EC2,EBS,EFS,ELB,AUTO SCALING,S3,IAM,VPC,ROUTE 53,RDS,CLOUD WATCH

CLOUD FORMATION, LINUX

**EC2**

* Aws resources hosted multiple locations in worldwide in that which is separated by geographical area, each region has multiple isolated locations called availability zones.
* Ec2 means elastic cloud computing, this is nothing but configure the server(instance)
* Important resources for launching or configure the server in Ami (OS), instance type (t2.micro), security group and key pair.
* Ami stands for Amazon machine image it is nothing but operating system, application it is region level, every Ami have its own Id.
* Security group is act as virtual firewall for server, it have 2 rules inbound and outbound ,instance that can allow request is inbound rule, instance that can send request to other is set in outbound rule,
* SG default outbound rule is all traffic, we can set only inbound rule while launching the instances.
* One security group can be attached to multiple instances, but one instance can have max up to 5 SGs
* Key pair is required to launch the instances download key pair must exist in computer, one key pair can be used multiple times for instance.
* Instance type is nothing but processer, varying combination of CPU, memory, storage and network are the required applications for server.

Ex : t2.micro,t2.small

* For launching the instances Ami, instance type, storage, security group, keypair are required.
* Login the instance by using the keypair,

ex: ssh - i “privatekeyname.pem” username@ public\_ip

* User data is the commands that executed in server booting, we can install any web server using user data, it can also be edited but instance is in stopped state.
* 1st line of the user data is shebang(#!/bin/bash)
* Change/add user data for instance is when instance is stopped then go instance setting-change/add user data field
* Stop, terminate and reboot the instance for these go to actions-instance setting-once terminate the instance it will shutdown the instance permanent collapse the infra structure.
* When we stop the instance public ip removed from the instance then start the instance new public ip will generate for instance
* When we reboot the instance public ip will remain same
* Elastic Ip is dedicated ip address, it is reserved for account it can be assigned to any ec2 instance, even stop the instances ip will attached then start the instance ip will remain same.
* Only 5 elastic Ips can be reserved for account.
* There are 3 storages in Aws EBS,EFS and S3,
* EBS is the elastic block storage is the primary storage for ec2 instances, it is block storage, only in ebs we can install Os and softwares,
* One ebs can attach to one instance at a time of launching instances, ebs volume and instance can be attached if both are in same availability zone.
* root volume means Os and softwares can install in this volume at the time of launching instance, identification “xvda” cant rename it)
* ebs volume(disc) performance is measured iops(input output per second).
* 5 types of ebs volumes gp2,io1,sc1,st1 and magnetic
* out of these only gp2,io1 and magnetic are the root volumes.
* Volume attachment to ec2 instance- by linux commands attach the volume to instance “lsblk” to check the attached or not then create file system

“mkfs –t ext4 /dev/xvdf”, check file system not created “file –s /dev/xvdf”

create directory ” mkdir dir\_name” then mount the directory to instance

“mount vol\_name dir\_name” go to directory create some files in that.\

* Then unmount directory from instance “umount vol\_name dir\_name” finally detach the volume from instance.
* Server up gradation means change instance type T2 micro to T2 small or any other.
* Snapshot is the in time backup of the volume, it is region level ,by using snapshot we can create new volume then attach to any instance and we can copy the snapshot for other regions also, snapshot stored in s3 bucket.
* One Az volume we can attach to other AZ instance means we can create snapshot for the one AZ volume then create new volume using snapshot in instance’s AZ then attach it to instance.
* Same like above region also we can attach one region volume to other region instance.
* Ami (amazon machine image) it is used to boot the server, by using the ami we can launch the multiple instances, we want webservers in our instance more then we create image for the instance then we create more number of instances by Ami in less time, snapshot will create along the Ami.

