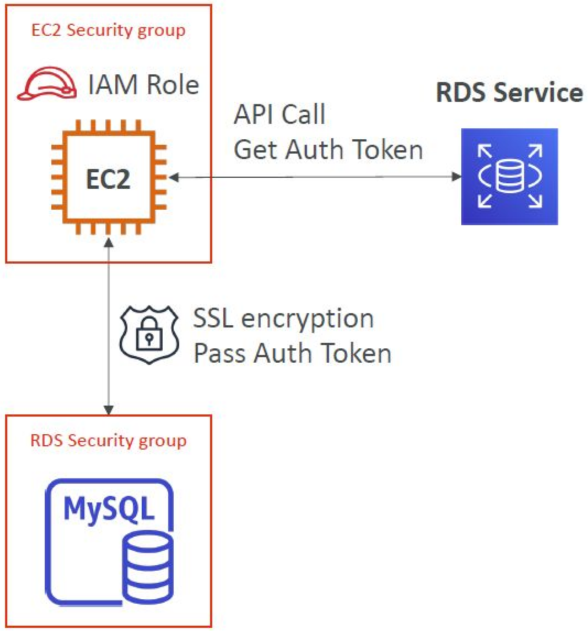
- **Encrypting RDS backups**
    - Snapshots of un-encrypted RDS databases are un-encrypted
    - Snapshots of encrypted RDS databases are encrypted
    - Can copy a snapshot into an encrypted one
- **To encrypt an un-encrypted RDS database:**
    - Create a snapshot of the un-encrypted database
    - Copy the snapshot and enable encryption for the snapshot
    - Restore the database from the encrypted snapshot
    - Migrate applications to the new database, and delete the old database

RDS Security – Network & IAM

- Network Security
    - RDS databases are usually deployed within a private subnet, not in a public one
    - RDS security works by leveraging security groups (the same concept as for EC2 instances) – it controls which IP / security group can communicate with RDS
- Access Management
    - IAM policies help control who can manage AWS RDS (through the RDS API)
    - Traditional Username and Password can be used to login into the database
    - IAM-based authentication can be used to login into RDS MySQL & PostgreSQL

RDS - IAM Authentication

- IAM database authentication works with MySQL and PostgreSQL
- You don’t need a password, just an authentication token obtained through IAM & RDS API calls
- Auth token has a lifetime of 15 minutes
- Benefits:
  - Network in/out must be encrypted using SSL
  - IAM to centrally manage users instead of DB
  - Can leverage IAM Roles and EC2 Instance profiles for easy integration



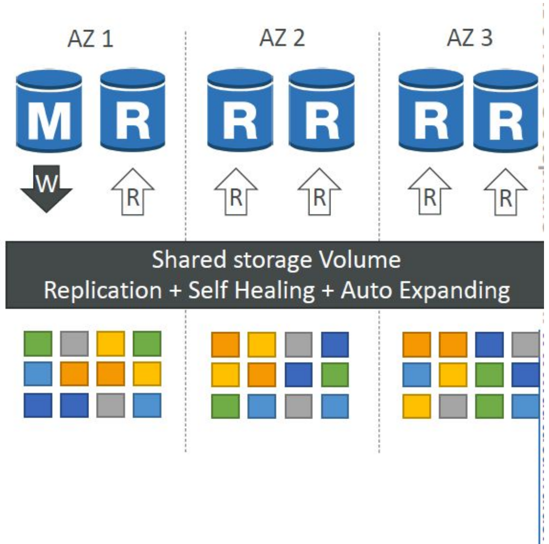
Amazon Aurora

- Aurora is a proprietary technology from AWS (not open sourced)
- Postgres and MySQL are both supported as Aurora DB (that means your drivers will work as if Aurora was a Postgres or MySQL database)
- **Aurora is “AWS cloud optimized” and claims 5x performance improvement over MySQL on RDS, over 3x the performance of Postgres on RDS**
- Aurora storage automatically grows in increments of 10GB, up to 64 TB.
- **Aurora can have 15 replicas while MySQL has 5, and the replication process is faster (sub 10 ms replica lag)**
- Failover in Aurora is instantaneous. It’s HA (High Availability) native.
- Aurora costs more than RDS (20% more) – but is more efficient

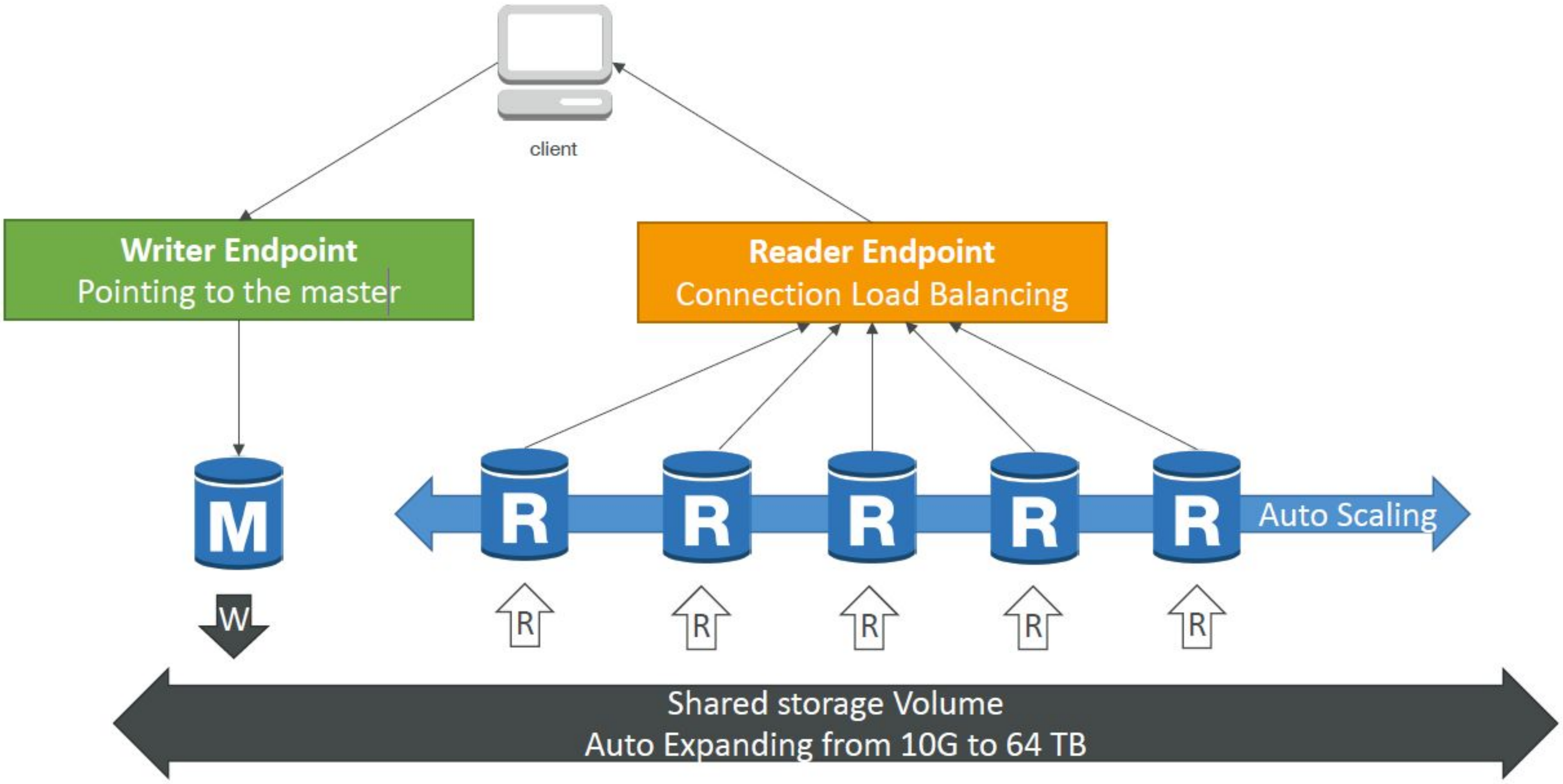
Aurora High Availability and Read Scaling

- 6 copies of your data across 3 AZ:
- Self healing with peer-to-peer replication
- Storage is striped across 100s of volumes
- One Aurora Instance takes writes (master)
- Automated failover for master in less than 30 seconds
- Master + up to 15 Aurora Read Replicas serve reads
- Support for Cross Region Replication.

Amazon Aurora features a distributed, fault-tolerant, self-healing storage system that auto-scales up to 128TB per database instance.
It delivers high performance and availability with up to 15 low-latency read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across three Availability Zones (AZs).



Aurora DB Cluster



Features of Aurora

- Automatic fail-over
- Backup and Recovery
- Isolation and security
- Industry compliance
- Push-button scaling
- Automated Patching with Zero Downtime
- Advanced Monitoring
- Routine Maintenance
- Backtrack: restore data at any point of time without using backups

Aurora Security

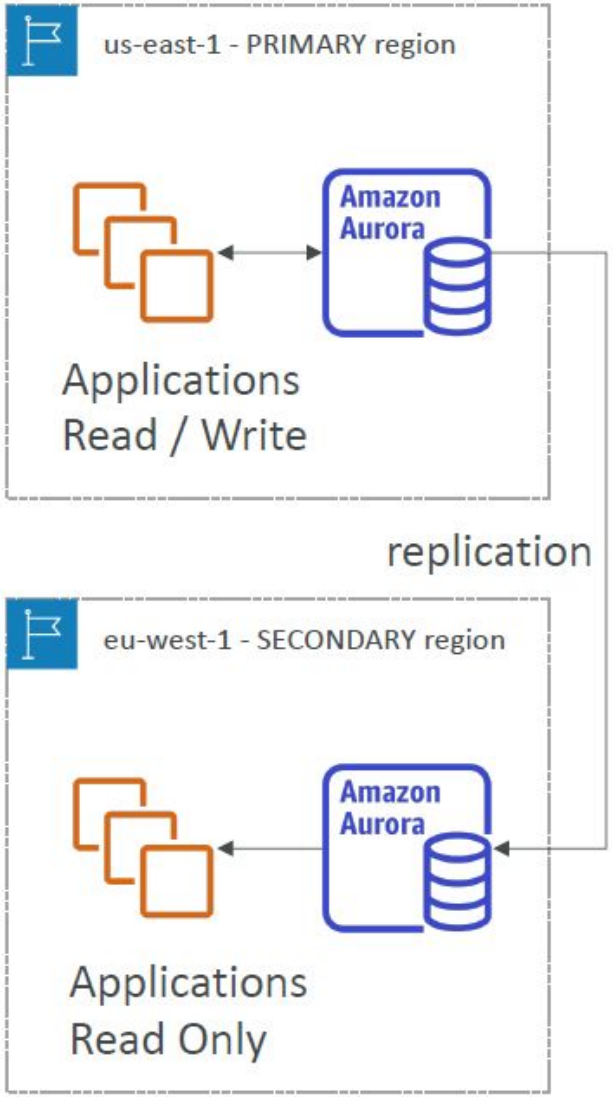
- Similar to RDS because uses the same engines
- Encryption at rest using KMS
- Automated backups, snapshots and replicas are also encrypted
- Encryption in flight using SSL (same process as MySQL or Postgres)
- Possibility to authenticate using IAM token (same method as RDS)
- You are responsible for protecting the instance with security groups
- You can’t SSH

Aurora Serverless

- Automated database Client instantiation and autoscaling based on actual usage
- Good for infrequent, intermittent or unpredictable workloads
- No capacity planning needed
- Pay per second, can be more cost-effective

Global Aurora

- Aurora Cross Region Read Replicas:
  - Useful for disaster recovery
  - Simple to put in place
- Aurora Global Database (recommended):
  - 1 Primary Region (read / write)
  - Up to 5 secondary (read-only) regions, replication lag is less than 1 second
  - Up to 16 Read Replicas per secondary region
  - Helps for decreasing latency
  - Promoting another region (for disaster recovery) has an RTO of < 1 minute

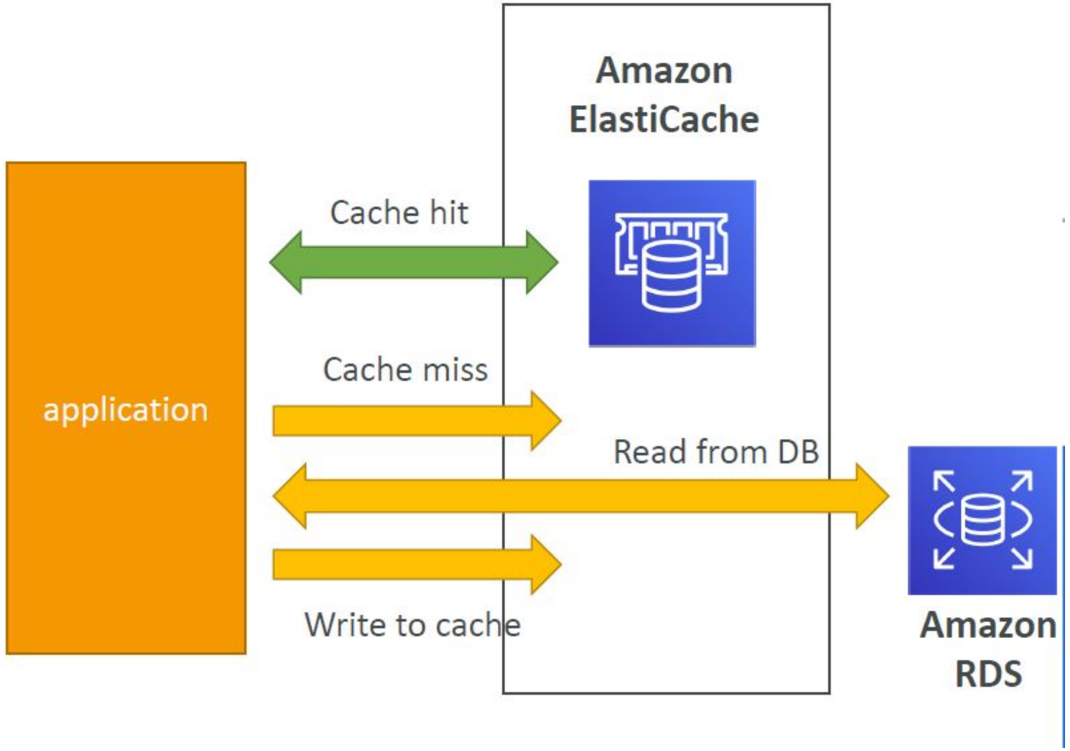

us-east-1 - PRIMARY region
Amazon Aurora
Applications Read / Write
replication
eu-west-1 - SECONDARY region
Amazon Aurora
Applications Read Only

