**LIFT AND SHIFT PROJECT**

**Problem**

Let's we have a datacenter which runs variety of Application services such as Tomcat, LAMP Stack or DB services such as Postgre, Oracle running on Physical/Virtual machines. Related with this services there are several teams managing the architecture. This cause; complex management, scaling problem and high Opex and Capex. Besides that the processes are manual and time consuming.

**Solution**

Cloud computing is the solution for this but we need to lift and shift our on-site deployment. (IAAC - for automation)

**AWS Services**

* EC2 Instances (VMs for TOMCAT, RABBITMQ, MEMCACHED, MYSQL)
* ELB (For replacement of NginX LB)
* Autoscaling
* S3/EFS Storage (Shared Storage)
* Route 53 (DNS)
* ACM

**On-premise Topology**

1 - When user try to the app over URI or IP we will redirect to traffic to Load Balancer.

2 - NginX is a web service just like Apache httpd and it's commonly use to create the load balancing.

3 - When request comes, NginX is going to route the request to the Tomcat.

4 - Apache Tomcat is a Java Web Application Service. If your web application is written in Java, Tomcat is a very famous service to host it.So the application written by the developer will be sitting in this. IF you need external shared storage you can use NFS servers.

5 - Rabbit MQ is a message broker or also called a queuing agent to connet to applications together. You can stream data from this.

6 - The application accessed by the users and the user will login with credentials. When this happends the application will run an SQL query to access the user information stored in MySQL Database.

7 - Before the credentials goes to the MySQL database the request will go to memcache service. Memcache is a database caching so it will be connected to MySQL server. MySQL will store the user information when the first time the request comes to the database. It will be sent from MySQL to Tomcat and then it will be cached to Memcache server. When the following requests come, the cached data will be served by browser cache.

**Cloud Topology**

1 - DNS 53 will direct the traffic over HTTPS to our ELB. Since we will need a Domain name, ACM will be use to validate the domain.

2 - ELB will direct the traffic to tomcat instances over the port 8080.

3 - We need EC2 instances for Tomcat, Memcache, Rabbit MQ, MySQL

We also need 3 sequrity group to allow communication between services.

4 - We need Amazon S3 bucket to store software artifact

A diagram of a computer flowchart

Description automatically generated

**Flow Of Execution**

1. Login to AWS Account
2. Create Key Pairs
3. Create Security Groups
4. Launch Instances with user data [BASH SCRIPTS]
5. Update IP to name mapping in route 53
6. Build application from the source code
7. Upload to S3 Bucket
8. Download artifact to Tomcat EC2 Instance
9. Setup ELB with HTTPS [Certificate from ACM]
10. Map ELB Endpoint to website name in GoDaddy DNS
11. Verify

**1 – Validate Your Domain**

To prove that we are the owner of the domain, we need to validate the domain on AWS. Go to AWS > Certificate Manager (ACM) > Request Certificate > Request Public Certificate and fill your domain information. After this point you will see a Certificate in Pending Validation status which ha a CNAME name and CNAME value variables. For CNAME name, remove the domain part at the end and for the value copy the whole value.

Go to your domain name provider site and create a new DNS record. Paste the CNAME name and value of your domain. In a while, in AWS you will see that the status of certificate is Issued.

**2 – Create Security Groups**

For communication between layers, we will create three Security Groups as follows;

* SG\_ALB\_PRJ3 (allow inbound 443 from any)
* SG\_Tomcat\_PRJ3 (allow inbound 8080 from SG\_ALB\_PRJ3)
* SG\_Backend\_PRJ3 (allow inbound 3306 from SG\_Tomcat\_PRJ3 for MySQL)
* SG\_Backend\_PRJ3 (allow inbound 11211 from SG\_Tomcat\_PRJ3 for Memcached)
* SG\_Backend\_PRJ3 (allow inbound 5672 from SG\_Tomcat\_PRJ3 for Rabbit MQ)
* SG\_Backend\_PRJ3 (allow all traffic from SG\_Backend\_PRJ3 for Backend Communication)

Additionally we need to add access to these 2 SGs from our IP over the ports 8080 and 22.

* SG\_Tomcat\_PRJ3 (allow inbound 8080 and 22 from my-ip)
* SG\_Backend\_PRJ3 (allow inbound 8080 and 22 from my-ip)

If you don’t have static IP, you may always be sure that your IP is still same.

**3 – Create Key-Pairs**

Now let’s create key-pairs (public and private key) for connecting to the instances. If your git repository is not private, then add the key-pair as expection. Prod\_Key\_PRJ3 created.

