EC2 Lab

Labs

1. Launch an EC2 instance (Amazon Linux, t2.micro). Connect to it via **EC2-connect** option from your browser. Install **Apache web server** on it. Access this public website from a browser. You may use your mobile to access it as well.
2. Stop this EC2 instance and then create an AMI of it. Launch another instance from the AMI and check if all your changes persist? (i.e. webserver installation).
3. After completing this, **terminate all** the EC2 instances. Also, deregister all the AMIs and delete the snapshots & volumes from your account.

1. Under **Search for services**, type in and then click **EC2**.
2. In the left-hand **EC2 Dashboard** navigation menu, click **Instances**.
3. Click on the **Launch Instances** button.
4. Under Step 1, click **Select** against Amazon Linux 2 AMI (HVM).
5. Under Step 2, select **t2.micro** and click on the Next button at the bottom.
6. Under Step 3, select **Default** for **Network.**
    a. Select any **Subnet** in the Subnet dropdown.
    b. For Auto-assign Public IP, select **Enable**.
    c. Click on the Next button at the bottom.
7. Under Step 4, click on the Next button at the bottom.
8. Under Step 5, add a tag with key = Name & value = Test Instance. Click on the Next button at the bottom.
9. Under Step 6, Create a new Security Group and it should have rules for port 22 & 80 incoming from anywhere (0.0.0.0/0). Click on the Review & Launch button at the bottom.
10. Under Step 7, Click on Launch button.
11. **Create a new key pair**, give it a name and download it. After that click on the **Launch Instances** button.
12. Select the launched instance and press on the **Connect** button. Execute the following command after you have SSHed into the instance (to install the web server and then start the service).

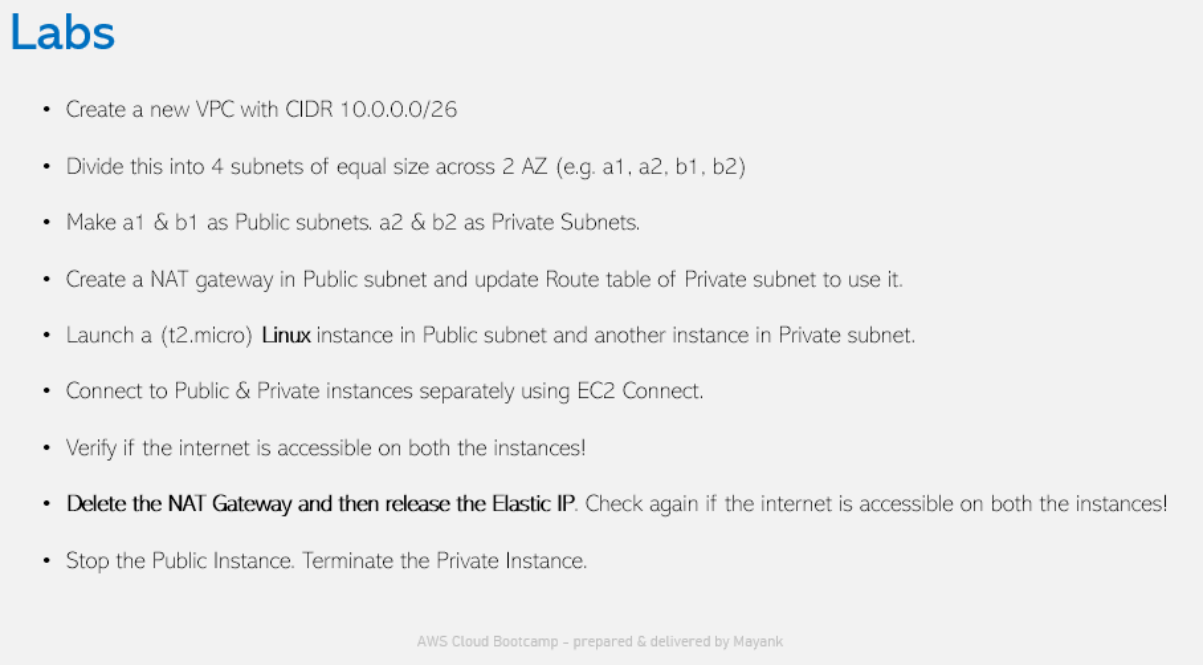
```
sudo yum install -y httpd
sudo systemctl enable httpd
sudo service httpd start
```

