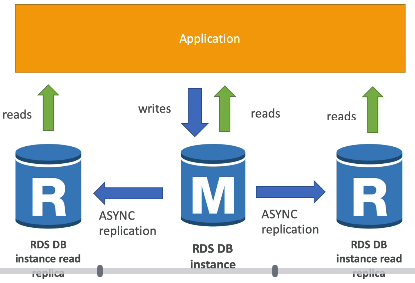
# RDS + Aurora + ElasticCache

## RDS Real Replicas v/s Multi AZ

RDS real replicas used for scalability, as main database instance can’t scale enough if received too many requests.

We can create up to max 5 read replicas that can be within AZ, Cross AZ or Cross Region.

There will be async replication between master and read replica instance, which also means reads are eventually consistent.

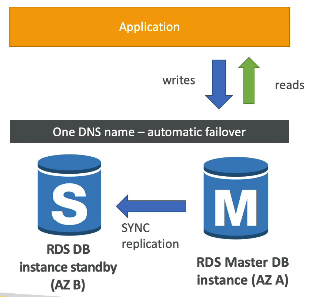


* In AWS there is a network cost when data goes from one AZ to other AZ. Within same AZ network cost is free.

**RDS Multi AZ:**

This used for disaster recovery.

* Data replication will be synchronous
* One DNS name – automatic app failover standby
* Increase availability
* Not used for scaling (as a standby database)
* Read Replicas can also be set for disaster recovery multi AZ.



## RDS Security Encryption + Security

At rest encryption

* Possible to encrypt Master and Read Replicas with AWS KMS-AES-256 encryption
* Encryption has to be defined in launch time
* If master is not encrypted, the read replicas can’t be encrypted

In flight encryption

* Data encryption during transit
* SSL certificate to encrypt data to RDS in flight
* Can create encrypted database from unencrypted database/snapshot

**RDS Security Network & IAM**

Network security

* RDS databases are usually deployed inside a private subnet
* Need to attach Security Group to decide which IP / security group can communicate with RDS.

Access Management

* IAM policy can help in control who can manage AWS RDS (through the RDS API)

## Amazon Aurora

Aurora is a cloud optimized claims 5x performance improvement over MySQL on RDS, over 3x performance improvement of Postgres on RDS.

Comes with MySql and Postgres compatibility.

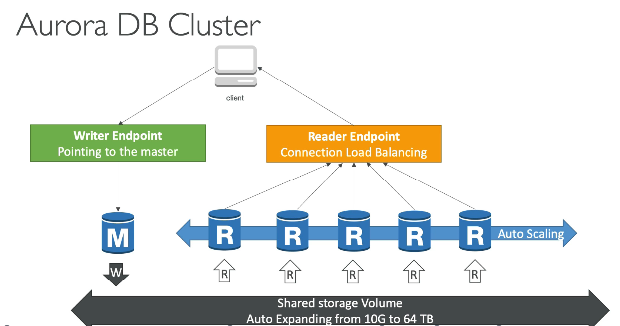
* Aurora storage automatically grows in increment of 10GB up to 64TB.
* Can have 15 replicas (MySql have 5), and replication process is faster.
* Highly available
* Aurora cost 20% more than RDS

### High availability and Read Scaling

* 6 copies of the data across 3 different AZ
* One Aurora instance takes writes (master)
* Automated failover for master less than 30 second
* Master + up to 15 aurora read replicas serve reads
* Supports cross region replication

### Aurora DB cluster

* It has Writer End Point always pointing to Master, so If Master fails we still be pointing and writing to same endpoint.
* Similarly it has “Reader End Point” pointing to all read replicas (as for us it’s very hard to know where all read replicas are located)



### Features

* Automatic failover
* Automated patching with zero downtime
* Advance monitoring
* Routine maintenance
* Backtrack: restore data at any point of time without using backups.

### Aurora Server less

* Good for infrequent, intermittent or unpredictable workloads
* No capacity planning needed
* Automated database instantiation and auto-scaling based on usages, scale based on demand

### Aurora Global Database

* 1 primary region (read/write)
* Up to 5 secondary (read-only) region, replication lag is less than 1 sec
* Up to 16 read replicas per secondary region
* Helps for decreasing latency
* Promoting another region (for disaster recovery) has an RTO of < 1 minute

Application from one region can read data from other region and perform read-only workloads.