**EBS**

* **Ec2-volume mounting:**

Launch linux ec2 instance, create another volume with same availability zone (AZ) of instances then attach that additional volume to the instances, check whether the volume is attached or not by using linux commands

1. lsblk to check the volume attached or not

2. check file system is created or not

3. if file system not created use "mkfs -t ext4 /dev/xvdf"(linux uses "ext4" file system)

4. create directory(newvol)

5. mount the directory to volume "sudo mount /dev/xvdf directory\_name"

6. go to directory "cd direc\_name"

7. create files using touch (f1 f2)

8. unmount the directory from volume "sudo umount /dev/xvdf dire\_name"

9. detach the volume from ec2 instance

10.create another ec2 linux instance attached that previous volume

11.connect the instances check volume attaced or not "lsblk"

12.create directory with same name(newvol)

13.mount that directry to (previous)instance "sudo mount /dev/xvdf newvol"

14.go to directory check "ls", f1 f2 files should display (wch we already created in 1st instance).

* 15.creating snapshots by using volume then delete the volume, make new volume by snp attach to instance
* then check the files

**EFS**

* EFS stands for elastic file system it is used to sharing of file between instances, it is not available in all regions like EBS and S3,efs less pricing

and unlimited storage.

* Efs is created within default VPC so we can mount the efs to instance whatever we created in same default VPC.
* Use of mounting the efs to instance is when we create some file in one instance then unmount the efs again attached to another instance then check same files should reflect other instances.
* **Mounting ubuntu instance and efs:**

1 .launch ubuntu instance then attach default security group to that instance login the instance

2. check the file system is available or not then create file system using default security group,AZs

3. Install the "nfs-common" by using "sudo apt-get install nfs-common"

4. Create directory “mkdir dir”

5. Mount the file system to ec2 instance by using the commands available on dashboard of file system

6. Go to directory create some files the then launch another ubuntu instance attach default security group

7. Login the instance create directory (previous name) which is already created in previous instance

8. mount the instance to file system then go to directory check the” ls” command

9. The files should display which is created in previous instance.

* For linux instance installation cmd “sudo yum install –y amazon-nfs-utils”

Remaining same like ubuntu.

**ELB**

* Elastic load balancer distributes the incoming application traffic across the multiple targets.
* It can handle the varying load of application traffic in single or multiple availability zones.
* 3 types of elb classic, application and network elbs
* 2 types of listeners in elb frontend and backend listeners, listener means is the process that checks for connection request,
* Frontend listeners- it is configured with protocol and port with client to elb or connection request from client to elb.
* Backend listeners – is configured with protocol and port from elb to ec2 instance connections or connection request from elb to ec2 instances
* Elb supports the following protocols :Http
* Elb monitoring the health checks of the registered instance if instance is in healthy then only elb routing the traffic to instance, healthy instance show in-service under the elb.
* When elb detects instance as unhealthy it is stop routing traffic to that instance, unhealthy instance shows out of-service under the elb.
* By default elb uses ping request to Http (port 80) for health checks.
* Health check intervel: period of time between health checks.
* Healthy threshold: No of consecutive success health checks that must occur before declaring the instance is healthy.
* Unhealthy threshold: No of consecutive failed health checks that must occur before declaring the instance is unhealthy.
* When elb declare the instance is came back to healthy then resumes the traffic routing to that instance.
* Cross zone load balancing: If this feature is disabled elb sends the traffic to evenly to AZs without consider number of the registered instances in each AZ.
* Sticky sessions: With this feature, we can configure the elb to bind the user sessions to the specific application instance, all the request from the users are sent to the same application instance only.
* Connection Draining: Connection draining allows existing requests to complete before elb shifts the application traffic away from the unhealthy back end instances.
* ELB idle timeout: it is the number of seconds the connection can be idle before elb closes the connection.

**Classic ELB:**

* 1.Launch 2 ubuntu instance is different AZs, allow http request in inbound rule

2. Connect and login then install apache2 in both instances

3. Create classic load balancer (elb: prtcl-http, port-80, instance: prtcl-http, port-80),

4. Attach new SG (type: http, prtcl: Tcp) then register with instances which is in running state,

5. change the apache homepage in one instance for sudo su->cd /var/www/html -> cat >index.html write anything (My home page),

6. Check the instances state then wait for state in service then copy and browse the DNS name

7. Check the load distribute between two or more instances (ips).

* Application elb: Based on the application context we can distribute the load between application related servers.
* Prepaid request need to be landed on prepaid servers
* Postpaid request need to be landed on postpaid servers.
* Target groups-group the servers based on the application context, it is called as the target groups