Amazon ElastiCache Overview

- The same way RDS is to get managed Relational Databases
- ElastiCache is to get managed Redis or Memcached
- Caches are in-memory databases with really high performance, low latency
- Helps reduce load off of databases for read intensive workloads
- Helps make your application stateless
- AWS takes care of OS maintenance / patching, optimizations, setup, configuration, monitoring, failure recovery and backups
- Using ElastiCache involves heavy application code changes

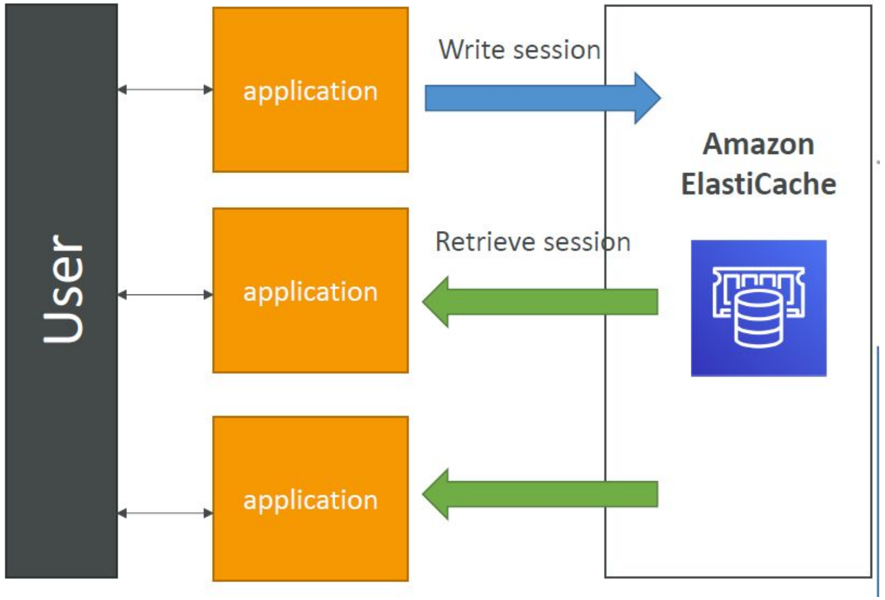
ElastiCache Solution Architecture - DB Cache

- Applications queries ElastiCache, if not available, get from RDSand store in ElastiCache.
- Helps relieve load in RDS
- Cache must have an invalidation strategy to make sure only the most current data is used in there.



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



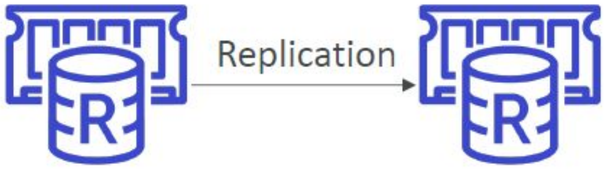
ElastiCache – Redis vs Memcached

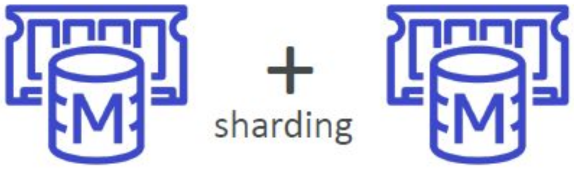
REDIS

- **Multi AZ** with Auto-Failover
- **Read Replicas** to scale reads and have **high availability**
- Data Durability using AOF persistence
- **Backup and restore features**

MEMCACHED

- Multi-node for partitioning of data (sharding)
- **Non persistent**
- **No backup and restore**
- Multi-threaded architecture

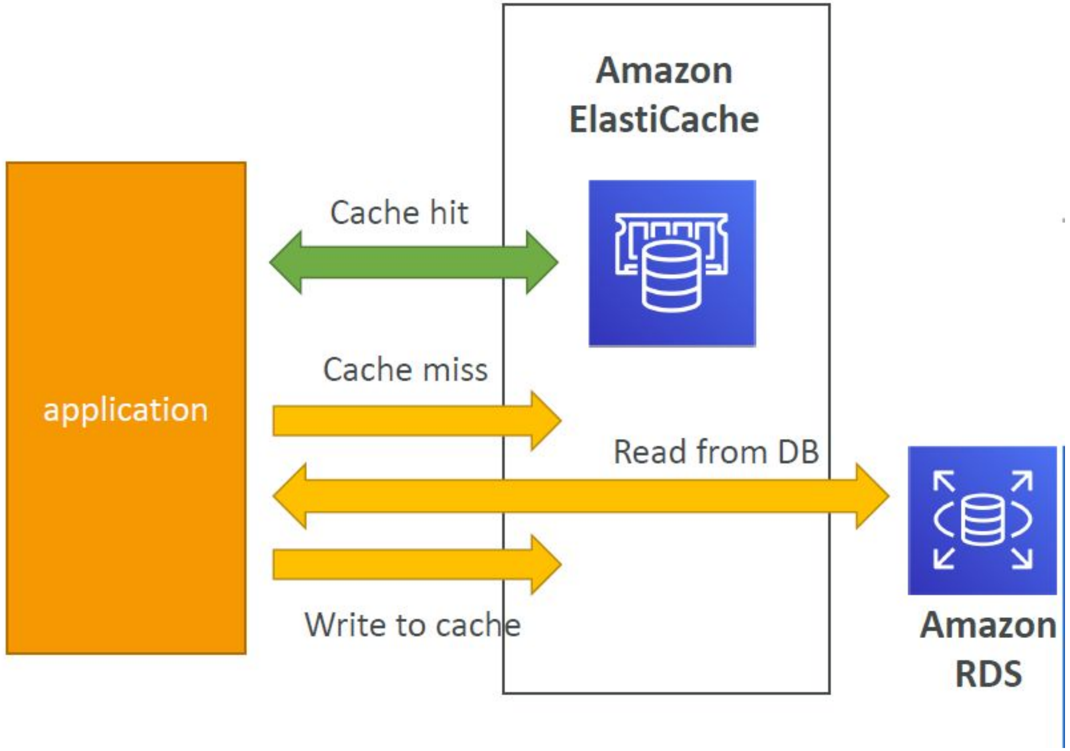




https://aws.amazon.com/caching/implementationconsiderations/

Caching Strategies

Lazy Loading / Cache-Aside / Lazy Population



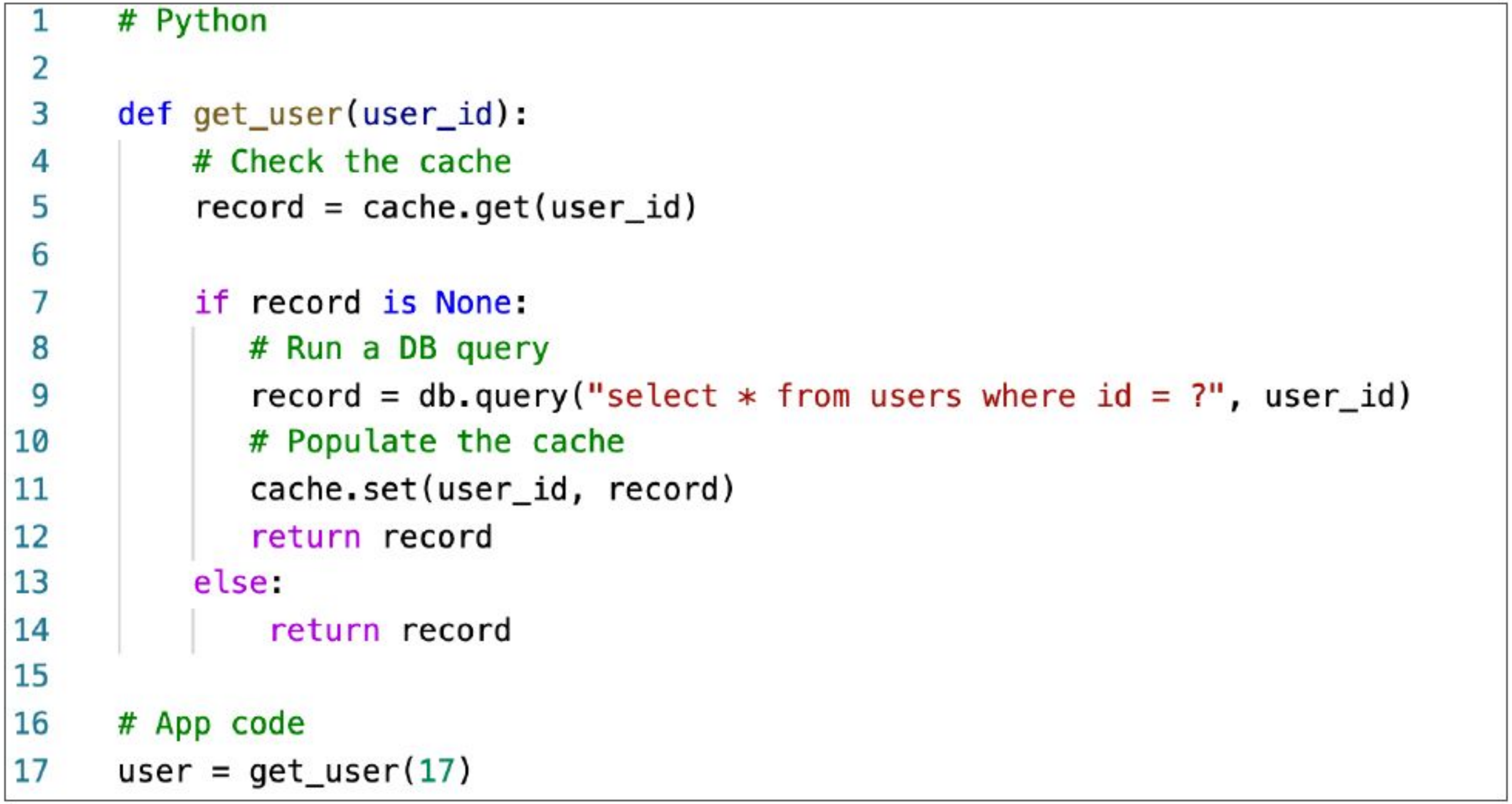
Pros

- Only requested data is cached (the cache isn’t filled up with unused data)
- Node failures are not fatal (just increased latency to warm the cache)

Cons

- Cache miss penalty that results in 3 round trips, noticeable delay for that request
- Stale data: data can be updated in the database and outdated in the cache

Python Pseudocode



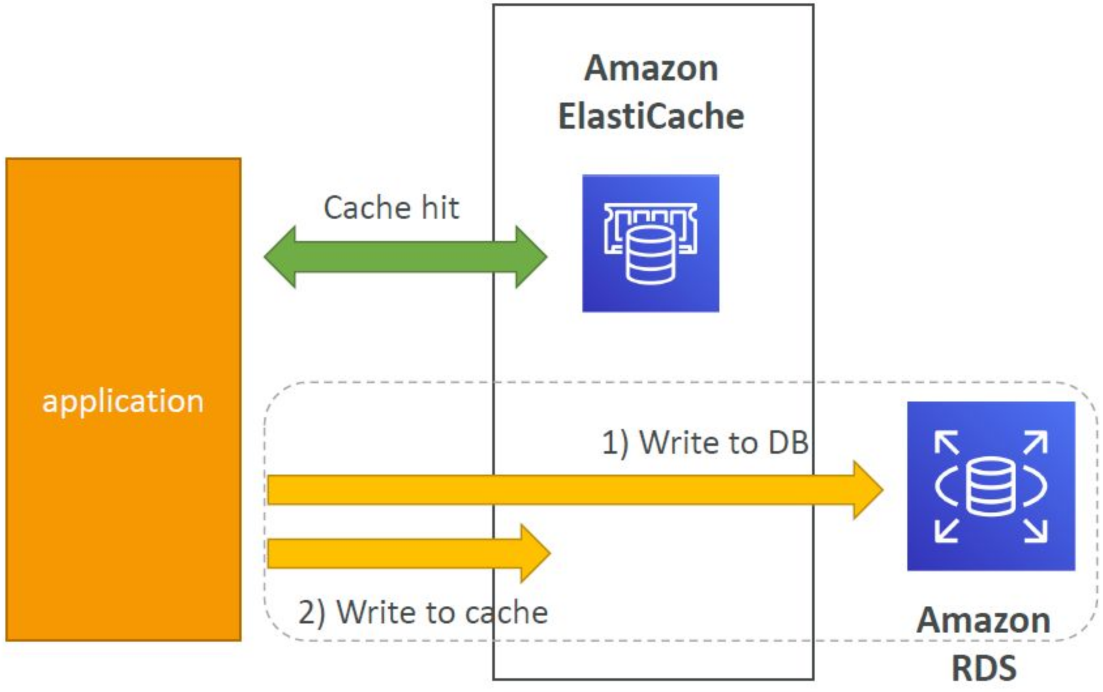
```python
# Python

def get_user(user_id):
    # Check the cache
    record = cache.get(user_id)

    if record is None:
       # Run a DB query
       record = db.query("select * from users where id = ?", user_id)
       # Populate the cache
       cache.set(user_id, record)
       return record
    else:
        return record

# App code
user = get_user(17)
```

Write Through – Add or Update cache when database is updated



Pros:

- Data in cache is never stale, reads are quick
- Write penalty vs Read penalty (each write requires 2 calls)

Cons:

- Missing Data until it is added / updated in the DB. Mitigation is to implement Lazy Loading strategy as well
- Cache churn – a lot of the data will never be read

Cache Evictions and Time-to-live (TTL)

Evictions occur when memory is over filled or greater than max memory setting in the cache, resulting into the engine to select keys to evict in order to manage its memory.