13. In a browser window, try to access the website using the EC2's Public IP or Public Hostname (**on port 80**). Ensure that you use **the HTTP url**.
14. Stop this instance and then create an AMI of it. Select the instance and explore the **Actions > Image & Templates > Create Image** button at the top.
15. Navigate to **AMIs** from the left hand menu.
16. Once the AMI is **Available**, select it and launch another instance from the AMI using the button given above (after this, follow the above steps **5 to 11**).
17. Check if all your changes persist in the new EC2 instance? (i.e. Webserver installation). You can check this by opening the new instance's Public IP/hostname from a browser.
18. After completing the above steps, **terminate** all the EC2 instances. **It will disappear**

from the console after an hour.

19. Look at the menu items on the left side for **AMI, Volumes** & **Snapshots.**
20. **Deregister** all the AMIs.
21. **Delete** all the snapshots from your account.
22. **Delete** all the volumes from your account.

VPC Lab



VPC & Subnets

In this section, we will create a new VPC and subnets in it.

1. Log in to the AWS Console using the provided **Username**, **Password**.
   Note: Now and throughout this lab, ensure that in the upper-right of the screen, **N. Virginia** (US East) is your selected region.
2. Under **Search for services**, type in and then click **VPC**.
3. In the left-hand **VPC Dashboard** navigation menu, click **Your VPCs**.
4. Click on the **Create VPC** button.
5. On the **Create VPC** page:
   a. For **Name tag**, provide `MyProdVPC`.
   b. For **IPv4 CIDR block**, provide `10.0.0.0/26`.
   c. Scroll down and click on the **Create VPC** button.
6. In the left-hand **VPC Dashboard** navigation menu, click **Subnets**.
7. Click on the **Create subnet** button.
8. On the **Create subnet** page:
   a. For **VPC ID**, choose **MyProdVPC**.
   b. Under **Subnet settings** > **Subnet 1 of 1**, for **Subnet name** provide `public-subnet-a`.

    c. For **Availability Zone**, choose **us-east-1a**.
    d. For **IPv4 CIDR block**, provide `10.0.0.0/28`.
    e. Click on the **Add new subnet** button.
    f. Under **Subnet 2 of 2**, for **Subnet name** provide `private-subnet-a`.
    g. For **Availability Zone**, choose **us-east-1a**.
    h. For **IPv4 CIDR block**, provide `10.0.0.16/28`.
    i. Click on the **Add new subnet** button.
    j. Under **Subnet 3 of 3**, for **Subnet name** provide `public-subnet-b`.
    k. For **Availability Zone**, choose **us-east-1b**.
    l. For **IPv4 CIDR block**, provide `10.0.0.32/28`.
    m. Click on the **Add new subnet** button.
    n. Under **Subnet 4 of 4**, for **Subnet name** provide `private-subnet-b`.
    o. For **Availability Zone**, choose **us-east-1b**.
    p. For **IPv4 CIDR block**, provide `10.0.0.48/28`.
    q. Click on the **Create subnet** button at the bottom of the page.
9. You should see a success message for subnet creation and the two subnets should be visible now.

Internet Gateway

In this section, we will create an Internet Gateway for our VPC. This will enable connectivity of our VPC to the Internet.

1. In the left-hand **VPC Dashboard** navigation menu, click **Internet Gateways**.
2. Click on the **Create internet gateway** button.
3. For **Name tag**, provide `prod-igw`.
4. Click on the **Create internet gateway** button at the bottom.
5. Upon successful creation of Internet Gateway, select **prod-igw** and click on **Actions** > **Attach to VPC**.
6. For **Available VPCs**, select **MyProdVPC**.
7. Click on the **Attach internet gateway** button.
8. You should see a success message for this action.

Public and Private Route Tables

1. In the left-hand **VPC Dashboard** navigation menu, click **Route Tables**.
2. Click on the **Create route table** button.
3. For **Name - optional**, provide `public-rt`.
4. For **VPC**, choose **MyProdVPC**.
5. Click on the **Create route table** button at the bottom.
6. Once the **public-rt** is created successfully, in the **Routes** tab, click on the **Edit routes** button.
7. Under **Edit routes**, click on the **Add route** button.
8. For **Destination**, provide `0.0.0.0/0`.
9. For **Target**, choose **Internet Gateway** and select **prod-igw**.
10. Click on the **Save changes** button.
11. Under the **Subnet associations** tab, click on the **Edit subnet associations** button.
12. Check both the **public subnets** and click on the **Save associations** button.