## Elastic Cache

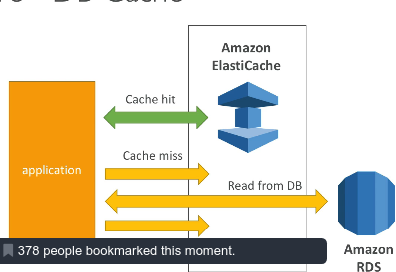
The same way RDS is to get managed to relation databases, Elastic cache is to get managed Redis or Memcached.

* Reduce load from database
* Helps application making stateless
* Write scaling using shading
* Read scaling using read replicas
* Multi AZ with failover capability
* AWS takes care of OS maintenance/patching, optimization, setup, configuration, monitoring, failure recovery and backups.

### Solution Architecture

**DB Cache:**

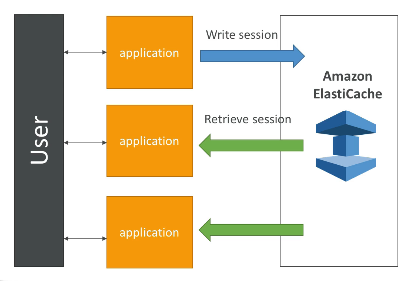
Application queries Elastic cache, if not available get from RDS and store in Elastic cache.



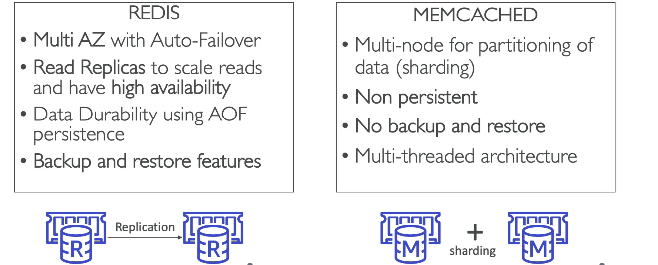
**User Session store:**

User will login to one of the application and user session will be stored to Elastic cache, when user hits another instance of application and by getting the data from cache user doesn’t have to re authenticate him.

So the application will be stateless.



### Redis v/s Memcached



Redis is more like RDS. But Memcached operates on **sharding.** In sharding data will be partitioned (based on some key) and persisted to multiple nodes.

# Route 53

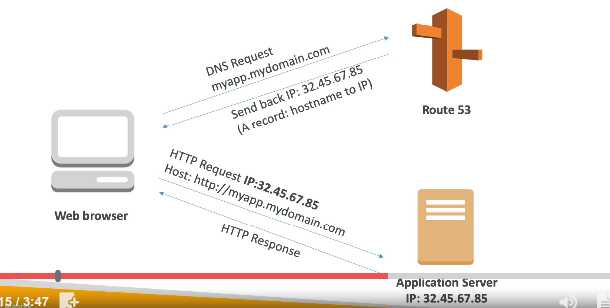
Route 53 is a managed DNS (Domain Name Service)

In AWS most common records are:

* A: hostname to IPv4
* AAAA: hostname to IPv6
* CNAME: hostname to hostname
* Alias: hostname to AWS resource

Note:

The DNS protocol doesn’t allow to create a CNAME record for top node of a DNS namespace (mycompany.com) also known as zone apex.



Route 53 has advanced feature such as:

* Load balancing through DNS (called client load balance)
* Health checks
* Routing policy: simple, failover, geolocation, latency, weight, multi value

## TTL (Time To Live)

For each DNS record it’s mandatory to specify a TTL. Setting up TTL will reduce load on DNS server (Route 53).

With TTL Browser will cache the DNS mapping and for that particular time will directly reach to the IP instead contacting to Route 53.

With high TTL we possibly may get outdated record.

## Routing Policy

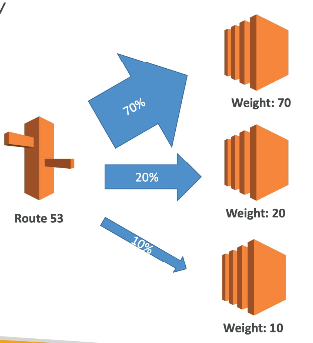
### Simple Routing Policy

This is just a simple mapping with DNS name and IP to redirect.