**Application ELB:**

* Launch 4 ubuntu instances in different AZs, allow http request in inbound rule
* Login the instances then install apache2 in 4 instances,
* Change the apache2 home page in two instances, follow the rules switch the user “sudo su” then go to apache2 home page “cd /var/www/html”
* Create directory “mkdir prepaid”, then go to directory create one html file “touch f.html” write something in that file “cat >f.html” “This is prepaid 1”
* Do the same for other instances also like prepaid
* Then change the page with postpaid, make directory with postpaid, create file t.html then write “This is postpaid 1”
* Do the same for other instances also like postpaid,
* Create 2 target groups one is prepaid and other is postpaid then register the prepaid instances to prepaid target group
* And register the Postpaid instances to postpaid target group,
* Create application load balancer then attach the existing target group prepaid then create the elb,
* Then go to add listener edit the rule then insert rule then add condition prepaid then add action forwarded to prepaid instances then save
* Then copy the DNS name into the browser then add /prepaid/f.html then check the load distribution between the prepaid instances
* Same like copy DNS name then add /postpaid/t.html like this then check the load distribution between postpaid instances.

**Auto Scaling:**

* Auto scaling is service in aws which is used to automatic scaling in and out the ec2 instances based on the user requirements and policies.
* Scaling-in means terminating or removing the instances which is not use for current requirements based on scaling policy
* Scaling-out is launching or creating the number of instances for requirements
* Auto scale always try to even distributions of the ec2 instances between the Azs
* Auto scale can span the multiple Azs within the same region, hence it can be used for fault tolerance design
* Auto scale will work with ELB and Cloud Watch, and it cannot span of the multiple regions.
* 2 Auto scaling components are there launch configuration and auto scale group, which type of the instance will launch ami, instance type, security group and key pair are will select in launch configuration it can’t be edited
* What type of scaling activity manual/event based, scaling activity parameters and scaling group health checks those are selected in auto scaling group it can edit based on the requirements
* Scaling activity parameters min: least number of instances should be running, max: highest number of instances that should be running.

Desire: the number of instances should be running it must greater than the min and lesser then the max number of the instance, desire will take min number as default if don’t mention the desire number.

**Auto scaling Rebalancing:**

* If Az finds the number of ec2 instance will not evenly distributed across Azs

Auto scale will initiate rebalancing activity, Target of the AS is evenly distribute the instances in Azs

* In rebalancing activity AS temporarily launching more number of instances than what you have define in ASG max value.

**Reasons for the im balancing in AS:**

* Selected Azs having not enough capacity of instance type, adding and removing the instances ,manually requesting terminating the instances from ASG
* 3 types of auto scaling, Manual auto scaling: Maintain current number of instances at all time and define the ASG with min, max and desire numbers

Scheduled: used for predictable time, you need to schedule the scaling out action at specified date and time, must follow unique date and time

You can configure the scheduled once at recurring, you can edit the schedule AS, Event based: scaling in response to an event.

**S3 bucket:**

* S3 is the simple storage service for the internet, web service interface storage and retrieval of any amount of data from anywhere at any time,
* S3 is object based storage, it has distributed data storage architecture, objects stored in multiple locations
* We can attach s3 bucket to ec2 instances manually,
* There are 4 types of the storage classes, we keep objects into any type of storage class, it can be divided based on the pricing, Durability and Availability.
* S3 standard: high durability, availability and high performance object storage for frequently accessed data 99.99999% durability and availability
* S3 standard A:Infrequently accessed, it is for data that accessed less frequently but it required rapid access when it needed, low storage price per GB and low retrieval fee per GB then standard class
* Reduced Redundancy storage: it the new option to customers reduce their cost by storing the non critical reproducible data at lower levels of redundancy then standard class.
* Glacier: it is archiving storage for infrequently access data, archived objects are not available for real time access, you need to request for retrieval data then restore before you read it. Archived data will copy to RRS class, after data is retrieved it will take upto24 hrs to download.
* You can store the data for 90days, upload size 1byte -40 TB
* You can upload the data to glacier through directly to cli, not through aws console, we can transitioning data to glacier from other classes via life cycle rules, durability 99.99999% but no availability.
* S3 bucket is object storage, bucket created in account, upto 100 buckets can create objects upload directly into bucket, access permissions, version controlling and storage are the properties of the bucket.
* By default bucket in private, we can write policies for public access, we can define the lifecycle rules to change from one storage class to other storage class inside the bucket.
* Versioning can be enable at bucket level, bucket name must be unique
* If we upload any object to s3 bucket it will create unique url for that object.