Cache eviction can occur in three ways:

  - You delete the item explicitly in the cache
  - Item is evicted because the memory is full and it’s not recently used (LRU)
  - You set an item time-to-live (or TTL)

- TTL are helpful for any kind of data:
  - Leaderboards
  - Comments
  - Activity streams
- TTL can range from few seconds to hours or days
- If too many evictions happen due to memory, you should scale up or out

DynamoDB : NoSQL Serverless Database

NoSQL databases are non-relational databases and are distributed

- NoSQL databases include MongoDB, DynamoDB, etc.

DynamoDB:

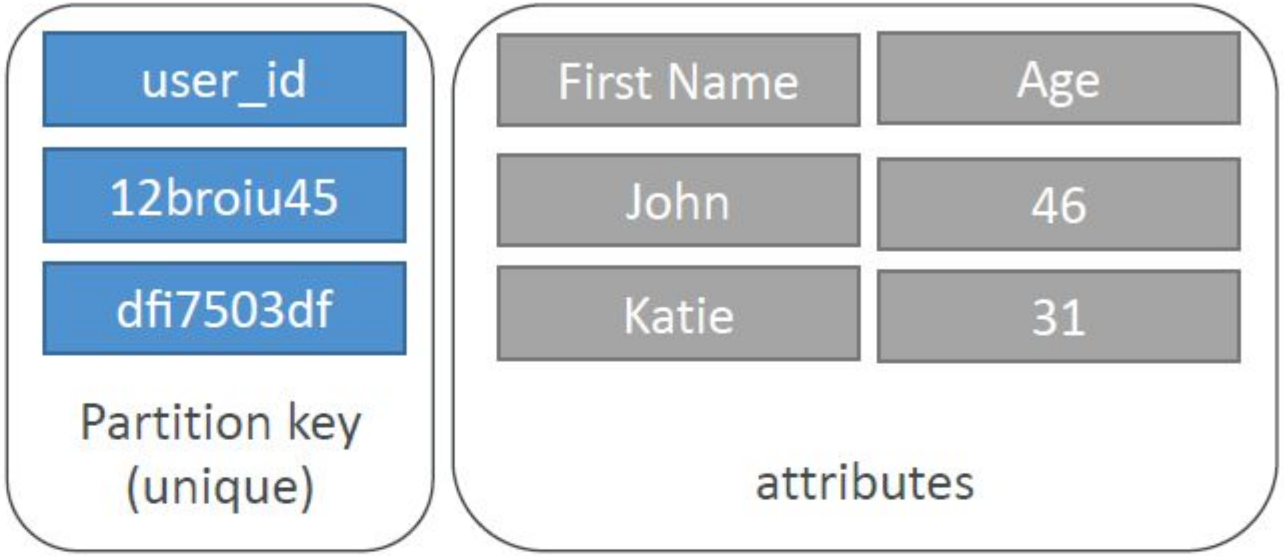
- **Fully Managed, Highly available with replication across 3 AZ**
- NoSQL database - not a relational database
- **Scales to massive workloads, distributed database**
- **Millions of requests per seconds, trillions of row, 100s of TB of storage**
- Fast and consistent in performance (low latency on retrieval)
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto scaling capabilities

DynamoDB - Basics

- DynamoDB is made of tables
- Each table has a primary key (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has attributes (can be added over time – can be null)
- Maximum size of a item is 400KB
- Data types supported are:
  - Scalar Types: String, Number, Binary, Boolean, Null
  - Document Types: List, Map
  - Set Types: String Set, Number Set, Binary Set

DynamoDB – Primary Keys

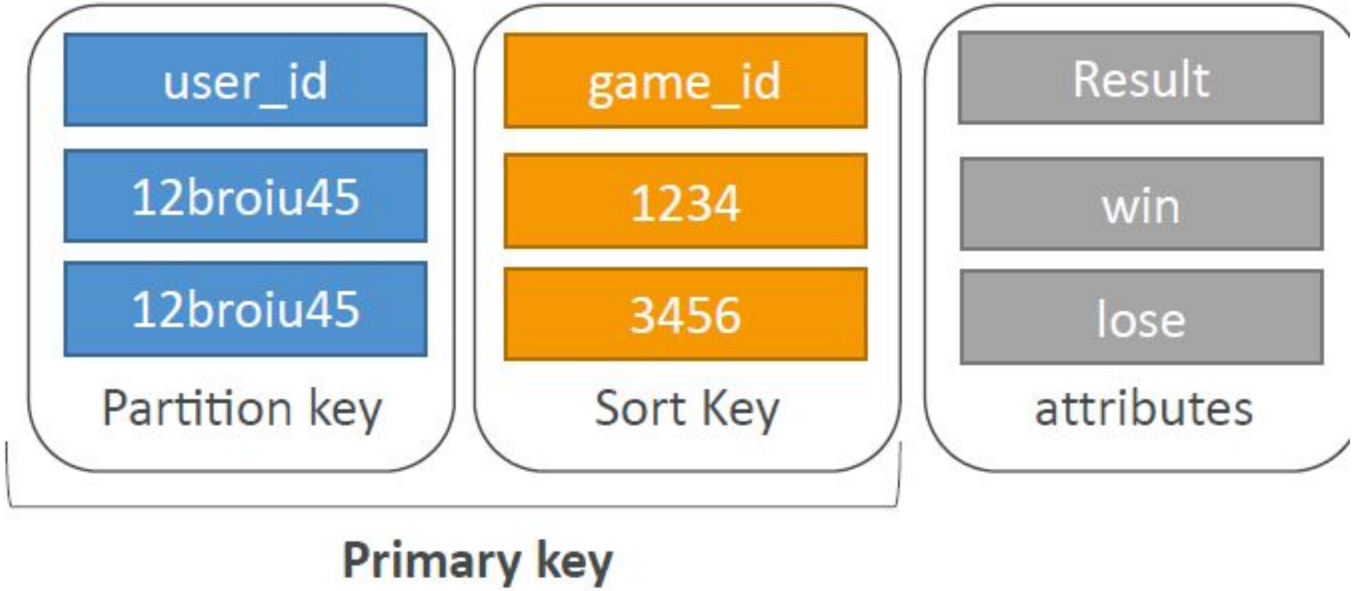
- Option 1: Partition key only (HASH)
- Partition key must be unique for each item
- Partition key must be "diverse" so that the data is distributed
- Example: user_id for a users table



user_id	First Name	Age
12broiu45	John	46
dfi7503df	Katie	31

DynamoDB – Primary Keys

- Option 2: Partition key + Sort Key
- The combination must be unique
- Data is grouped by partition key
- Sort key == range key
- Example: users-games table
    - user_id for the partition key
    - game_id for the sort key



user_id	game_id	Result
12broiu45	1234	win
12broiu45	3456	lose
Partition key	Sort Key	attributes

Primary key

DynamoDB – Partition Keys exercise

- We're building a movie database
- What is the best partition key to maximize data distribution?
- movie_id
- producer_name
- leader_actor_name
- movie_language

- movie_id has the highest cardinality so it's a good candidate
- movie_language doesn't take many values and may be skewed towards English so it's not a great partition key

DynamoDB – Provisioned Throughput

- Table must have provisioned read and write capacity units
- Read Capacity Units (RCU): throughput for reads
- Write Capacity Units (WCU): throughput for writes
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using “burst credit”
- If burst credit are empty, you’ll get a “ProvisionedThroughputException”.
- It’s then advised to do an exponential back-off retry

DynamoDB – Write Capacity Units

- One write capacity unit represents one write per second for an item up to 1 KB in size.
- If the items are larger than 1 KB, more WCU are consumed

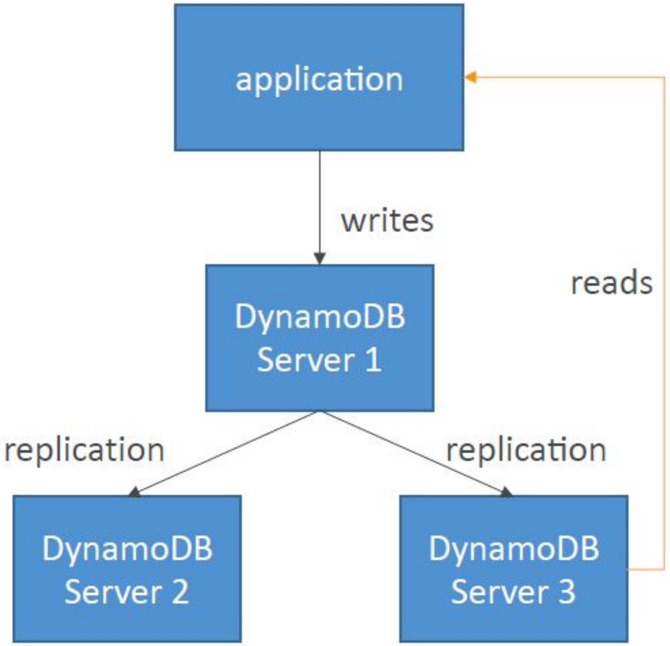
- Example 1: we write 10 objects per seconds of 2 KB each.
- We need 2 * 10 = 20 WCU

- Example 2: we write 6 objects per second of 4.5 KB each
- We need 6 * 5 = 30 WCU (4.5 gets rounded to the upper KB)

- Example 3: we write 120 objects per minute of 2 KB each
- We need 120 / 60 * 2 = 4 WCU

Strongly Consistent Read vs Eventually Consistent Read

- Eventually Consistent Read: If we read just after a write, it’s possible we’ll get unexpected response because of replication
- Strongly Consistent Read: If we read just after a write, we will get the correct data
- By default: DynamoDB uses Eventually Consistent Reads, but GetItem, Query & Scan provide a “ConsistentRead” you can set to True



DynamoDB – Read Capacity Units

- One *read capacity unit* represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size.
- If the items are larger than 4 KB, more RCU are consumed

- Example 1: 10 strongly consistent reads per seconds of 4 KB each
- We need 10 * 4 KB / 4 KB = 10 RCU

- Example 2: 16 eventually consistent reads per seconds of 12 KB each
- We need (16 / 2) * (12 / 4) = 24 RCU

- Example 3: 10 strongly consistent reads per seconds of 6 KB each
- We need 10 * 8 KB / 4 = 20 RCU (we have to round up 6 KB to 8 KB)

DynamoDB - Throttling

- If we exceed our RCU or WCU, we get ProvisionedThroughputExceededExceptions
- Reasons:
  - Hot keys: one partition key is being read too many times (popular item for ex)
  - Hot partitions:
  - Very large items: remember RCU and WCU depends on size of items
- Solutions:
  - Exponential back-off when exception is encountered (already in SDK)
  - Distribute partition keys as much as possible
  - If RCU issue, we can use DynamoDB Accelerator (DAX)

DynamoDB – Writing Data

- PutItem - Write data to DynamoDB (create data or full replace)
  - Consumes WCU
- UpdateItem – Update data in DynamoDB (partial update of attributes)
  - Possibility to use Atomic Counters and increase them
- Conditional Writes:
  - Accept a write / update only if conditions are respected, otherwise reject
  - Helps with concurrent access to items
  - No performance impact

DynamoDB – Deleting Data

- DeleteItem
  - Delete an individual row
  - Ability to perform a conditional delete
- DeleteTable
  - Delete a whole table and all its items
  - Much quicker deletion than calling DeleteItem on all items

DynamoDB – Batching Writes

- BatchWriteItem
  - Up to 25 PutItem and / or DeleteItem in one call
  - Up to 16 MB of data written
  - Up to 400 KB of data per item
- Batching allows you to save in latency by reducing the number of API calls done against DynamoDB
- Operations are done in parallel for better efficiency
- It's possible for part of a batch to fail, in which case we have the try the failed items (using exponential back-off algorithm)

DynamoDB – Reading Data

- GetItem:
  - Read based on Primary key
  - Primary Key = HASH or HASH-RANGE
  - Eventually consistent read by default
  - Option to use strongly consistent reads (more RCU - might take longer)
  - ProjectionExpression can be specified to include only certain attributes
- BatchGetItem:
  - Up to 100 items
  - Up to 16 MB of data
  - Items are retrieved in parallel to minimize latency

DynamoDB – Query

- Query returns items based on:
  - PartitionKey value (must be = operator)
  - SortKey value (=, <, <=, >, >=, Between, Begin) – optional
  - FilterExpression to further filter (client side filtering)
- Returns:
  - Up to 1 MB of data
  - Or number of items specified in Limit
- Able to do pagination on the results
- Can query table, a local secondary index, or a global secondary index