* Use when we need to redirect to a single resource
* We can’t attach health checks to a simple routing policy
* Can return multiple values to client and client will randomly choose a value (client side load balancing)

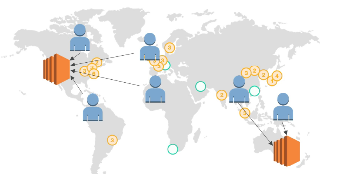
### Weighted Routing Policy

Controls the % of the requests that going to specific endpoint.



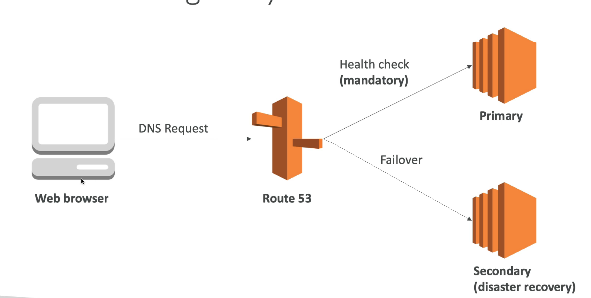
### Latency Routing Policy

Redirects the user that has least latency close to the user.



### Failover Routing Policy

Here we will have 2 record set with Primary and Secondary Record type and if primary health fails Route 53 with redirect to secondary type.



### Geo Location Routing policy

* Different from Latency based
* This is routing based on user location
* Here we specify traffic from UK should go to specific IP.
* Should create default policy (In case of no match situation)

### Multi Value Routing Policy

* When routing traffic to multiple resources
* Want to associate a Route 53 health checks
* Up to 8 healthy records are returned for each multi value query
* This is client side load balancing
* Multi value is not a substitute for having ELB

## Route 53 Health Check

Once we define health check in Route 53, they can be linked to Record Set or DNS queries.

* Have X health check failed => unhealthy (default 3)
* Have X health check passed => healthy (default 3)
* Can have HTTP, TCP, and HTTPS health checks (no SSL verification)
* Can mapped to Cloud Watch

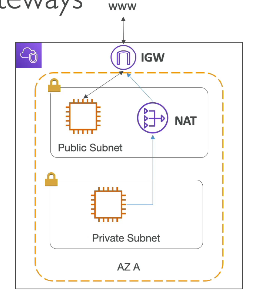
# VPC Fundamentals

VPC is a private network to deploy our resources (regional service). We use subnet to partition the VPC.

## IGW and NAT Gateways

Public subnet connects to internet through Internet Gateway (IGW)

NAT Gateways (AWS Managed) & NAT Instances (Self-managed) allows the instances in private subnets to access internet while remaining private.



## Network ACL, Security Group & VPC Flow Log

NACL

* NACL is firewall controls traffic in and out of subnet. Can have ALLOW and DENY rules.
* We attach NACL at subnet level.
* Rule only includes IP address.

SG:

* A firewall that controls traffic to and from an ENI / EC2 instance
* Can have only ALLOW rule
* Rule includes IP address and other security groups

**VPC Flow Log:**

Capture information about IP traffic going into your instances. Used for troubleshooting network connectivity issue.

* VPC Flow Logs
* Subnet Flow Logs
* Elastic Network Interface Flow Logs

## VPC Peering, Endpoints, VPN, DX

### VPC Peering

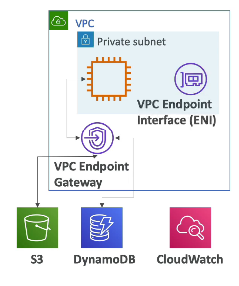
Connecting two VPC privately using AWS network, make them behave as if they were in the same network.

* VPC’s mustn’t have overlapping CIDR (IP address range)
* VPC peering connection not transitive

### VPC Endpoints

Endpoint allows connecting AWS service using a private network instead of the public www network.

* This gives enhanced security and lower latency to access AWS services



### Site to Site VPN & Direct Connect

Using these we can connect our on-premises datacenter with AWS.