**S3 life cycle rules:**

* We can define the life cycle rules for objects move from one storage class to other storage class, standard to infrequent, infrequent to glacier and glacier to expire
* If objects moving from standard to infrequent class objects must be min 30 days in standard class,
* If objects moving from standard to glacier class objects must be min one day in standard class,
* You can define separate lifecycle rules for current version objects and previous version objects. You can write multiple lifecycle rules for same objects.
* Cross region replication: Replicate the s3 objects from one region bucket to other region bucket, if cross region replication needs both source and destination buckets must in different regions and versioning enable in both buckets.

**Aws CLI** (Command line interface)**:**

* Cli is the special software to run the aws activities whatever we create using GUI those are can do with cli also, like create, stop and terminate instances ,creating/deleting s3 objects and upload objects using instances.

**Access S3 bucket from ec2 instance:**

* Install cli in ubuntu instance sudo apt-get install awscli
* Configure the account to s3 “aws configure” then enter access key and secret access key then access s3 in instance
* Check aws s3 ls for displaying list of buckets then upload/down load the files from s3 to instance by using linux comds
* Create s3 buckets through ec2 instances “aws s3 mb s3://<bucket name>”

**IAM (identity access management**):

* Iam is a service in aws by using this service we can give restricted access to custom users, it is account level service,
* By using policy we can give permission to users policies are 2 types

User based policy it is assigned to user and resource based policy means policy is directly assigned to resource itself only, ex: s3

* Policy contain effect, action, resources, version, statement, condition and principle
* Effect having 2 options Allow and deny and it is written in Json format.
* Action contain start instance or terminate instance etc
* Resource contain create/destroy the instances if we launch 2 instances those 2 are resources, resource name give in ARN format (Amazon resource name) ex: arn:aws:ec2:<region>:<account id>:instance/<instance id>
* Example policy: Launch all other ec2 action allow except instance termination

“Version “:”2012-10-17”

“Statement”:[{

“Effect”:”Allow”,

“Action”:”ec2:\*”,

“Resource”:”\*”

}{

“Effect”:”Deny”,

“Action”:”ec2: Terminate Instances”,

“Resource”:”\*”

}]

**Iam roles:**

* By using iam roles we can access one account resource to another account, role can be assigned to ec2 instances, if ec2 instance want to access other aws services we need to configure access key and secret access key, we are compromising about our aws account security to avoid these problem we create iam roles, roles can be attached to instances,

**Accessing account1 services from account2 by using roles:**

* Create role in account 1,attach policies to that role
* Create user in account2 then attach consume account1 services policy to that user,

**Account1**: create role then select another aws account enter the account2 id then attach policies to that role,

**Account2**: create user then attaché assume policy which is having with account1 id, login the user switch role then enter account1 then select role which is having account1 service policies then submit go into account1 then try to access services of account1.