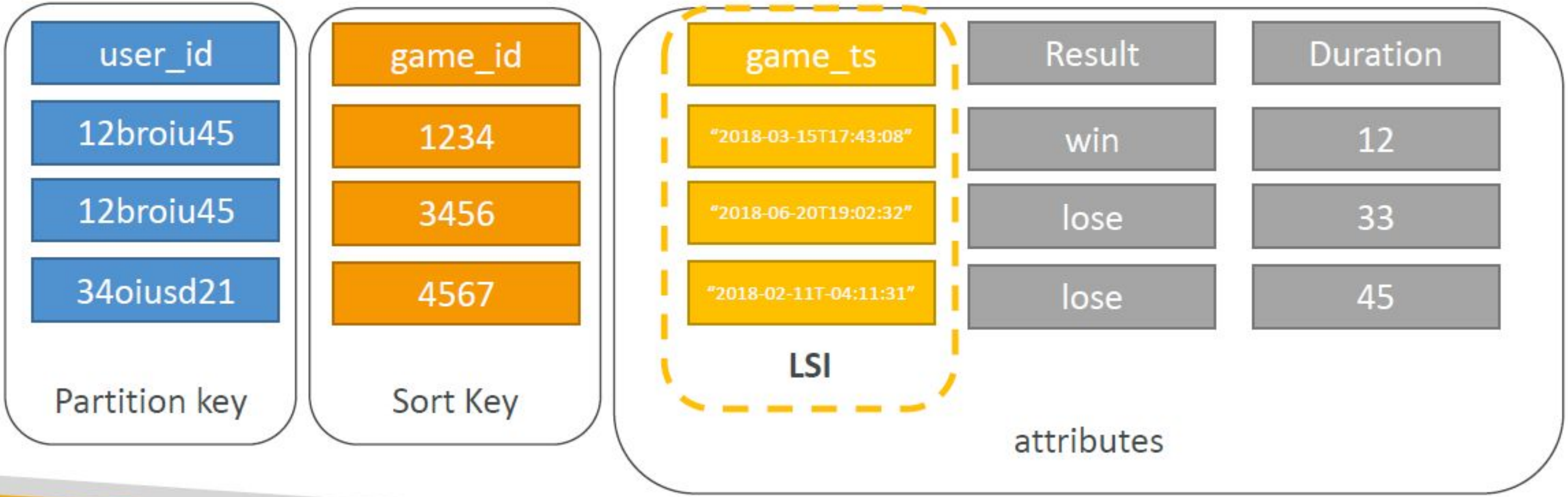
Most efficient way of querying the db

DynamoDB - Scan

- Scan the entire table and then filter out data (inefficient)
- Returns up to 1 MB of data – use pagination to keep on reading
- Consumes a lot of RCU
- Limit impact using Limit or reduce the size of the result and pause
- For faster performance, use parallel scans:
  - Multiple instances scan multiple partitions at the same time
  - Increases the throughput and RCU consumed
  - Limit the impact of parallel scans just like you would for Scans
  - Can use a ProjectionExpression + FilterExpression (no change to RCU)

DynamoDB – LSI (Local Secondary Index)

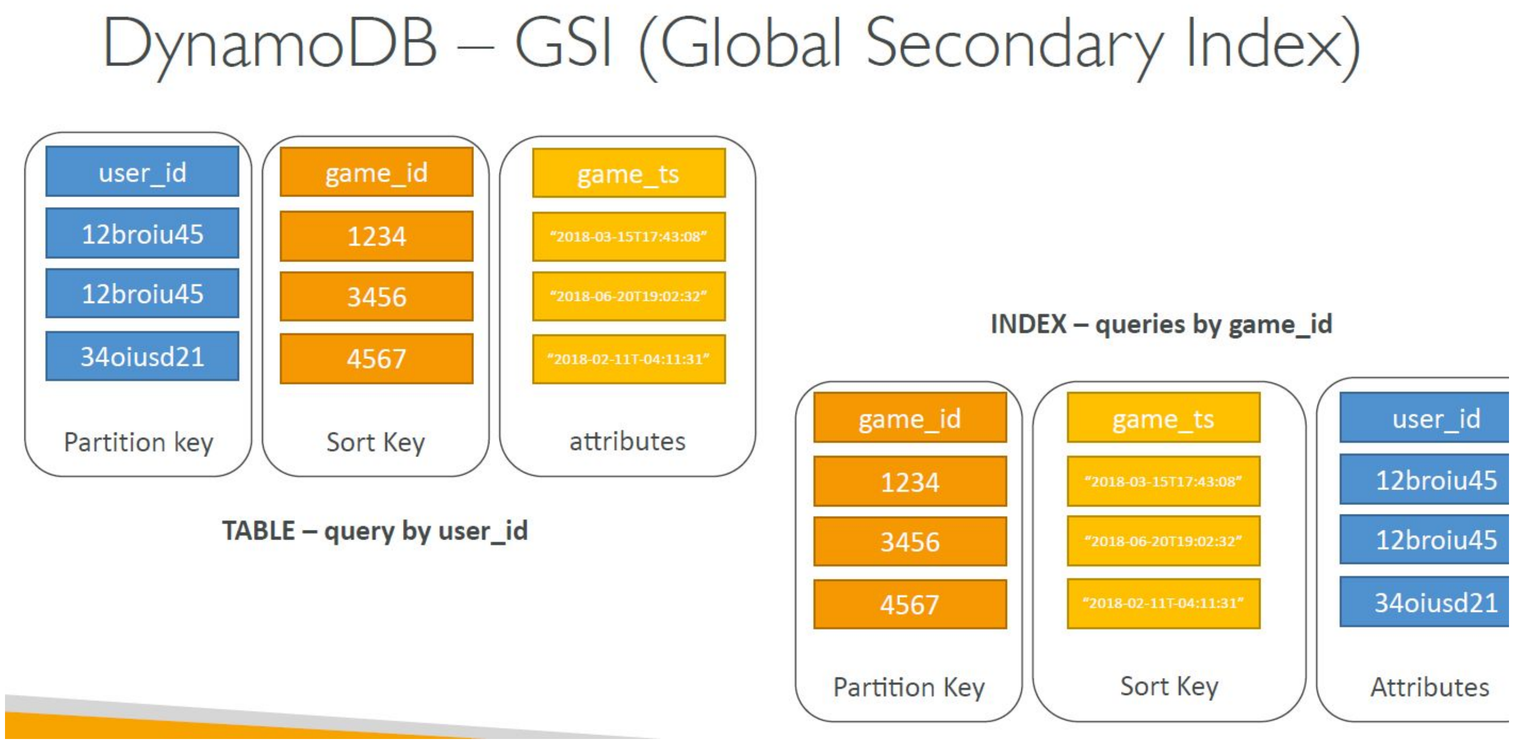
- Alternate range key for your table, local to the hash key
- **Up to five local secondary indexes per table**.
- The sort key consists of exactly one scalar attribute.
- The attribute that you choose must be a scalar String, Number, or Binary
- LSI must be defined at table creation time



user_id	game_id	game_ts	Result	Duration
12broiu45	1234	“2018-03-15T17:43:08”	win	12
12broiu45	3456	“2018-06-20T19:02:32”	lose	33
34oiusd21	4567	“2018-02-11T-04:11:31”	lose	45

DynamoDB – GSI (Global Secondary Index)

- To speed up queries on non-key attributes, use a Global Secondary Index
- GSI = partition key + optional sort key
- The index is a new “table” and we can project attributes on it
  - The partition key and sort key of the original table are always projected (KEYS_ONLY)
  - Can specify extra attributes to project (INCLUDE)
  - Can use all attributes from main table (ALL)
- Must define RCU / WCU for the index
- Possibility to add / modify GSI (not LSI)



DynamoDB – GSI (Global Secondary Index)

user_id	game_id	game_ts
12broiu45	1234	“2018-03-15T17:43:08”
12broiu45	3456	“2018-06-20T19:02:32”
34oiusd21	4567	“2018-02-11T-04:11:31”
Partition key	Sort Key	attributes

TABLE – query by user_id

INDEX – queries by game_id

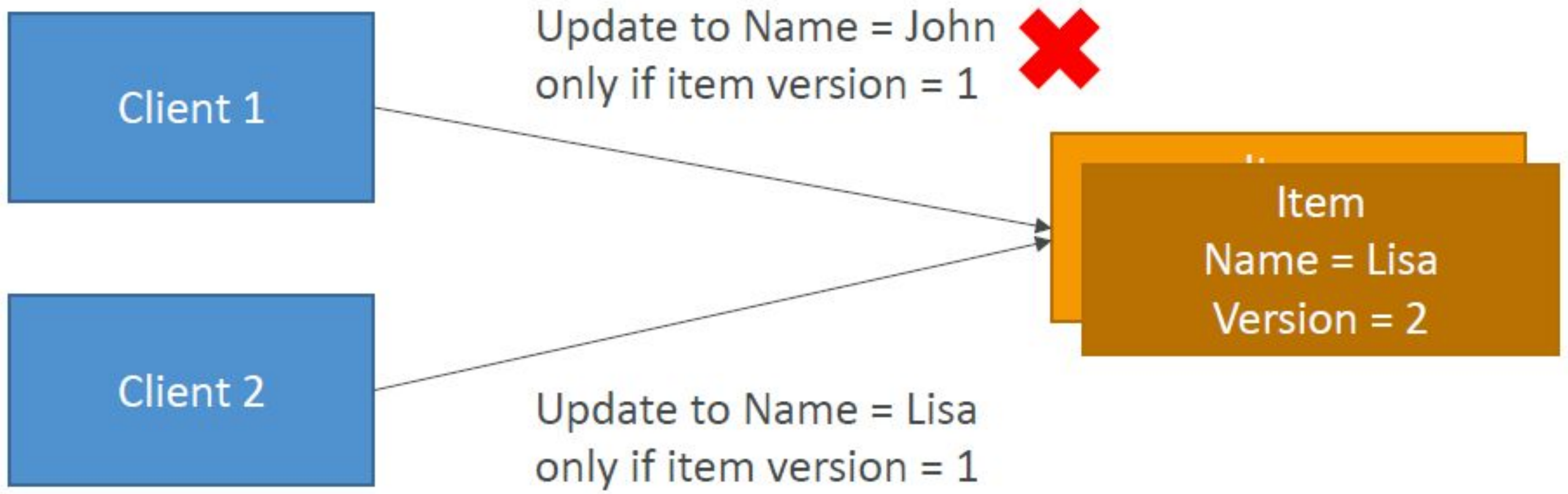
game_id	game_ts	user_id
1234	“2018-03-15T17:43:08”	12broiu45
3456	“2018-06-20T19:02:32”	12broiu45
4567	“2018-02-11T-04:11:31”	34oiusd21
Partition Key	Sort Key	Attributes

DynamoDB Indexes and Throttling

- GSI:
  - If the writes are throttled on the GSI, then the main table will be throttled!
  - Even if the WCU on the main tables are fine
  - Choose your GSI partition key carefully!
  - Assign your WCU capacity carefully!
- LSI:
  - Uses the WCU and RCU of the main table
  - No special throttling considerations

DynamoDB Concurrency

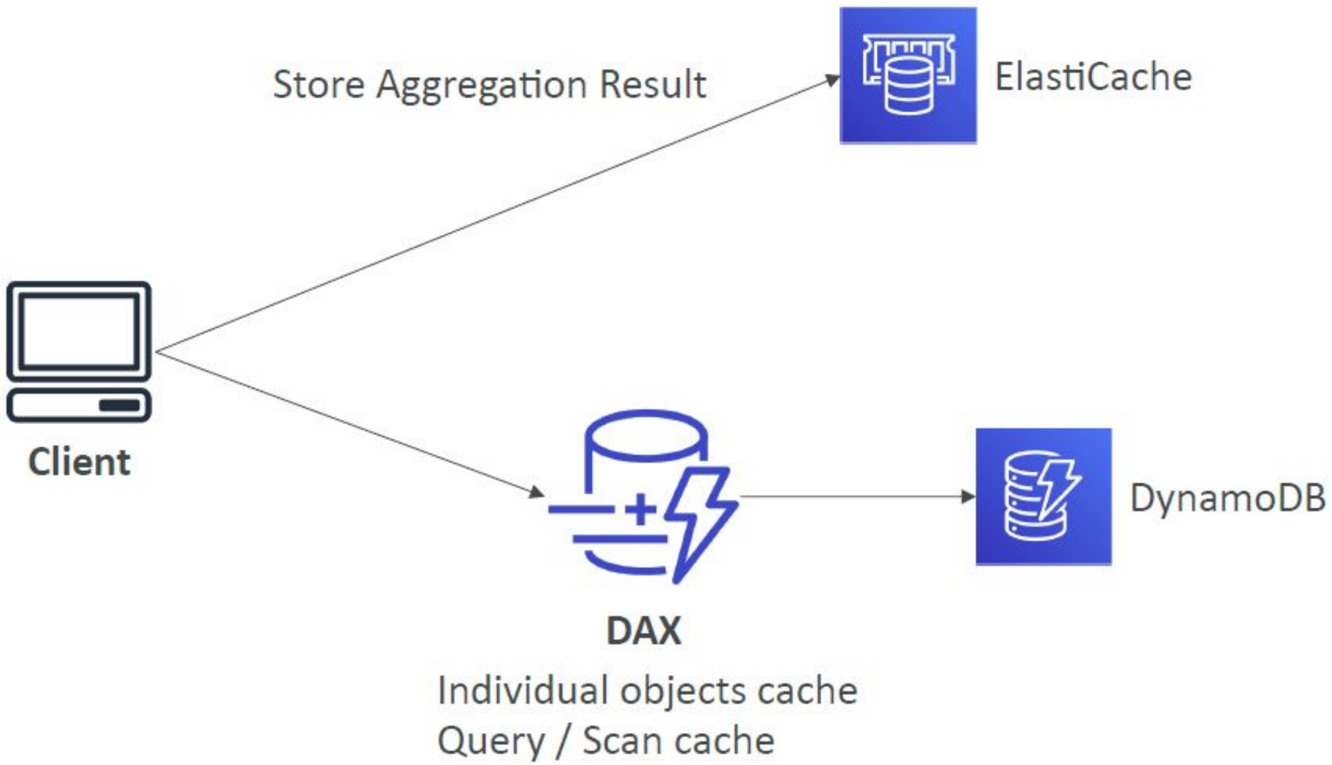
- DynamoDB has a feature called “Conditional Update / Delete”
- That means that you can ensure an item hasn’t changed before altering it
- That makes DynamoDB an optimistic locking / concurrency database