**Site to Site VPN:**

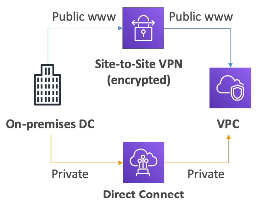
* Connect on-premises VPN with AWS
* The connection is automatically encrypted
* Goes over public internet

**Direct Connect:**

* Establish a physical connect between on-premises and AWS
* The connection is private, secure and fast
* Goes over a private network
* Takes at least a month to establish

Note:

Site-to-Site VPN and Direct connect can’t access VPC endpoints.



# Amazon S3

## S3 Buckets and Objects

Amazon S3 allows people to store objects (files) in “buckets” (directories).

* Buckets must have globally unique name
* Buckets are defined at region level

Objects (files) have a key (prefix + object name)

* S3://my-bucket/my\_files.txt
* S3://my-bucket/my\_folder1/another\_folder/my\_files.txt

## Versioning

The files/objects in Amazon S3 can be versioned but need to enable from bucket level. This means if we re-upload the file (same key) it will create a new version of the file.

* Protect against unintended deletes (ability to restore to previous version)
* Easy rollback to previous version
* Any file uploaded to bucket prior enabling versioning will have version NULL.

## S3 Encryption For Objects

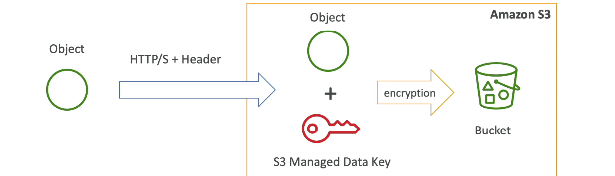
Amazon gives 4 options to encrypt object in S3

* SSE-S3: Encrypt S3 objects using keys managed and handled by AWS
* SSE-KMS: Leverage KMS key management service to manage encryption keys
* SSE-C: When we want to manage our own encryption keys
* Client Side Encryption

### SSE-S3

Encryption using keys handled and managed by Amazon S3

* Object is encrypted server side
* AES-256 encryption type
* Must set header: “x-amz-server-side-encryption”:”AES256”



### SSE-KMS

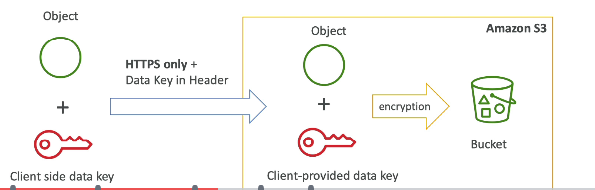
Encryption using keys will be managed and handled by KMS. KMS advantage user control + audit trail.

* Object encrypted server side
* Must set the 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.

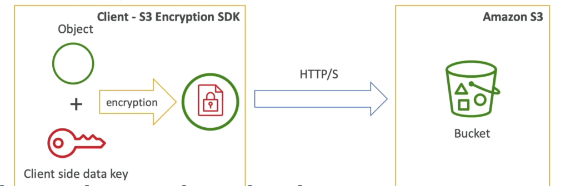
* Amazon S3 doesn’t store the encryption key you provided
* HTTPS must be used
* Encryption key must be provided in HTTP headers, for every HTTP request made



### Client Side Encryption

Encrypt the objects before uploading to S3. Some client side library will help to do it.

* Client library such as the Amazon S3 Encrypt Client
* Client must encrypt the data themselves before sending to S3
* Client must decrypt the data themselves when received from S3
* Customer fully manages the key and encryption cycle



### Encryption in transit (SSL/TLS)

The data transfer between client and S3 will be encrypted. Need to use SSL/TLS certificate

### Default Encryption

We can set Default Encryption for S3 bucket, so every object uploaded will be encrypted.

We can set this from bucket properties.

## S3 Security

### User Based

* IM Policy – Which API calls should be allowed for a specific user from IAM console

### Resource Based

Bucket wide rules from the S3 console – allows cross account

**Note:**

An IAM principal can access an S3 object if

* The user IAM permission ALLOW it OR the resource policy ALLOW it AND there’s no explicit DENY

S3 bucket policies are JSON based policies.

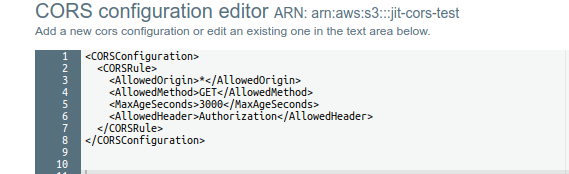
## S3 Websites

S3 can host static website.

