**S3 vs EBS vs EFS**

**Block Storage –** Block level operations are possible; hence one block is changed (piece of the file) that contains the changed data

* Very fast read and write operations block storage would be best
* EBS – block level (EFS also block level storage)

**Object Storage –** Entire file must be removed and new file needs to be put there

* Put the files occasionally object level would be best
* S3 – Object level

S3 & EBS

**S3 –**

* used for WORN operations (Write Once Read Many times)
* writing continuously (but not reading) s3 is good. log file from different sources continuous ingestion S3 would be best
* store static assets of a website you put once and access world wide(write once read many times)
* Scalable, size not be planned. No restriction on sizing, you can keep loading the files.
* But not suitable for hosting OS or Database (but you can store) reason is OS and DB are required very fast read and write operations.

**EBS –**

* EBS works best for hard disk or server disks. So use with EC2
* Persistent and high performance in terms of read write. Even you stop EC2 instance data will not be lost. Depending on EBS type you can achieve high read and write performance.
* Replicated with in AZ and could be mounted to one EC2 in the same AZ. 2 times it will replicated in same AZ by AWS. Create an EBS in one AZ will **not** be possible to attach in EC2 in other AZ.

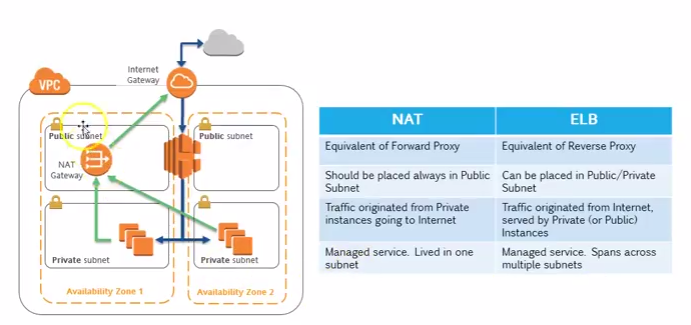
**EFS –**

* Replicated across AZs in a region.
* One EFS you can mount on ‘n’ number of EC2 instances within a region. Ex: you create one EC2 in AZ-a and second EC2 in AZ-b and mount same EFS volume.
* When you create EFS in AWS, underlaying the EFS is replicated across different AZ in the same region by AWS.
* Could be mounted to on premise server as well (over VPN or DirectConnect)
* No sizing to be done (unlike EBS). Auto scales.

[**AWS - Difference between NAT and ELB - Comparison**](https://www.youtube.com/watch?v=G67TaU4qSYE)

NAT – Network Address Translation (NAT Gateway)

ELB – Elastic Load Balancer



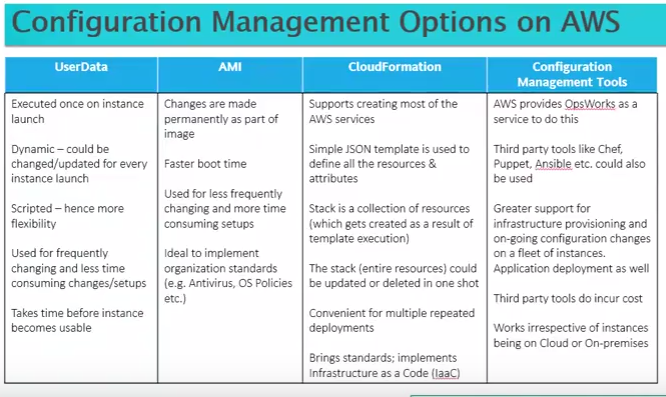
**ELB –**

* Webservers are kept in private subnet and serving public people(internet) through ELB which is in public subnet. If we are not using ELB we should put our webservers in public subnet to access the public people.
* **ELB** – does 2 things 1) accept the incoming traffic from internet(public people) and distribute among the instances which are registered to it. 2) continuous do the health check of the instances. If any of the instance is unhealthy it will stop sending the traffic to that particular instance.
* Some times you can keep ELB in private also, between your web tier and app tier
* ELB is **managed service** and highly available**(HA)** in nature, you do not need to maintain the internal instances of the ELB, ELB will takes care of it.
* ELB can span in multiple subnets in multiple AZ (see this in the above pic).

**NAT Gateway –**

* When your instances call APIs like whether, currency etc in internet, so your private instances should reach internet through NAT Gateway (forward proxy). It is similar to your desktop (having private IPs) accessing internet.
* The traffic which is originated towards internet by the private machines goes via NAT instance. And the reply comes back.
* NAT Gateway is **managed service**, its HA within one AZ. NAT Gateway never span across multiple AZ. For more HA you can put separate NAT Gateways in each AZ.

**AWS Configuration Managment Options - UserData, AMI, CloudFormation, OpsWorks**



**User Data –**

* User data is nothing but scripts that run on cmd prompt in linux or windows for customization.
* More efforts required, you can run once when your instance is launched

**AMI –**

* Create once use many times, less effort when you do the same on multiple instances.

**CloudFormation –**

* Specify all your resources in JSON template and feed this template into CloudFormation engine and CF engine create a stack (nothing but collection of resources). All the resources come up for you in one shot.
* It can help to automate your deployments, also you can do multiple deployments and ensure that same standards that are used in one environment remains the same when other environments come up.
* It is concept of infrastructure as a code
* When you use a tag, the same tag will be replicated in all the resources in environments, good thing is if you enable this tag as cost allocation tag everything flows to detail billing file.

**Configuration Mgt tools –**

* 3rd party tools – chef, puppet, ansible etc
* Aws provides OpsWorks to do the same thing. You can also use this on premise, but they will charge for it.

**Demo on User Data –**

#!/bin/sh

Yum -y install httpd (apache web server)

Chkconfig httpd on (checking server is on or not)

/etc/init.d/httpd start (start the service)

* Put this code in ‘User Data’ under ‘Advanced Details’ in ‘3.Conguration Instance’ step in EC2 instance creation process through AWS console.
* Any user data code is executing as root in linux.

**AWS CloudFormation - Stack, Template, Parameters, Mapping, IAM Role, Stack Policy**

Stack, Template, Parameters, Mapping, Functions, Pseudo Parameters, Deletion Policy, Tags, Using IAM Role, Stack Policy.

* CF is a service give you the ability create complete aws environment with different resources, services in a template called JSON.
* **Template** – collection of resources, it contain attributes etc.
* **Stack** – Once templates is feeded to CF engine it will create the resources for you, collection of all these created resources are called Stack.
* After your work is finished you can go ahead and delete the stack, CF will take care of deleting all your resources as part of the stack.
* Json editor can help to write templates.
* CF designer can help to write templates; it has some intelligence.
* Provision AWS resources via Code (JSON or YAML).
* Create, Manage and Delete related AWS resources together as a unit called a **Stack**.
* **Templates** are JSON files or YAML file which define your stack.
* Version Control and Track changes

**CF template Elements:**

1. **AWSTemplateFormatVersion**
2. **Description**
3. **Metadata**
4. **Parameters**
5. **Mapping**
6. **Conditions**
7. **Transform**
8. **Resources – Required Section**
9. **Outputs**
10. **Intrinsic Functions**