**4 – Initiate the VMs** with following properties;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Database | Memcached | RabbitMQ | Tomcat |
| Hostname | db01 | mc01 | rmq01 | app01 |
| AMI | AlmaLinux Os9 | AlmaLinux Os9 | AlmaLinux Os9 | Ubuntu 22.04 TLS |
| Instance Type | t2.micro | t2.micro | t2.micro | t2.micro |
| Key-pair | Prod\_Key\_PRJ3 | Prod\_Key\_PRJ3 | Prod\_Key\_PRJ3 | Prod\_Key\_PRJ3 |
| Security Group | SG\_Backend\_PRJ3 | | | SG\_Tomcat\_PRJ3 |
| User Data | mysql.sh | mamcache.sh | rabbitmq.sh | tomcat\_ubuntu.sh |

Nginx was used for Load Balancing in previous projects but in this we will use AWS ALB for http/https load balancing. That is why we don’t have a VM for Nginx.

**5 – SSH to Servers**

Let’s ssh to the servers and check the configurations. If you don’t know how to connect to the server you can always check **ssh client connection** properties as follow.

A screenshot of a computer

Description automatically generated

To ssh linux servers created with this AMI we need to type;

* **ssh -i Prod\_Key\_PRJ3.pem root@<PublicIP>**

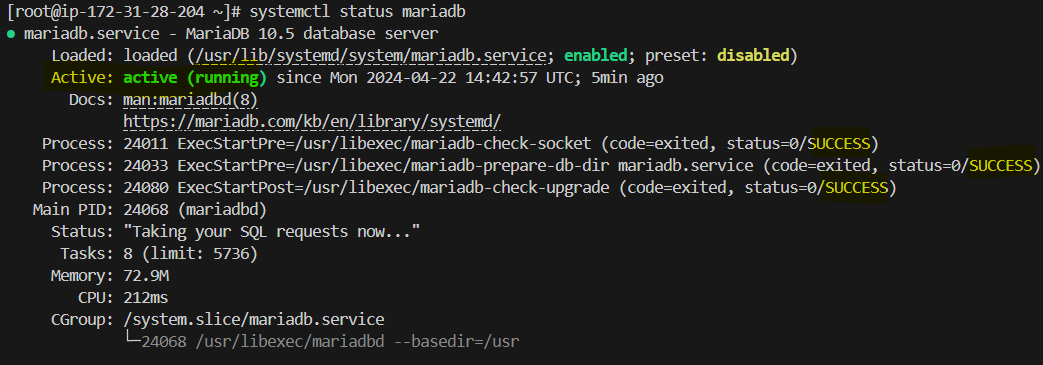
A computer screen with white text

Description automatically generated

Let’s check whether the configuration scripts have done their job properly.

* **systemctl status mariadb**

As you can see the DB service is working properly and the DB table is in order.

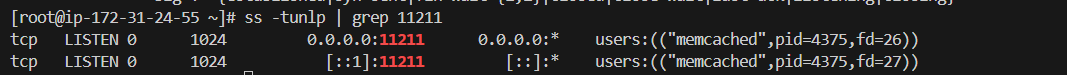


A screenshot of a computer

Description automatically generated

The ss (socket statistics) command is a powerful tool in Linux used for examining sockets. 11211 is default port for memcache server. As seen below, the server is listening the port.

* **ss -tunlp | grep 11211**



Rabbitmq is working as expected.

* **systemctl status rabbitmq-server**

A computer screen with white text

Description automatically generated

Finall we will check the application server (Tomcat) with the following command.

* **systemctl status tomcat9**

A computer screen with white text

Description automatically generated

Additionaly, we may check the /var/lib/tomcat9 directory and control the webapps which is the home folder for tomcat.



**6 – Route 53 DNS Configuration**

Configure Amazon Route 53 for private DNS as follows.

A screenshot of a computer screen

Description automatically generated

The important thing is, the following three server will only be reachable inside of the system. So DNS must resolve their private IP adresses. Enter A records as follows.



We only need these three because App Server (Tomcat) is going to connect to these backend services.

**7 – Build and Deploy Artifacts**

We need to use Git Bash. To change default shell to git, press Ctrl + Shift + P in Visual Studio Code then write Terminal Select Default Profile. The following options will appear. The select Git Bash.

A screenshot of a computer

Description automatically generated

Before changing when we start a terminal, it was starting a powershell but now we have git bash terminal.

A black screen with white text

Description automatically generated

In my IDE, maven and aws cli are already installed. To build artifacts and upload them to AWS S3, we need them to be installed. Do not forget to add domain names of hosts to the file.