**Note**: Install awscli in windows machine

**Aws configuration**:

aws configure

enter access key

enter secret access key

region

json

aws sts get-caller-identity for checking whom logged in account

**VPC (virtual private cloud):**

* Vpc is a network inside aws, span of vpc is region level only, if we want to create vpc we need to define the ipv4 address for vpc in classless inter-domain route (CIDR) block, ex :192.168.0.0/16
* Vpc can be divided into subnets, for subnets also having CIDR block, subnets exit only within Azs, vpc cidr block can be divided for subnets cidr block, it similar to have create over own network inside aws,
* 2 types of vpc default vpc and custom vpc, default vpc exist in every region in aws and it have default subnets, security group, route table, internet gateway and Nacl, for custom vpc we need to create cidr block and subnets and route tables
* Once vpc is created it can’t be edit the cidr block range, min size of cidr is 28, max size of cidr is 16,cant overlap the subnets within vpc
* Aws reserved 5 ips in each subnet 1st four and last one ip address
* **Route table**: a route table contain set of rules called routes that is used to determine where network traffic is directed , 200 RT in each subnet and each subnet connect with only one RT, if you do not specify the subnet to RT association, subnet associate with default RT, 50 route entries can give in RT, if any association want to give to subnet it will attach to RT 1st then that RT is connect to subnet,
* If you want to change the subnet association with other RT, you can edit the main RT then attach it to required subnet, you can’t delete the main RT,
* Every RT in the vpc comes with default routes that allow all subnets communicate to one another in Vpc, you can’t modify/delete this rule.
* **Internet gateway:** by using this we can give public access to the network, one IGW can connect to one vpc only, ex: google.com,facebook.com
* **Nat gate way**: you can connect to the network address translation is enable in instance in private subnet to connect to the internet or other aws network, always need to launch nat in public subnet only then connect to private subnet, while creating nat elastic ip will a create inside nat, if we need to access internet from private subnet we need to made nat entry in private subnet RT,
* **NACL(network access control list**): Nacl allow/deny the request from/to to subnet nacl consist inbound and outbound rules, each associate with one nacl but one nacl can be mapped with multiple subnets, nacl rules are stateless.
* **Working on VPC:**

1. Create vpc(10.0.0.0/16)

2. Create sn1 and sn2 make sn1 as public attach internet gate way(0.0.0.0/0),create IGW attach it to route table(rt\_igw)then attach route to subnet(sn1)

3. Launch instance by using ami and sn1 subnent

4. login the instance with public ip of the instance (sn1).

5. Then connect the instance (sn2) with private ip

6. Again login the sn2 instance to sn1 by using nat gate way,

Create nat in public subnet(sn1)attach to route table(rt\_nat)then attach route to private subnet(sn2),

Then login the sn2 from sn1

7. Same like other vpc(192.168.0.0/16)

ex:

vpc1 (10.0.0.0/16)

------------------

sn1-10.0.0.0/24--public

sn2-10.0.1.0/24--private

Sn1-sn2-connected (igw in sn1)

Sn2-sn1-connected (nat in sn2)

vpc2 (192.168.0.0/16)

----------------------

sn1-192.168.0.0/24--public

sn2-192.168.1.0/24--private

Sn3-sn4-connected (igw in sn3)

Sn4-sn3-connected (nat in sn4)

Sn1-sn2-connected (igw in sn1)

Sn2-sn1-connected (nat in sn2)

Sn3-sn4-connected (igw in sn3)

Sn4-sn3-connected (nat in sn4)

Sn1-sn3-connected (both public)

Sn3-sn1-connected (both public)

Sn4-sn1-connected (private to public different vpcs)

Sn2-sn3-connected (private to public different vpcs)

* **Vpc peering**: vpc peering connection is network connection between both vpc’s which is located in same region and having different Cidr blocks, it enables the network traffic routing between the private subnets, it helps in transfer of data, network file sharing between the accounts through the vpc’s, vpc peering connection in aws account allow to access the resources that you have in one of your vpc
* sn1-sn4-

sn3-sn2-

sn2-sn4-

sn4-sn2-

* vpc peering sn1-sn4:

1. Create vpc peering, requester is vpc1(sn1) and acceptor vpc2(sn4),

Then select the peering accept the request.

2. Go to route and edit (rt\_igw) route add vpc2 (192.168.0.0/24) and peering then save

3. Edit (rt\_nat) route add vpc1 (10.0.0.0/24) and peering then save

* Vpc peering for sn3-sn2,sn2-sn4 and sn4-sn2:

4. Same like other 2 route tables also for connecting any subnet instance to instance.

**Route 53:**

* DNS means domain name service it is globally distributed service that human readable domain names into corresponding ip address, this request resolves domain name to numerical ip address is called b queryb, it is compatible with ipv6 also,
* If we enter any website name like facebook.com then dns resolver for the isp(internet service provider) request to TLD name server then it request to hosted zone, the name server responds to that request with names of amazon route 53 that are associated with that dns name,
* 2 types of Dns systems Recursive Dns and Authoritative Dns, client typically 1st connect to recursive dns which is knows where to get corresponding ip address for dns name, then it request to Authoritative dns name to get the information, it keeps the ip addresses for domain names,
* Route 53 is authoritative, it is scalable, reliable and cost effective, it allows management of mapping ip addresses to dns names and replies to b queriesb for translating domain names to corresponding ip address.
* Route 53 policies- simple, weight based routing, latency based routing, Geo based routing and Failover(back up from the healthy application request)
* **Hosted zones**: in route 53 is easy allow to manage the multiple domain names and records, each hosted zone is created for second level-domain,

2 types of hosted zones public can access from anywhere, and private it can accessible within Vpc only.