The 2 public subnets now **have access to the internet** via Internet Gateway **through this route table**.

1. In the left-hand **VPC Dashboard** navigation menu, click **NAT Gateways**.
2. Click on the **Create NAT Gateway** button.
3. For **Name - optional**, provide **myNAT**.
4. For Subnet, select **one of the two public subnets** from **your VPC**.
5. Click on the **Allocate Elastic IP** button.
6. Scroll down and click on the **Create NAT Gateway** button.
7. Your NAT Gateway will become available in a few minutes.
8. In the left-hand **VPC Dashboard** navigation menu, click **Route Tables**.
9. Click on the **Create route table** button.
10. For **Name - optional**, provide `private-rt`.
11. For **VPC**, choose **MyProdVPC**.
12. Click on the **Create route table** button at the bottom.
13. Once the **private-rt** is created successfully, in the **Routes** tab, click on the **Edit routes** button.
14. Under **Edit routes**, click on the **Add route** button.
15. For **Destination**, provide `0.0.0.0/0`.
16. For **Target**, choose **NAT Gateway** and select **myNAT**.
17. Click on the **Save changes** button.
18. Under the **Subnet associations** tab, click on the **Edit subnet associations** button.
19. Check both the **private subnets** and click on the **Save associations** button.
20. This route table will help the instances in the private subnet to utilize the NAT Gateway to access the internet.

This part gets completed with **successful creation of both route tables and adding the required routes**.

Launch a Public Instance

We will now launch an EC2 instance in the Public Subnet.

1. Under **Search for services**, type in and then click **EC2**.
2. In the left-hand **EC2 Dashboard** navigation menu, click **Instances**.
3. Click on the **Launch Instances** button.
4. Under Step 1, click **Select** against Amazon Linux 2 AMI (HVM).
5. Under Step 2, select **t2.micro** and click on the Next button at the bottom.
6. Under Step 3, select **MyProdVPC** for **Network.**
    a. Select a **Public Subnet** in the Subnet dropdown.
    b. For Auto-assign Public IP, select **Enable**.
    c. Click on the Next button at the bottom.
7. Under Step 4, click on the Next button at the bottom.
8. Under Step 5, add a tag with key = Name & value = Public Instance. Click on the Next button at the bottom.
9. Under Step 6, Create a new Security Group and it should have rules for port 22 & 80 incoming from anywhere (0.0.0.0/0). Click on the Review & Launch button at the bottom.
10. Under Step 7, Click on Launch button.
11. **Create a new key pair**, give it a name and download it. After that click on the **Launch Instances** button.

Launch a Private Instance

We will now launch an EC2 instance in the Private Subnet.

1. Under **Search for services**, type in and then click **EC2**.
2. In the left-hand **EC2 Dashboard** navigation menu, click **Instances**.
3. Click on the **Launch Instances** button.
4. Under Step 1, click **Select** against Amazon Linux 2 AMI (HVM).
5. Under Step 2, select **t2.micro** and click on the Next button at the bottom.
6. Under Step 3, select **MyProdVPC** for **Network.**
    a. Select a **Private Subnet** in the Subnet dropdown.
    b. For Auto-assign Public IP, select **Disable**.
    c. Click on the Next button at the bottom.
7. Under Step 4, click on the Next button at the bottom.
8. Under Step 5, add a tag with key = Name & value = Private Instance. Click on the Next button at the bottom.
9. Under Step 6, Create a new Security Group and it should have rules for port 22 & 80 incoming from anywhere (0.0.0.0/0). Click on the Review & Launch button at the bottom.
10. Under Step 7, Click on Launch button.
11. **Create a new key pair**, give it a name and download it. After that click on the **Launch Instances** button.

Final Testing:

- Upload the PEM file for your private instance to an S3 bucket (you can create a new bucket).
- Click on the bucket name, click on the **Permissions** tab and navigate to the **Block public access** section. Edit it and turn it **Off**.
- Then, make this file public and copy its public URL. (select the file, go to **Actions** and click on **Make public via ACL).** Click on the file to get its **Object URL.** This is the public URL for your file.
- Now, connect to the Public Instance and access google.com. You have to select the Public Instance and click on the **Connect** button. Then, click **Connect** again. (You can execute "`wget https://google.com`")
- You can download the PEM file (of private instance) on the Public instance using this command "`wget <public_URL>`"

- After that SSH into **Private Instance** and try to access google.com. See the following instructions:

```
chmod 400 <private_instance.pem>
ssh -i "16Nov2021.pem" ec2-user@<private-ip-of-private-instance>
```

- Verify if the internet is accessible on both the instances! (You can execute "`wget https://google.com`").
- Now, **delete the NAT Gateway** and then release the Elastic IP (navigate from the left hand menu to Elastic IP, you will have to wait for a few minutes). Check again if the internet is accessible on both the instances!