* Enable Static Website Hosting from properties
* From Permissions: Disable block public access
* Add a Bucket policy for public access to Get Object

## CORS

When a try to access resource from a different S3 bucket (hosted static website) from a S3 bucket (hosted static website), CORS should have been enabled in second S3 bucket.



## Consistency Model in S3

S3 is eventually consistent. So the PUT and DELETE operation we made may take some time to reflect the latest state.

## AWS CLI, SDK, IAM Roles & Policies

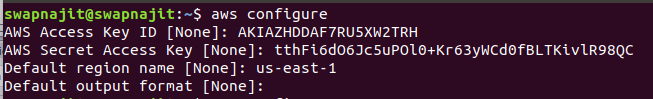
### AWS CLI on Linux

1. curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"
2. unzip awscliv2.zip
3. sudo ./aws/install
4. aws --version (To check)

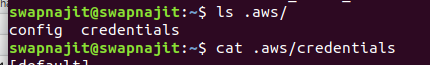
After installing CLI if you get error, *aws: command not found.* The reason could be “the aws executable is not in the PATH environment variable”.

### AWS CLI Configuration

To configure AWS CLI



After configuration it will create a fine under



### AWS CLI on EC2

We can run “*aws configure”* in EC2 and it will run (but it’s really BAD, as it will store your credentials in EC2 in .aws/credentials)

The GOOD way to configure AWS CLI on EC2 is with IM Roles.

* IAM Role can be attached to EC2 instances
* IAM Role can come with a policy authorizing exactly what the EC2 instance should able to do.

This is the best practice on AWS and you should 100% do this.

## IAM Roles and Policies Hands On

IAM policies can be used as AWS Managed policies or Customer managed (we can create our own policy).

To role also we can add an *inline policy* (this is not a recommended). Recommended is to create a policy and attach to Role.

We can graphically generate AWS policies using *Visual Editor* (In IAM Console)or *AWS Policy Generator.*

## AWS Policy Simulator

AWS Policy Simulator is an online tool provided by Amazon to test the policy and checking the behavior (ex. why something got denied).

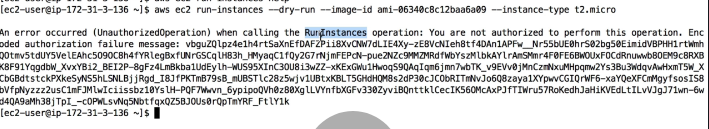
## AWS CLI Dry Run

Sometimes checking something if we have permission can be quite expensive (ex. If we check if we have permission to create an EC2 instance, it might down the line create an EC2 instance)

So we just need to simulate the API calls, but not actually run the command. So we have **--dry-run** option is for.

(Not all commands have dry run option)

Let’s check from an EC2 instance if I have permission to create an EC2 instance.



If we add “RunInstance” policy for EC2 to our IAM Role, the *dry-run* will be successful but instance will not be created.

To decode the above message we need to execute as

$ *aws sts decode-authorization-message –encoded-message <<encoded message>>*

## AWS EC2 Instance Metadata

EC2 Instance Metadata allows AWS EC2 instances to “learn about themselves” without using an IAM Role for that purpose.

The URL is [*http://169.254.169.254/latest/meta-data*](http://169.254.169.254/latest/meta-data)

* This URL is internal to AWS
* You can retrieve IAM role name from the metadata, but you can’t retrieve the IAM policy.
* Metadata = Info about the EC2 instance
* Userdata = launch script of the EC2 instance

## AWS CLI Profiles

If I have multiple AWS account and to configure those in my system (it will store the credentials)

$ *aws configure –profile my-other-account*

## MFA with CLI

* To use MFA with the CLI, you must create a temporary session
* To do so, you must run the STS GetSessionToken API call

## AWS SDK

If we want to perform actions on AWS directly from your applications code we need to use AWS SDK (Software development kit)

There are bunch of SDK for AWS in different languages.

AWS CLI is written in Python.

So we use SDK when coding against AWS services such as DynamoDB.

**Note:**

If we don’t specify or configure a region, then *us-east-1* will be chosen by default.