A screenshot of a computer

Description automatically generated

Then run mvn install to build artifacts.

A black screen with white text

Description automatically generated

**NOTE:** Artifacts is designed to simplify package management in the software development process. Artifacts provides storage, management and sharing of packages. Be sure you are under the same directory with required files before creating artifacts. Be aware of output file named vprofiel-v2.war.

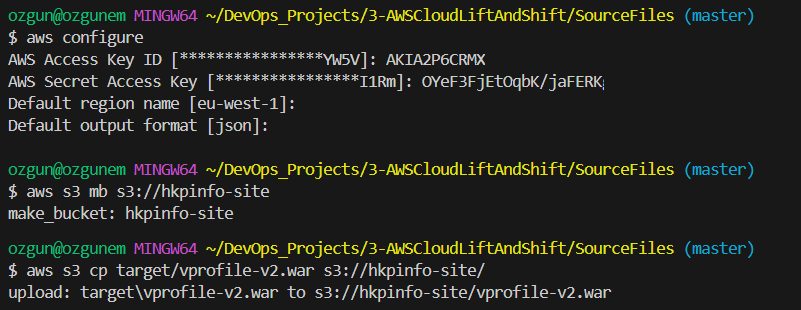
**8 – IAM User for S3 and Upload The Artifacts to S3**

We create an IAM user named S3admin with the s3fullaccess policy and create an access key under it. The access key will be use to connect AWS over CLI. When creating this access key, we select the CLI for the use case. Because with this we will provide CLI access to AWS from our local machine.

A screenshot of a computer program

Description automatically generated

Now let's go back to Visual Studio and create an S3 bucket where we will later upload our artifacts via CLI.



To better understand the folders, this is the directory that we are copying.

A screenshot of a computer

Description automatically generated

Now go back to AWS S3 service and check whether you will see the files or not. It looks like as follows.

A screenshot of a computer

Description automatically generated

**9 – Tomcat Server & Artifacts**

Tomcat server needs to connect to the S3 bucket. To give access, we need a IAM user with the s3fullaccess policy. While creating role, the trusted entity must be AWS Service and Use Case should be EC2. Thus, Tomcat server won’t need key to access it.

A screenshot of a computer

Description automatically generated

To test the connection, SSH to Tomcat server and install the awscli.

* **apt install awscli -y**

When we list aws s3, we should see the previously created bucket.

root@ip-172-31-19-84:~# aws s3 ls

2024-04-22 19:19:53 hkpinfo-site

Copy the artifacts in the S3 buckets to Tomcat’s /tmp folder.

root@ip-172-31-19-84:~# aws s3 cp s3://hkpinfo-site/vprofile-v2.war /tmp/

download: s3://hkpinfo-site/vprofile-v2.war to ../tmp/vprofile-v2.war

Now we need to deploy the files on Tomcat server. To do this first stop tomcat9 service then remove default webapp files. Then copy the artifact files to default path. Finally start the tomcat9 service again.

root@ip-172-31-19-84:~# systemctl stop tomcat9

root@ip-172-31-19-84:~# rm -rf /var/lib/tomcat9/webapps/ROOT

root@ip-172-31-19-84:~# cp /tmp/vprofile-v2.war /var/lib/tomcat9/webapps/ROOT.war

root@ip-172-31-19-84:~# systemctl start tomcat9

root@ip-172-31-19-84:~# ls /var/lib/tomcat9/webapps/

ROOT ROOT.war

Below, you can see the artifact file.

A screenshot of a computer program

Description automatically generated

**10 – Load Balancer ve DNS**

To create a Load Balancer, we must first create a target group. Since the load balancer is in front of the tomcat servers and needs to monitor the 8080 port, we set the port as 8080.

A screenshot of a computer

Description automatically generated

Since we will be monitoring HTTP/HTTPS traffic, let's move on to the load balancer definition by selecting ALB (Application Load Balancer).

A screenshot of a computer

Description automatically generated

A computer screen shot of a person

Description automatically generated

Since the ALB will be entry point for incoming traffic, in our domain we should create a CNAME entry with target ALB.

A screenshot of a computer

Description automatically generated

To check the result, you can try to access to the domain over a browser.

A screenshot of a login screen

Description automatically generated

**11 – Autoscaling for Tomcat Instances**

To autoscale instances we need to create image of the reference server, in this scenario App01 server. After that we need to create launch template as a pre-requisite of Autoscaling. Finally we will follow the autoscaling wizard and define the autoscaling group. There are several parameters depends on your choice and requirements but this is a very straight foward procedure.