- The private instance should not be able to access the internet now. Public instance should still be able to access the internet via IGW.

- **Now, stop the Public Instance and terminate the Private Instance.**

RDS Lab

Labs

- Launch a Multi-AZ RDS (**PostgreSQL**) instance in private subnets of your VPC. (type – db.t3.micro, Storage – GP SSD – 30GB)
- Once it is ready, connect it via the Public instance you had launched earlier.
- Check the private IP of the RDS instance serving you. Make a note of it.
  - You can use "ping –a <<RDS endpoint>>"
- Now, reboot your RDS **with failover option** and check the private IP of the RDS instance serving you. Make a note of this IP as well.
- Verify these in the ENI section (under EC2 Dashboard).
- Follow the instructions and use **psql** to access the RDS database.
  - Create a table and insert data into it.
- Use python code on your EC2 instance to access the data RDS table.

AWS Cloud Bootcamp - prepared & delivered by Mayank

Subnet Group

AWS gets to know the subnets in which the 2 RDS nodes should be launched (via Subnet Group).

1. Under **Search for services**, type in and then click **RDS**.
2. In the left-hand **RDS Dashboard** navigation menu, click **Subnet Groups**.
3. Delete the Subnet Groups (if there are any).
4. Click on the Create DB Subnet Group button.
5. Give the name **myRDSSubnetGroup**. Provide a **description**.
6. For VPC, choose MyProdVPC.
7. Select both the Private Subnets of **your VPC** in the section below.
8. Click on the Create button.

DB Security Group

This security group protects the traffic coming to the RDS instance.

1. Navigate to EC2 Dashboard > Security Groups.
2. Create a new Security Group for your RDS.
3. Give the name as **RDSsg**.
4. Choose the VPC as your **myProdVPC**.
5. In the Inbound Rules,
   a. Specify Port as 5432 and source as the **Security Group ID of your Public EC2 Instance**. You can find this by opening a new tab for EC2 instances.
6. Click on the Create button.

DB Instance

1. In the left-hand **RDS Dashboard** navigation menu, click **Databases**.
2. Click on the Create Database button.
3. Choose the Standard create radio-button.
4. Choose **PostgreSQL** engine type.
5. For Availability & Durability, select **Multi-AZ DB instance**.
6. Make note of the Master username.
7. Provide a strong Master password. Make note of this as well in a notepad.
8. For DB Instance class, choose **Burstable classes** and select **db.t3.micro**.
9. For Storage, choose **gp2**.
10. Under Connectivity, choose myProdVPC in the network.
11. Select the Subnet Group you created in the previous step.
12. For Security Group, **remove** the default security group. Add **RDSsg**.
13. Under Additional Configuration,
    a. For Initial database name, provide **mydb**.
    b. Uncheck Enable Performance Insights.
    c. Uncheck Enable Enhanced Monitoring.
    d. Uncheck Enable deletion protection.
14. Click on the **Create Database** button at the bottom.

Test the Database Connectivity

1. Once RDS is ready, connect it via the Public instance you had launched earlier.
2. Start the Public EC2 Instance and **Connect** to it.
3. Execute the following:

```
sudo yum upgrade -y
sudo amazon-linux-extras install postgresql10 -y
mkdir rds-lab
cd rds-lab
wget https://usaa-aws-resources.s3.amazonaws.com/rds-lab/database.ini
wget https://usaa-aws-resources.s3.amazonaws.com/rds-lab/query_postgres.py
wget https://usaa-aws-resources.s3.amazonaws.com/rds-lab/requirements.txt
wget https://usaa-aws-resources.s3.amazonaws.com/rds-lab/db_config.py
pip3 install boto3
pip3 install --upgrade -r requirements.txt
```