## Exponential Backoff & Service Limit Increase

AWS API Rate Limits

* DescribeInstances API for EC2 has a limit of 100 calls per seconds
* GetObject on S3 has limit of 5500 GET per second per prefix
* For intermittent Errors: implement Exponential backoff (Retry mechanism included in all AWS CLI and SDK, every time you retry double the time of previous)
* For Consistent Errors: request an API throttling limit increase

Service Quotas (Service Limits)

* Running On-Demand standard instances: 1152 vCPU
* You can request a service limit increase by opening a ticket
* You can request a service quota increase by using the Service Quotas API

## AWS CLI Credentials Provider Chain

The CLI will look for credentials in this order

1. Command line options: --region, --profile, --output
2. Environment variables: AWS\_ACCESS\_KEY, AWS\_SECRET\_ACCESS\_KEY and AWS\_SESSION\_TOKEN
3. CLI credentials file: aws configure
4. CLI configuration file: aws configure
5. Container Credentials
6. Instance profile credentials

AWS SDK (ex. java SDK) will look for credentials in this order

1. Environment variables: AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY
2. Java system properties: aws.accessKeyId and aws.secretKey
3. The default credential profiles file: ex at: ~/.aws/credentials shared by many SDK
4. Amazon ECS container credentials – for ECS containers
5. Instance profile credentials – used on EC2 instances

## Signing AWS API requests

When we call the AWS HTTP API, we should sign the request so that AWS can identify us, using your AWS credentials (access key & secret key).

* Some requests to Amazon S3 don’t need to be signed
* If you use the SDK or CLI, the HTTP requests are signed for you
* We should sign an AWS HTTP request using Signature v4 (Sig V4)

# Advanced S3 and Athena

## S3 Access Logs

For audit purpose we may have to log all the access to S3 bucket (either ALLOW or DENY) for future analysis purpose.

The monitoring bucket and logging bucket should be different (else it will create a logging loop).

## S3 Replication (Cross Region and Same Region)

To replicate S3 bucket asynchronously into another bucket in another region, to do so

* Must enable versioning in source and destination
* Cross Region Replication (CRR)
* Same Region Replication (SRR)
* Must have an IAM role with permission to copy from one bucket to other.

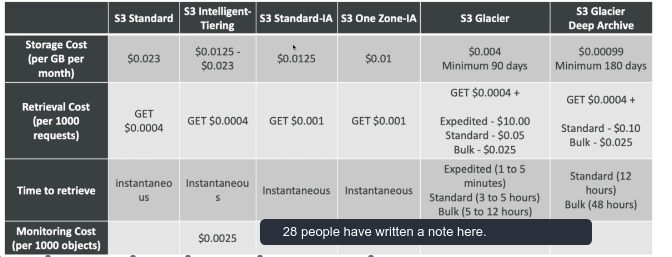
Delete operations will not be replicated.

## S3 pre-signed URLs

Pre-signed URLs are the ways for the user to access bucket/file without giving direct access to bucket.

* We give default expiration time while creating
* Default expiration time is 3600s
* Give region while creating
* Can be created using CLI or SDK

## S3 Storage Classes



## S3 Select and Glacier Select

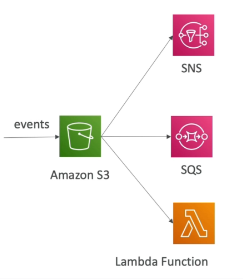
Retrieve less data using SQL by performing server side filtering. Can filter by rows and columns (simple SQL statement). This will lead to less network transfer less CPU cost client-side.

## S3 Event Notification

Events can be created on S3 bucket.S3:ObjectCreated,S3:ObjectRemoved,S3:ObjectRestore,S3:Replication…

Use case: Generate thumbnail of images uploaded to S3

* We can create as many “S3 events” as desired.
* To ensure event notification sent, need to enable versioning on the bucket.



## AWS Athena

* AWS Athena is a serverless service to perform analytics directly against S3 files
* We can query directly on S3 itself, without loading the file to database
* Can use SQL language
* Charged per query and amount of data scanned.
* Supports CSV,JSON,ORC,Avro (built on Presto – query engine)
* Use cases: Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, Cloud Trails etc