* Dns has several types of the records for different functions, A record it maps human readable domain name to ip address of the machine where the files is reside, like AAAA record, CNAME record, NS record, MX record, TXT record and TTL record(time to live- how many seconds name server catches the information)
* Ex: we enter google.com-DNS root name server-Name server for .com TLD- Route 53 name server-web server Ip-output

**RDS (Relational** **Database service):**

**My SQL:**

* Create database- To create new database
* Use database –To change the databases from one database to other
* Drop database- To remove the database from databases list,
* Show databases- To display the created databases
* Create table- To create table in selected database it required table specifications like columns, columns field
* Show tables–To display the created table in database
* Describe table –To display the content in table format
* Alter table- To modify/add/remove content or columns from particular table
* RDS: Amazon relational database is easy to setup and scale a relational database in cloud, it provides cost efficiency and resizable capacity while automatic time consuming hardware heavy applications, database setup, patching and backups, failure detection and recovery.
* Rds doesn’t provide the shell access to DB instances, it restricted access to certain system procedures, in addition to the security to rds you can attach IAM role to that instances you can define users and permissions, you can also protect by placing in VPC,
* DB instances: basic building blocks of the rds is db instances, it isolates the db environment in the cloud, it contain the multiple user-created databases, you can create the DB instances by using amazon cli, each DB instance run on DB engine, currently it supports on oracle, mysql, mariaDB,
* RDS uses the network time protocol (NTP) to synchronize the time on DB instances,

**Creating RDS with MySQL:**

* My sql is the most popular open source database in world, rds offers the mysql is community editions to flexibility to easily scale compute resources or storage capacity of your database,
* It supports database size upto 32 TB, general purpose, memory optimized and burstable performance instance classes, automated backup and point in-time recovery and it upto 5 read replicas per instance within single or cross regions,
* Create the RDS give required configuration then login to the mysql cli or workbench then connect rds to that cli or workbench by using username, password and endpoint of RDS then create tables in mysql cli or in workbench.
* With cli use mysql version syntax:

Mysql –h <endpoint> -p 3306 –u <username> -p <password>

**Ex:** mysql -h <mysql–instance1.123456789012.us-east-1.rds.amazonaws.com> -P 3306 –u < mymasteruser> -p

* For workbench login with username and password of the RDS.

**CLOUD WATCH:**

* Aws cloud watch is the component of the aws services that provides monitoring for aws resources and customer running applications in aws infrastructure,
* Cloud watch enables real-time monitoring of aws resources like ec2 instances, ebs volumes, load balancers and RDS databases, it provides automatic metrics for CPU utilization, latency and request count,
* Users can stipulate additional metrics to be monitored such as memory usage, transaction volumes or error rates, users can access cloud watch functions through an API, command line tools, one of the aws sdk (software development kit) or the aws management console,
* Cloud watch interface provides current statistics that can be viewed in graph format, users can set notifications called alarms to be sent something being monitoring as specified threshold, the app can detect unused or underused ec2 instances, gain system-wide visibility into resource utilization, application performance and operational health,
* Cloud watch related aws services **sns(simple notification service)** manages delivery or sending of calls to subscribing endpoints or clients,ec2 auto scaling, **cloud trail:** it monitor the calls made by aws cloud watch API for your account, aws cli and other services, cloud trail logging on cloud watch writes log files to the s3 bucket that you specified when to configured cloud trail, **Iam:** it is also helps you securely access to aws resources, who can access the resources(authentication)
* **What is cloud front?**

Amazon CloudFront is a service that speeds up transfer of your static and dynamic web content such as HTML files, IMAGE files., etc., CloudFront delivers your particulars thru worldwide data centers named Edge Locations.