DynamoDB - DAX

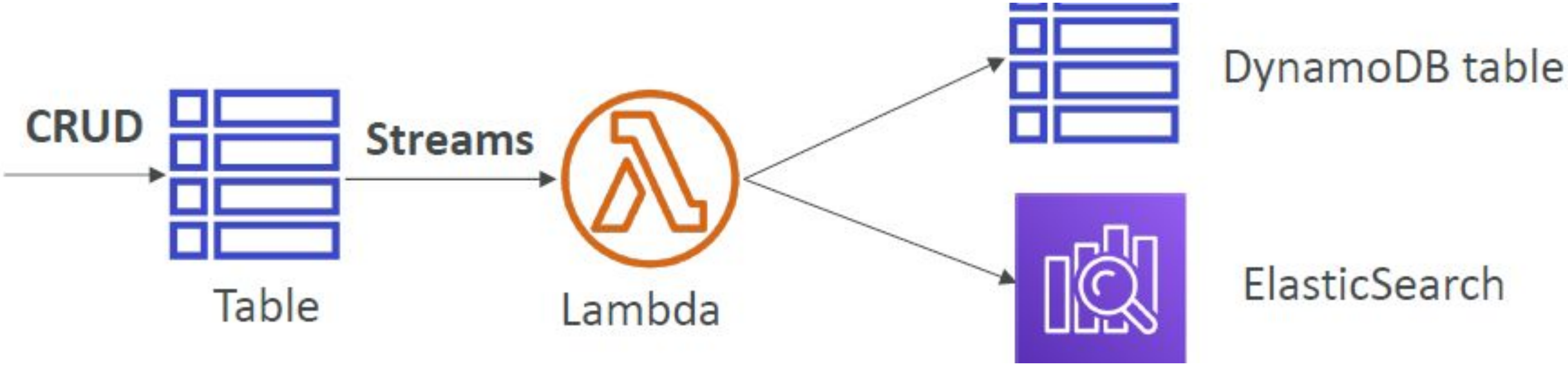
- DAX = DynamoDB Accelerator
- Seamless cache for DynamoDB, no application rewrite
- Writes go through DAX to DynamoDB
- Micro second latency for cached reads & queries
- Solves the Hot Key problem (too many reads)
- 5 minutes TTL for cache by default
- Up to 10 nodes in the cluster
- Multi AZ (3 nodes minimum recommended for production)
- Secure (Encryption at rest with KMS, VPC, IAM, CloudTrail…)

DynamoDB – DAX vs ElastiCache



DynamoDB Streams

- Changes in DynamoDB (Create, Update, Delete) can end up in a DynamoDB Stream
- This stream can be read by AWS Lambda & EC2 instances, and we can then do:
  - React to changes in real time (welcome email to new users)
  - Analytics
  - Create derivative tables / views
  - Insert into ElasticSearch
- Could implement cross region replication using Streams
- Stream has 24 hours of data retention

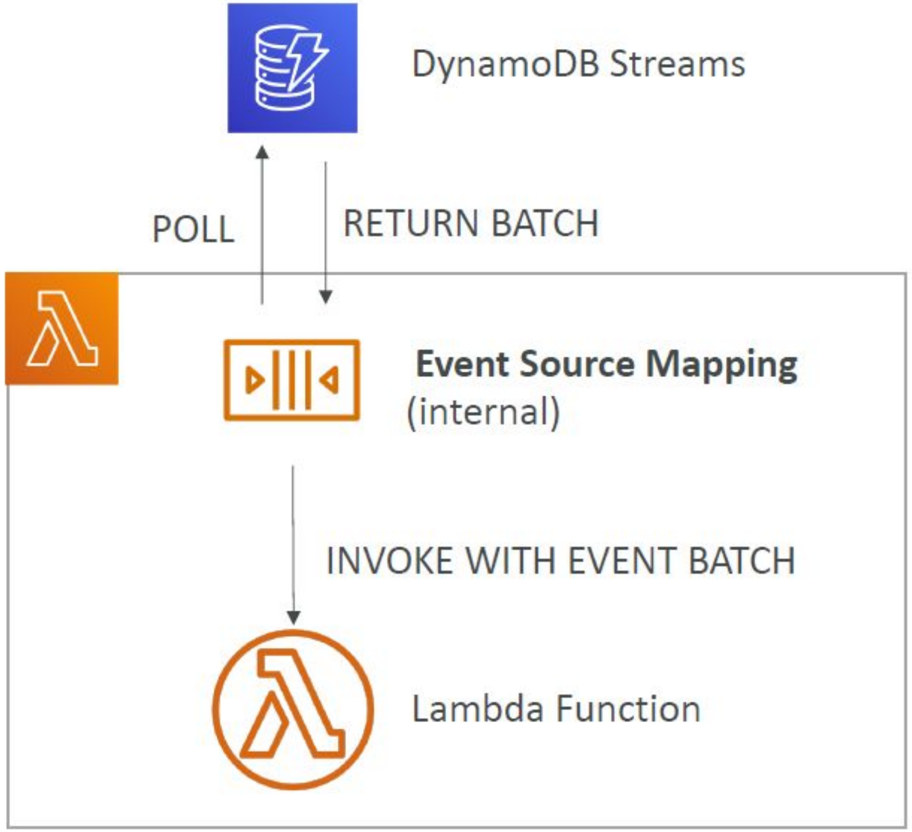


DynamoDB Streams

- Choose the information that will be written to the stream whenever the data in the table is modified:
- 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.
- DynamoDB Streams are made of shards, just like Kinesis Data Streams
- You don’t provision shards, this is automated by AWS
- Records are not retroactively populated in a stream after enabling it

DynamoDB Streams & 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



DynamoDB - TTL (Time to Live)

- TTL = automatically delete an item after an expiry date / time
- TTL is provided at no extra cost, deletions do not use WCU / RCU
- TTL is a background task operated by the DynamoDB service itself
- Helps reduce storage and manage the table size over time
- Helps adhere to regulatory norms
- TTL is enabled per row (you define a TTL column, and add a date there)
- DynamoDB typically deletes expired items within 48 hours of expiration
- Deleted items due to TTL are also deleted in GSI / LSI
- DynamoDB Streams can help recover expired items

DynamoDB CLI – Good to Know

- --projection-expression : attributes to retrieve
- --filter-expression : filter results
- General CLI pagination options including DynamoDB / S3:
- Optimization:
    - --page-size : full dataset is still received but each API call will request less data (helps avoid timeouts)
- Pagination:
    - --max-items : max number of results returned by the CLI. Returns NextToken
    - --starting-token: specify the last received NextToken to keep on reading

DynamoDB Transactions

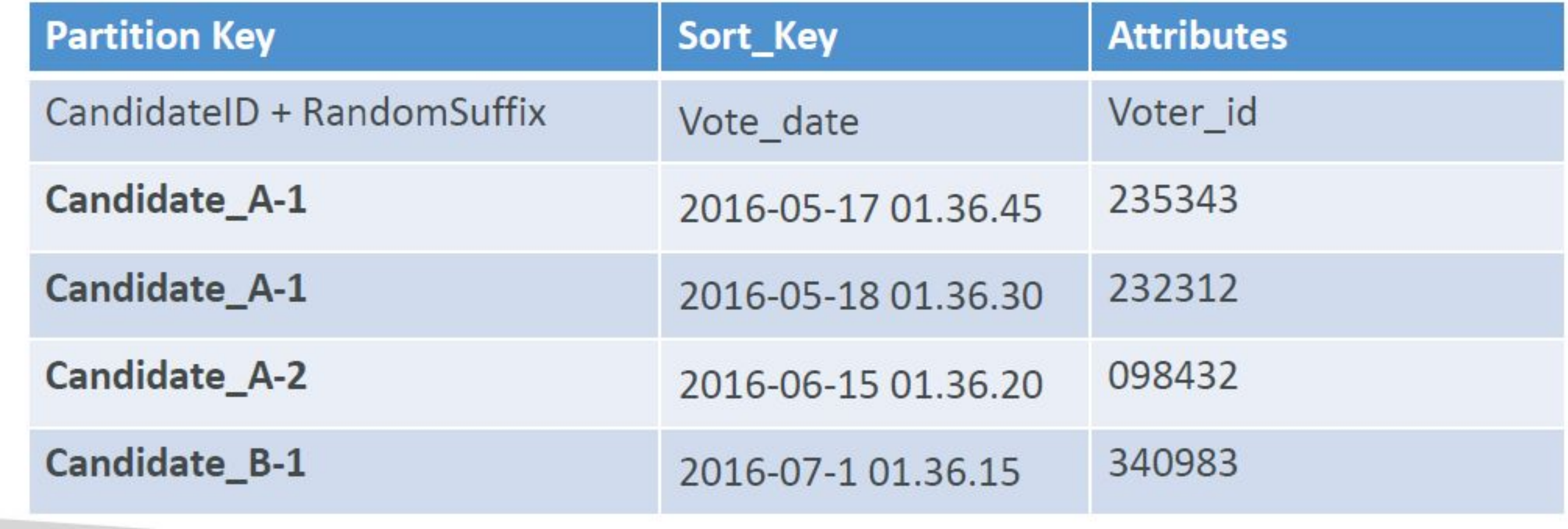
- New feature from November 2018
- Transaction = Ability to Create / Update / Delete multiple rows in different tables at the same time
- It’s an “all or nothing” type of operation.
- Write Modes: Standard, Transactional
- Read Modes: Eventual Consistency, Strong Consistency, Transactional
- Consume 2x of WCU / RCU

DynamoDB as Session State Cache

- It’s common to use DynamoDB to store session state
- vs ElastiCache:
  - ElastiCache is in-memory, but DynamoDB is serverless
  - Both are key/value stores
- vs EFS:
  - EFS must be attached to EC2 instances as a network drive
- vs EBS & Instance Store:
  - EBS & Instance Store can only be used for local caching, not shared caching
- vs S3:
  - S3 is higher latency, and not meant for small objects

DynamoDB Write Sharding

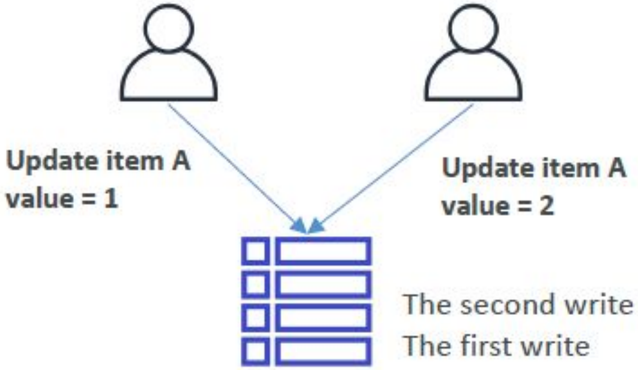
- Imagine we have a voting application with two candidates, candidate A and candidate B.
- If we use a partition key of candidate_id, we will run into partitions issues, as we only have two partitions
- Solution: add a suffix (usually random suffix, sometimes calculated suffix)



Partition Key	Sort_Key	Attributes
CandidateID + RandomSuffix	Vote_date	Voter_id
Candidate_A-1	2016-05-17 01.36.45	235343
Candidate_A-1	2016-05-18 01.36.30	232312
Candidate_A-2	2016-06-15 01.36.20	098432
Candidate_B-1	2016-07-1 01.36.15	340983