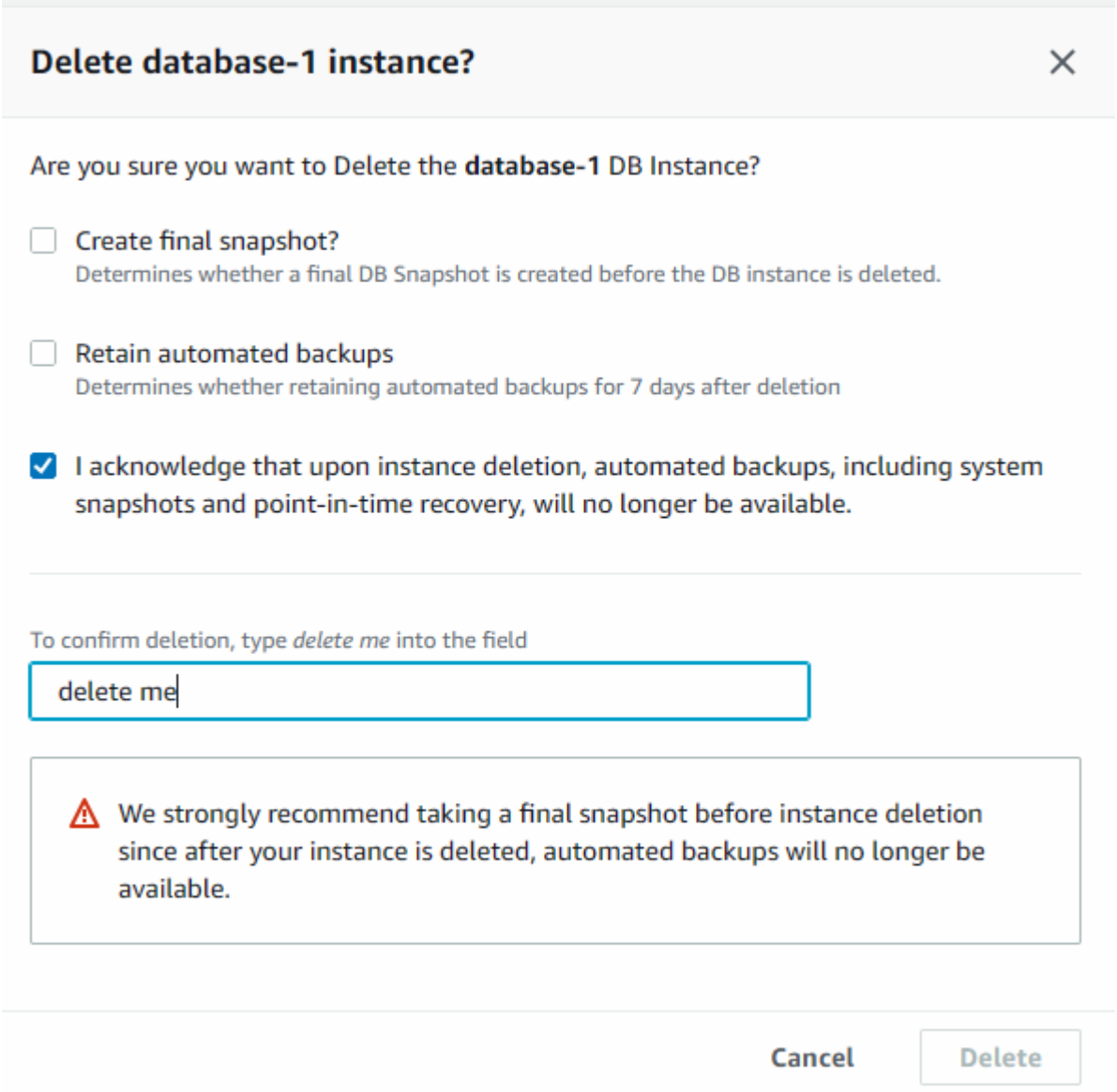
4. Now connect to RDS and create a table and insert data into it. Find the hostname from RDS details. Look for the **Endpoint** value. You need to remove the square brackets in the following command.

```
psql -h [HOSTNAME] -p [PORT] -U [USERNAME] -W -d [DATABASENAME]

create table film (title varchar(100) primary key, genre varchar (50),
release_year varchar (5));

insert into film (title, genre, release_year) values ('matrix 1', 'scifi',
1998), ('matrix 2', 'thriller', 2002);

select * from film;

#Exit from psql
\q
```

5. Update **database.ini** to show the info of your RDS. You can use vi editor. You will have to edit the hostname & password (provided you followed the instructions above).
6. Execute the following to connect to RDS via python code.
   `python3 ./query_postgres.py --instance master`
7. You should see the data from the table as part of the above python code execution. This part is completed now.