DynamoDB – Write Types

Concurrent Writes

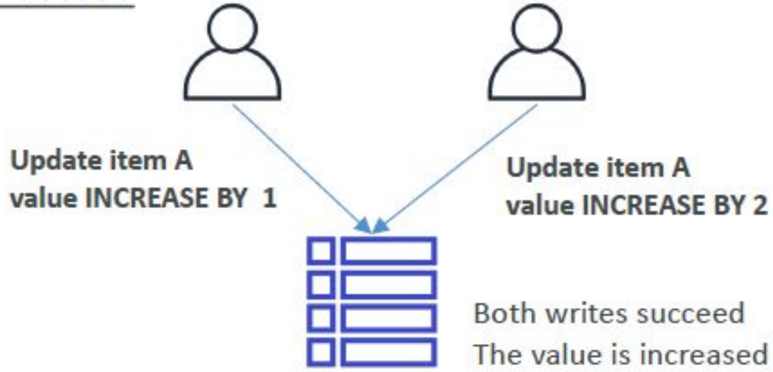


Update item A
value = 1

Update item A
value = 2

The second write overwrites
The first write

Atomic Writes

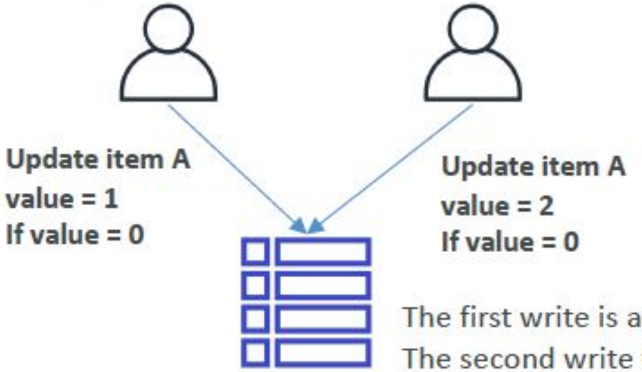


Update item A
value INCREASE BY 1

Update item A
value INCREASE BY 2

Both writes succeed
The value is increased by 3 in total

Conditional Writes



Update item A
value = 1
If value = 0

Update item A
value = 2
If value = 0

The first write is accepted
The second write fails

Batch Writes

Write / update
Many items at a time

DynamoDB Operations

- Table Cleanup:
- Option 1: Scan + Delete => very slow, expensive, consumes RCU & WCU
- Option 2: Drop Table + Recreate table => fast, cheap, efficient
- Copying a DynamoDB Table:
    - Option 1: Use AWS DataPipeline (uses EMR)
    - Option 2: Create a backup and restore the backup into a new table name (can take some time)
    - Option 3: Scan + Write => write own code

DynamoDB – Security & Other Features

- Security:
  - VPC Endpoints available to access DynamoDB without internet
  - Access fully controlled by IAM
  - Encryption at rest using KMS
  - Encryption in transit using SSL / TLS
- Backup and Restore feature available
  - Point in time restore like RDS
  - No performance impact
- Global Tables
  - Multi region, fully replicated, high performance
- Amazon DMS can be used to migrate to DynamoDB (from Mongo, Oracle, MySQL, S3, etc…)
- You can launch a local DynamoDB on your computer for development purposes

CloudFormation

Infrastructure as Code

- Currently, we have been doing a lot of manual work
- All this manual work will be very tough to reproduce:
    - In another region
    - in another AWS account
    - Within the same region if everything was deleted
- Wouldn’t it be great, if all our infrastructure was… code?
- That code would be deployed and create / update / delete our infrastructure

What is CloudFormation

- CloudFormation is a declarative way of outlining your AWS Infrastructure, for any resources (most of them are supported).
- For example, within a CloudFormation template, you say:
- I want a security group
- I want two EC2 machines using this security group
- I want two Elastic IPs for these EC2 machines
- I want an S3 bucket
- I want a load balancer (ELB) in front of these machines
- Then CloudFormation creates those for you, in the right order, with the exact configuration that you specify

Benefits of AWS CloudFormation (1/2)

- Infrastructure as code
  - No resources are manually created, which is excellent for control
  - The code can be version controlled for example using git
  - Changes to the infrastructure are reviewed through code
- Cost
  - Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
  - You can estimate the costs of your resources using the CloudFormation template
  - Savings strategy: In Dev, you could automation deletion of templates at 5 PM and recreated at 8 AM, safely
- Productivity
  - Ability to destroy and re-create an infrastructure on the cloud on the fly
  - Automated generation of Diagram for your templates!
  - Declarative programming (no need to figure out ordering and orchestration)
- Separation of concern: create many stacks for many apps, and many layers. Ex:
  - VPC stacks
  - Network stacks
  - App stacks

How CloudFormation Works

- Templates have to be uploaded in S3 and then referenced in CloudFormation
- To update a template, we can’t edit previous ones. We have to reupload a new version of the template to AWS
- Stacks are identified by a name
- Deleting a stack deletes every single artifact that was created by CloudFormation.

Deploying CloudFormation templates

- Manual way:
    - Editing templates in the CloudFormation Designer
    - Using the console to input parameters, etc
- Automated way:
    - Editing templates in a YAML file
    - Using the AWS CLI (Command Line Interface) to deploy the templates
    - Recommended way when you fully want to automate your flow

CloudFormation Building Blocks

Templates components (one course section for each):

1. Resources: your AWS resources declared in the template (MANDATORY)
2. Parameters: the dynamic inputs for your template
3. Mappings: the static variables for your template
4. Outputs: References to what has been created
5. Conditionals: List of conditions to perform resource creation
6. Metadata

Templates helpers:

1. References
2. Functions

What are resources?

- Resources are the core of your CloudFormation template (MANDATORY)
- They represent the different AWS Components that will be created and configured
- Resources are declared and can reference each other
- AWS figures out creation, updates and deletes of resources for us
- There are over 224 types of resources (!)
- Resource types identifiers are of the form:

    AWS::aws-product-name::data-type-name

http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html

FAQ for resources

- Can I create a dynamic amount of resources?

ØNo, you can’t. Everything in the CloudFormation template has to be declared. You can’t perform code generation there

- Is every AWS Service supported?

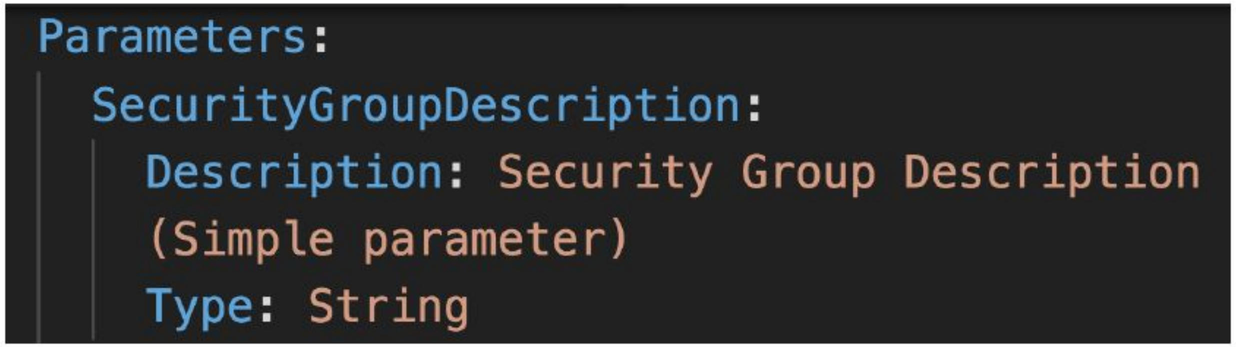
ØAlmost. Only a select few niches are not there yet
ØYou can work around that using AWS Lambda Custom Resources

What are parameters?

- Parameters are a way to provide inputs to your AWS CloudFormation template
- They’re important to know about if:
    - You want to reuse your templates across the company
    - Some inputs can not be determined ahead of time
- Parameters are extremely powerful, controlled, and can prevent errors from happening in your templates thanks to types.

When should you use a parameter?

- Ask yourself this:
    - Is this CloudFormation resource configuration likely to change in the future?
    - If so, make it a parameter.
- You won’t have to re-upload a template to change its content



```
Parameters:
  SecurityGroupDescription:
    Description: Security Group Description
    (Simple parameter)
    Type: String
```

Parameters Settings

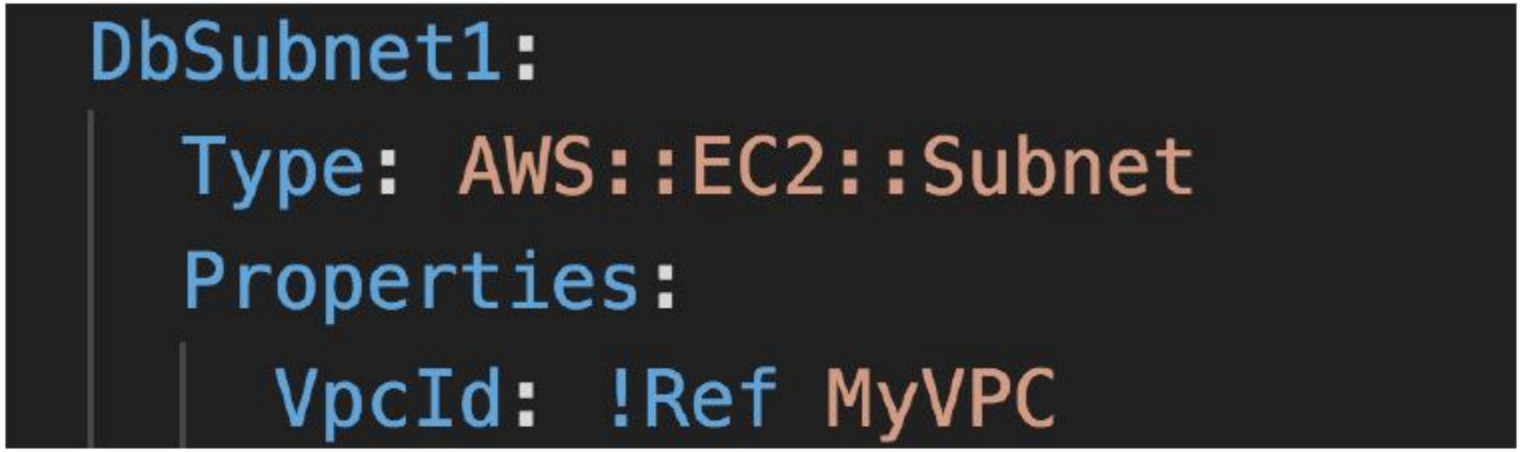
Parameters can be controlled by all these settings:

Type:

- String
- Number
- CommaDelimitedList
- List<Type>
- AWS Parameter (to help catch invalid values – match against existing values in the AWS Account)
- Description
- Constraints
- ConstraintDescription (String)
- Min/MaxLength
- Min/MaxValue
- Defaults
- AllowedValues (array)
- AllowedPattern (regexp)
- NoEcho (Boolean)

How to Reference a Parameter

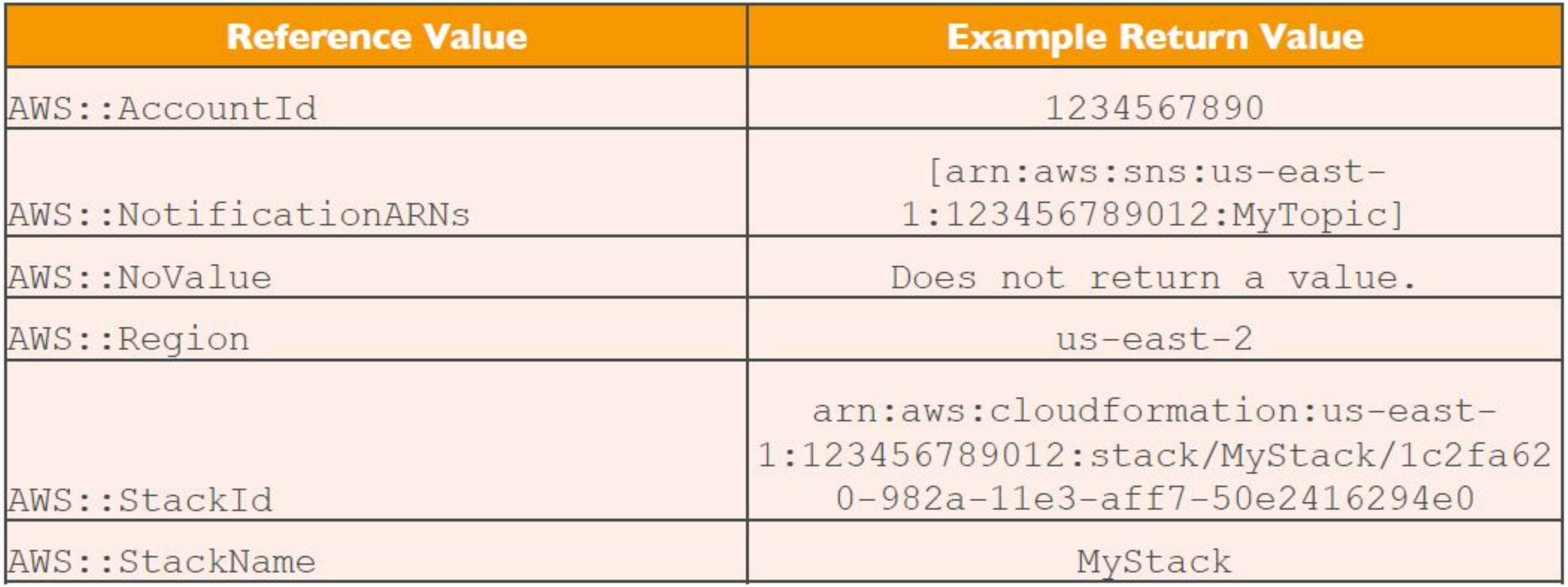
- The Fn::Ref function can be leveraged to reference parameters
- Parameters can be used anywhere in a template.
- The shorthand for this in YAML is !Ref
- The function can also reference other elements within the template



```
DbSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref MyVPC
```

Concept: Pseudo Parameters

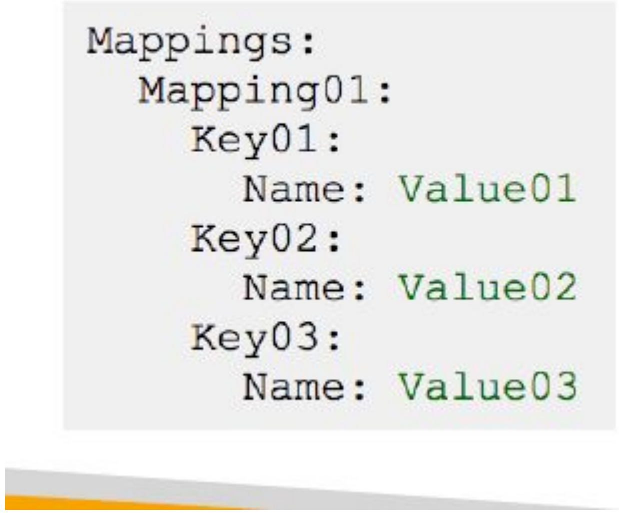
- AWS offers us pseudo parameters in any CloudFormation template.
- These can be used at any time and are enabled by default



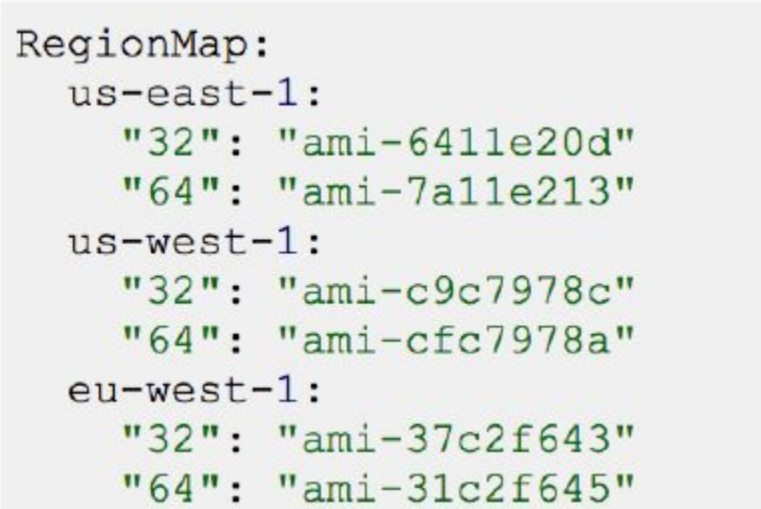
Reference Value	Example Return Value
AWS::AccountId	1234567890
AWS::NotificationARNs	[arn:aws:sns:us-east-1:123456789012:MyTopic]
AWS::NoValue	Does not return a value.
AWS::Region	us-east-2
AWS::StackId	arn:aws:cloudformation:us-east-1:123456789012:stack/MyStack/1c2fa620-982a-11e3-aff7-50e2416294e0
AWS::StackName	MyStack

What are mappings?

- Mappings are fixed variables within your CloudFormation Template.
- They’re very handy to differentiate between different environments (dev vs prod), regions (AWS regions), AMI types, etc
- All the values are hardcoded within the template
- Example:



```
Mappings:
  Mapping01:
    Key01:
      Name: Value01
    Key02:
      Name: Value02
    Key03:
      Name: Value03
```



```
RegionMap:
  us-east-1:
    "32": "ami-6411e20d"
    "64": "ami-7a11e213"
  us-west-1:
    "32": "ami-c9c7978c"
    "64": "ami-cfc7978a"
  eu-west-1:
    "32": "ami-37c2f643"
    "64": "ami-31c2f645"
```

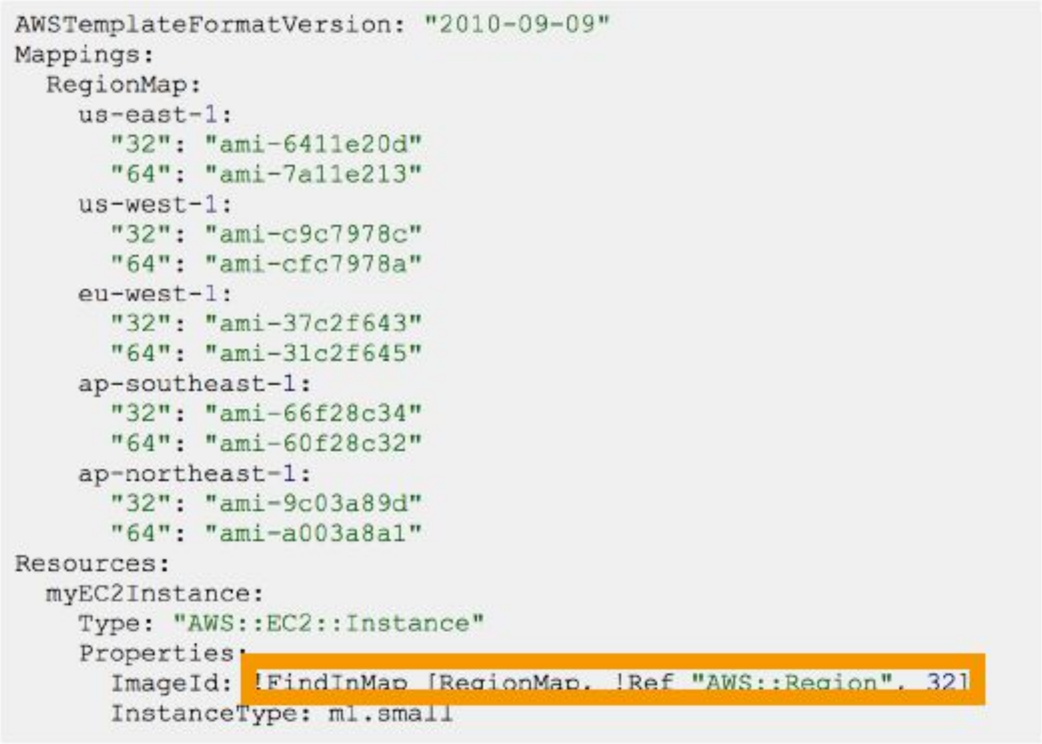
When would you use mappings vs parameters ?

- Mappings are great when you know in advance all the values that can be taken and that they can be deduced from variables such as
    - Region
    - Availability Zone
    - AWS Account
    - Environment (dev vs prod)
- They allow safer control over the template.
- Use parameters when the values are really user specific

Fn::FindInMap

Accessing Mapping Values

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



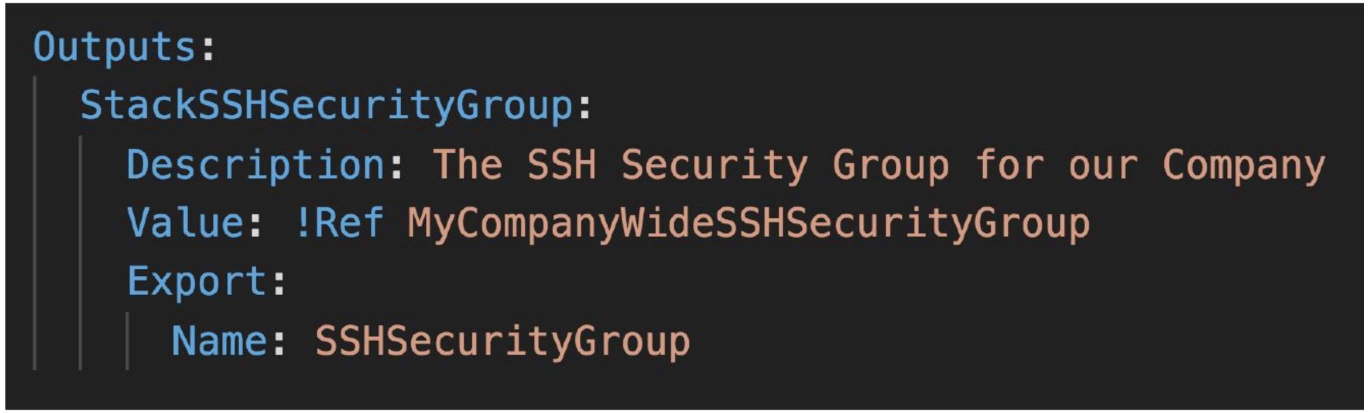
```
AWSTemplateFormatVersion: "2010-09-09"
Mappings:
  RegionMap:
    us-east-1:
      "32": "ami-6411e20d"
      "64": "ami-7a11e213"
    us-west-1:
      "32": "ami-c9c7978c"
      "64": "ami-cfc7978a"
    eu-west-1:
      "32": "ami-37c2f643"
      "64": "ami-31c2f645"
    ap-southeast-1:
      "32": "ami-66f28c34"
      "64": "ami-60f28c32"
    ap-northeast-1:
      "32": "ami-9c03a89d"
      "64": "ami-a003a8a1"
Resources:
  myEC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: !FindInMap [RegionMap, !Ref "AWS::Region", 32]
      InstanceType: m1.small
```

What are outputs?

- The Outputs section declares optional outputs values that we can import into other stacks (if you export them first)!
- You can also view the outputs in the AWS Console or in using the AWS CLI
- They’re very useful for example if you define a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
- It’s the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack
- You can’t delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack

Outputs Example

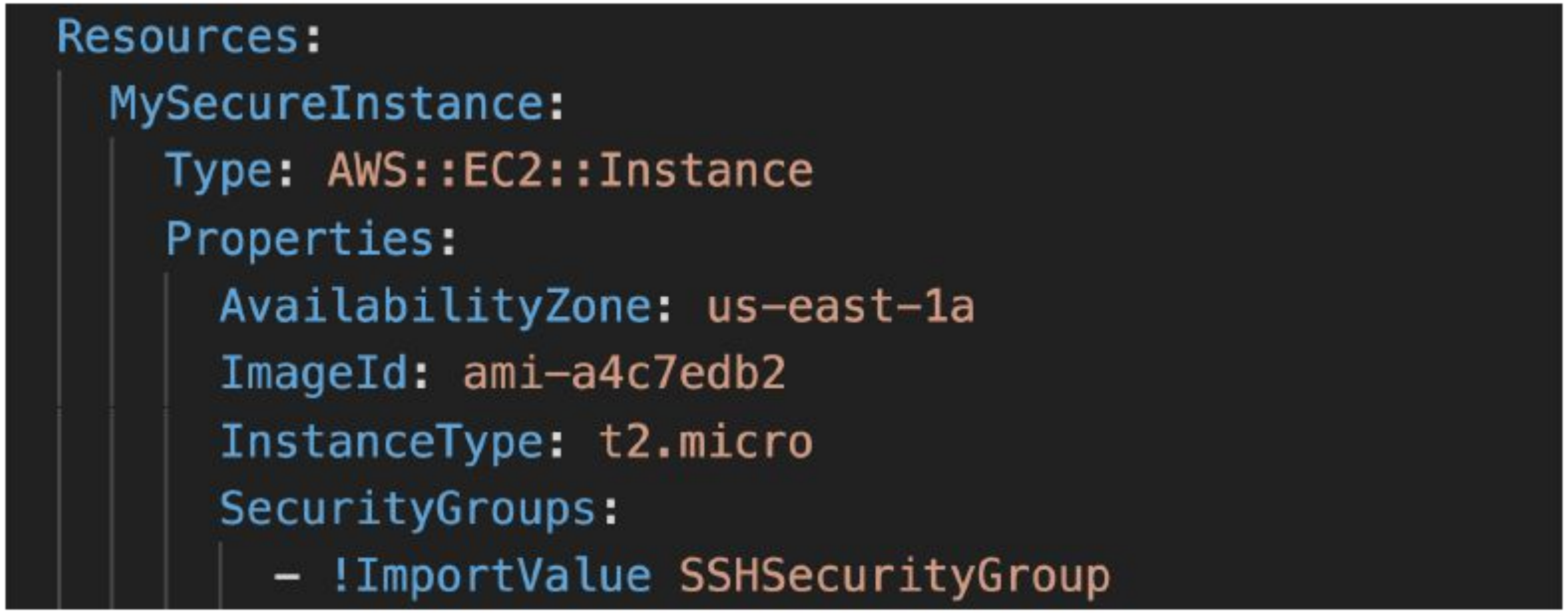
- Creating a SSH Security Group as part of one template
- We create an output that references that security group



```
Outputs:
  StackSSHSecurityGroup:
    Description: The SSH Security Group for our Company
    Value: !Ref MyCompanyWideSSHSecurityGroup
    Export:
      Name: SSHSecurityGroup
```

Cross Stack Reference

- We then create a second template that leverages that security group
- For this, we use the Fn::ImportValue function
- You can’t delete the underlying stack until all the references are deleted too.



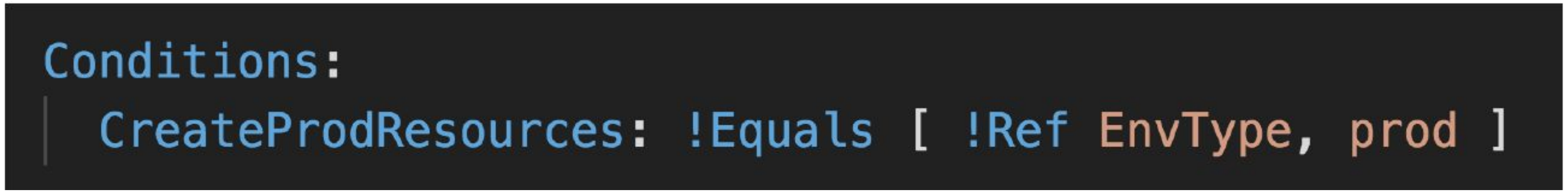
```
Resources:
  MySecureInstance:
    Type: AWS::EC2::Instance
    Properties:
      AvailabilityZone: us-east-1a
      ImageId: ami-a4c7edb2
      InstanceType: t2.micro
      SecurityGroups:
        - !ImportValue SSHSecurityGroup
```

What are conditions used for?

- Conditions are used to control the creation of resources or outputs based on a condition.
- Conditions can be whatever you want them to be, but common ones are:
    - Environment (dev / test / prod)
    - AWS Region
    - Any parameter value
- Each condition can reference another condition, parameter value or mapping

How to define a condition?

- The logical ID is for you to choose. It’s how you name condition
- The intrinsic function (logical) can be any of the following:
- Fn::And
- Fn::Equals
- Fn::If
- Fn::Not
- Fn::Or



```
Conditions:
  CreateProdResources: !Equals [ !Ref EnvType, prod ]
```

Conditions can be applied to resources / outputs / etc

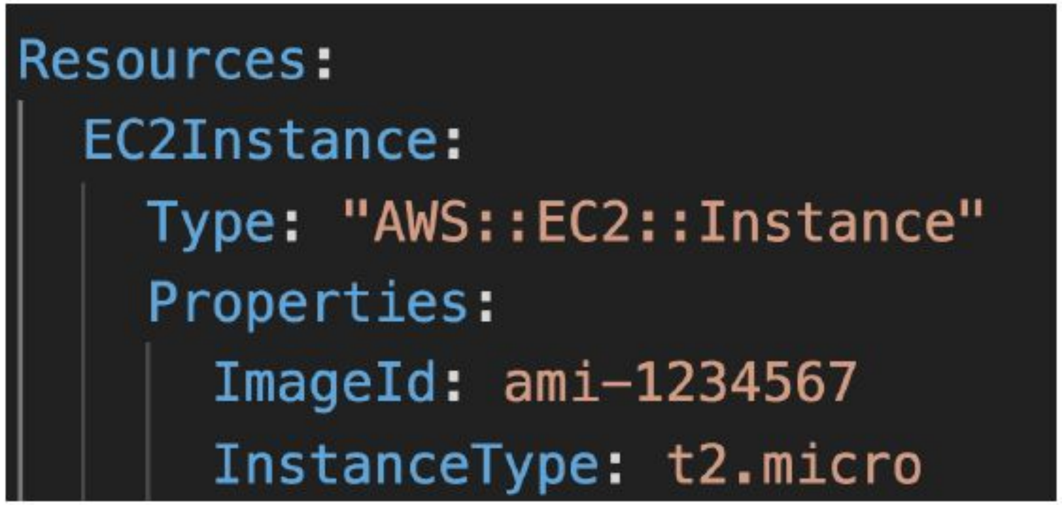
CloudFormation

Must Know Intrinsic Functions

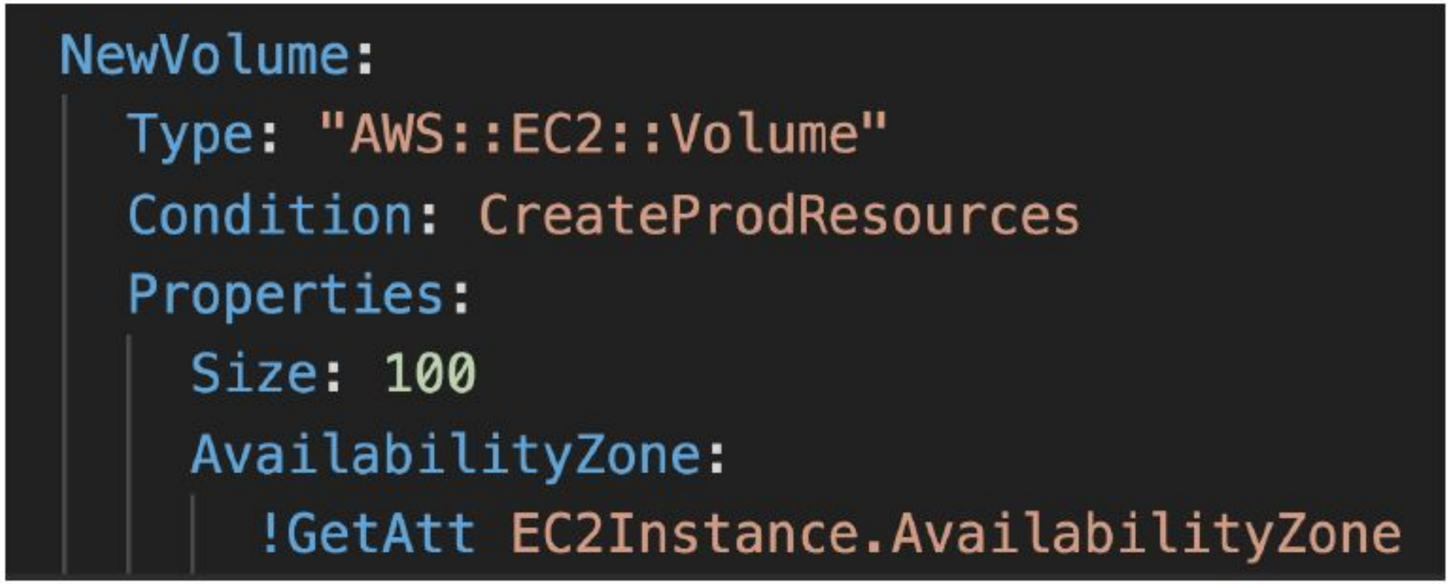
- Ref
- Fn::GetAtt
- Fn::FindInMap
- Fn::ImportValue
- Fn::Join
- Fn::Sub
- Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc…)

Fn::GetAtt

- Attributes are attached to any resources you create
- To know the attributes of your resources, the best place to look at is the documentation.
- For example: the AZ of an EC2 machine!



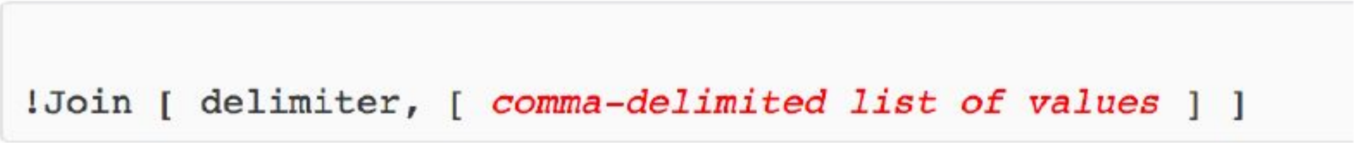
```
Resources:
  EC2Instance:
    Type: "AWS::EC2::Instance"
    Properties:
      ImageId: ami-1234567
      InstanceType: t2.micro
```



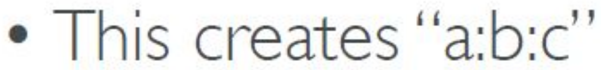
```
NewVolume:
  Type: "AWS::EC2::Volume"
  Condition: CreateProdResources
  Properties:
    Size: 100
    AvailabilityZone:
      !GetAtt EC2Instance.AvailabilityZone
```

Fn::Join

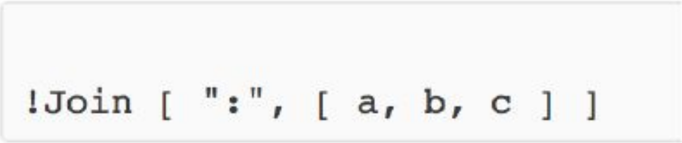
- Join values with a delimiter



```
!Join [ delimiter, [ comma-delimited list of values ] ]
```



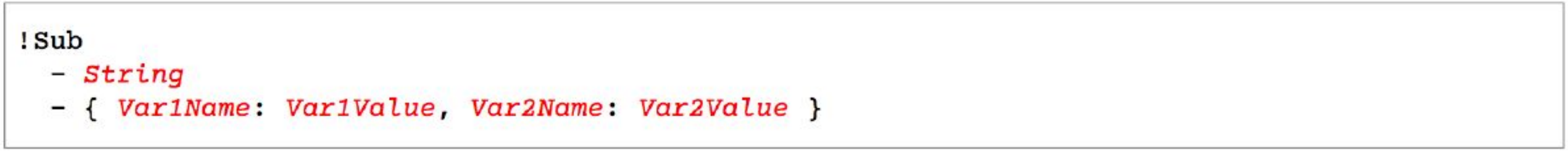
- This creates “a:b:c”



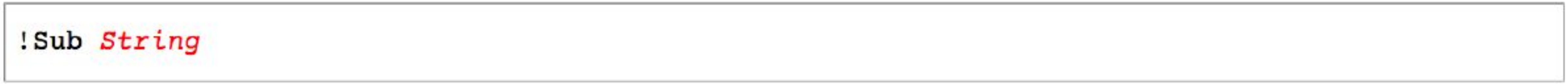
```
!Join [ ":", [ a, b, c ] ]
```

Function Fn::Sub

- Fn::Sub, or !Sub as a shorthand, is used to substitute variables from a text. It’s a very handy function that will allow you to fully customize your templates.
- For example, you can combine Fn::Sub with References or AWS Pseudo variables!
- String must contain ${VariableName} and will substitute them



```
!Sub
  - String
  - { Var1Name: Var1Value, Var2Name: Var2Value }
```



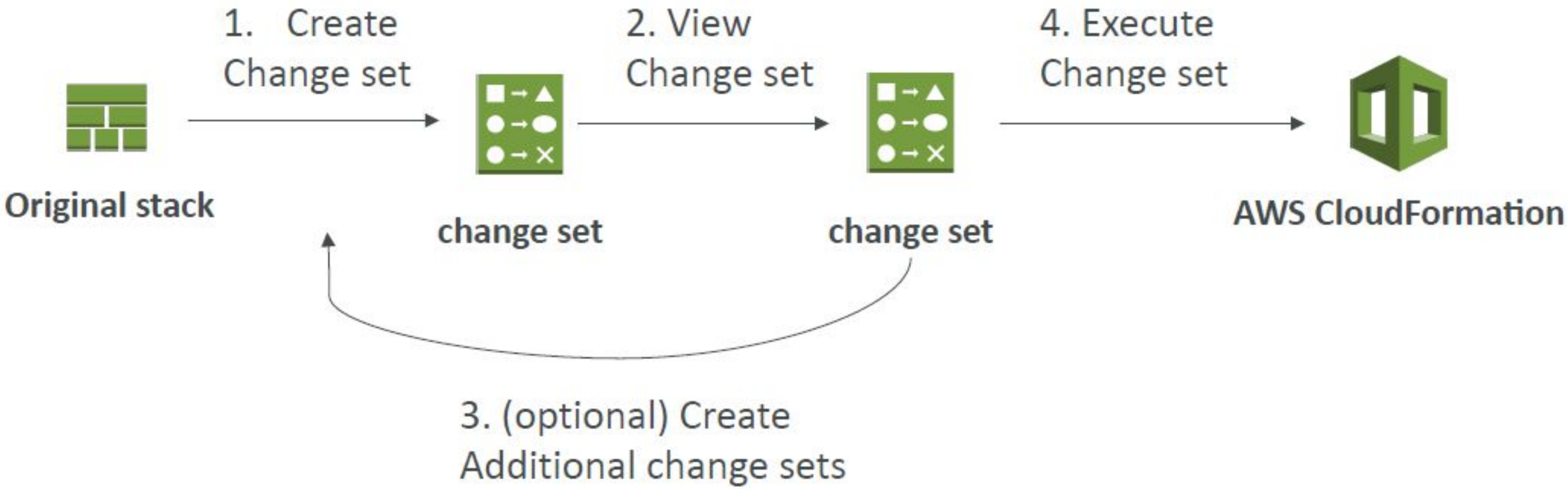
```
!Sub String
```

CloudFormation Rollbacks

- Stack Creation Fails:
  - Default: everything rolls back (gets deleted). We can look at the log
  - Option to disable rollback and troubleshoot what happened
- Stack Update Fails:
  - The stack automatically rolls back to the previous known working state
  - Ability to see in the log what happened and error messages

ChangeSets

- When you update a stack, you need to know what changes before it happens for greater confidence
- ChangeSets won’t say if the update will be successful

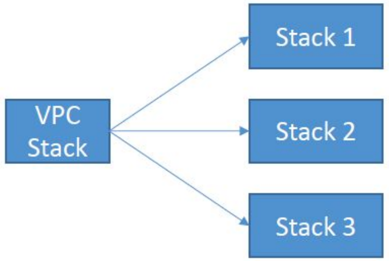


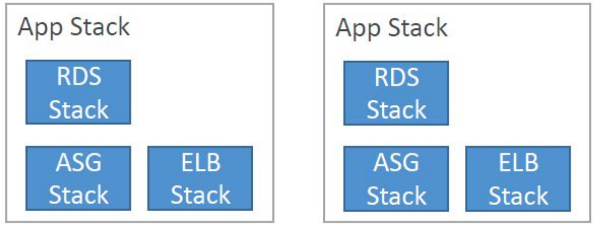
Nested stacks

- Nested stacks are stacks as part of other stacks
- They allow you to isolate repeated patterns / common components in separate stacks and call them from other stacks
- Example:
    - Load Balancer configuration that is re-used
    - Security Group that is re-used
- Nested stacks are considered best practice
- To update a nested stack, always update the parent (root stack)

CloudFormation – Cross vs Nested Stacks

- Cross Stacks
    - Helpful when stacks have different lifecycles
    - Use Outputs Export and Fn::ImportValue
    - When you need to pass export values to many stacks (VPC Id, etc…)
- Nested Stacks
    - Helpful when components must be re-used
    - Ex: re-use how to properly configure an Application Load Balancer
    - The nested stack only is important to the higher level stack (it’s not shared)





CloudFormation - StackSets

- Create, update, or delete stacks across multiple accounts and regions with a single operation
- Administrator account to create StackSets
- Trusted accounts to create, update,delete stack instances from StackSets
- When you update a stack set, all associated stack instances are updated throughout all accounts and regions.