Test the Multi-AZ instance Failover

1. Check the private IP of the RDS instance serving you. Make a note of it.
   `ping -a <rds hostname>`
2. Now, reboot the RDS and choose **Reboot with Failover** option. **Wait for DB to be available again.** Check the private IP again and make a note of it.
   `ping -a <rds hostname>`
3. Go ahead and check these IPs in the EC2 dashboard > **Network Interfaces**.
4. **Stop the EC2 instance now.**
5. **Delete the RDS instance & Subnet Group now.**



DynamoDB Lab

Labs

- Create an **IAM role** having permissions to work with DynamoDB.
- Allocate this role to your EC2 instance.
- Use python code from your EC2 instance to:
  - Create DynamoDB table
  - Insert data into this table
- Verify the inserted data from Management Console (DynamoDB Dashboard).

1. Create a new **IAM role**. Name it **DDRole**.
2. It should be assumed by **EC2**.
3. Allocate it **AmazonDynamoDBFullAccess** policy.
4. Also, select **DND_permission_boundary** as the Permissions Boundary.
5. Allocate this IAM role to your Public EC2 instance. For this, select your public EC2 instance. Go to Actions > Security > Modify IAM Role.
6. **Connect** to the Public Instance and execute the following:

```
mkdir dynamo-lab
cd dynamo-lab
aws sts get-caller-identity

wget https://usaa-aws-resources.s3.amazonaws.com/dynamodb-lab/
create_table.py

wget https://usaa-aws-resources.s3.amazonaws.com/dynamodb-lab/items.json

wget https://usaa-aws-resources.s3.amazonaws.com/dynamodb-lab/
bulk_load_table.py
```

7. Use python code from your EC2 instance to:
   a. Create DynamoDB table
   b. Insert data into this table. Execute following:

```
python3 ./create_table.py

python3 ./bulk_load_table.py
```

Verify the inserted data from Management Console (**DynamoDB** Dashboard > **Tables**). Select the “**battle-royale**” table and click on the **View Items** button.

Delete the DynamoDB table and the IAM role. Also, detach the IAM role from the EC2 instance.

ElastiCache Lab

Labs

- Create required security group for ElastiCache cluster.
- Create a subnet group for ElastiCache cluster (with private subnets).
- Create a Multi-AZ ElastiCache Redis cluster in the private subnets of your VPC.
  - Choose cache.t2.micro
  - Multi-AZ
  - Number of Replicas: 1
- Check its connectivity using Python from the EC2 instance in the same VPC.

Subnet Group

1. Under **Search for services**, type in and then click **ElastiCache**.
2. In the left-hand menu, click **Subnet Groups**.
3. Delete the Subnet Groups (if there are any).
4. Click on the Create Subnet Group button.
5. Give the name **myRedisSubnetGroup**.
6. Give a description.
7. For VPC, choose MyProdVPC.
8. Select both the **Private Subnets** of your VPC in the section below.
9. Click on the Create button.

Security Group

1. Navigate to **EC2 Dashboard** > Security Groups.
2. Create a new Security Group for your RDS.
3. Choose the VPC as your **myProdVPC**.
4. Give the name as **Redissg**.
5. In the Inbound Rules,
   a. Specify Port as **6379** and source as the **Security Group ID of your Public EC2 Instance**.

Redis Cluster

1. In the left-hand navigation menu of **Elasticache Dashboard**, click **Redis**.

2. Click on the Create button.
3. For Name, provide **myRedis**.
4. For Node type, choose **cache.t2.micro**.
5. Number of replicas is 1.
6. Check **Multi-AZ**.
7. Select the Subnet Group you created in the previous step.
8. For the Security Group, select the security group created in the previous step. **Uncheck** the Default security group.
9. Scroll down and press the **Create** button. It will take 10-15 mins for cluster creation.

Test Connectivity

NOTE: Once the cluster is in **Available** state, pick the **primary endpoint** of your Redis Cluster and replace it in the highlighted part below.

From the EC2 instance, execute following:

```
pip3 install redis

$ python3

>>> import redis
>>> client = redis.Redis.from_url('redis://
your_redis_endpoint:6379')
>>> client.ping()

**True**
```

The connection is successful if you see **True** message.

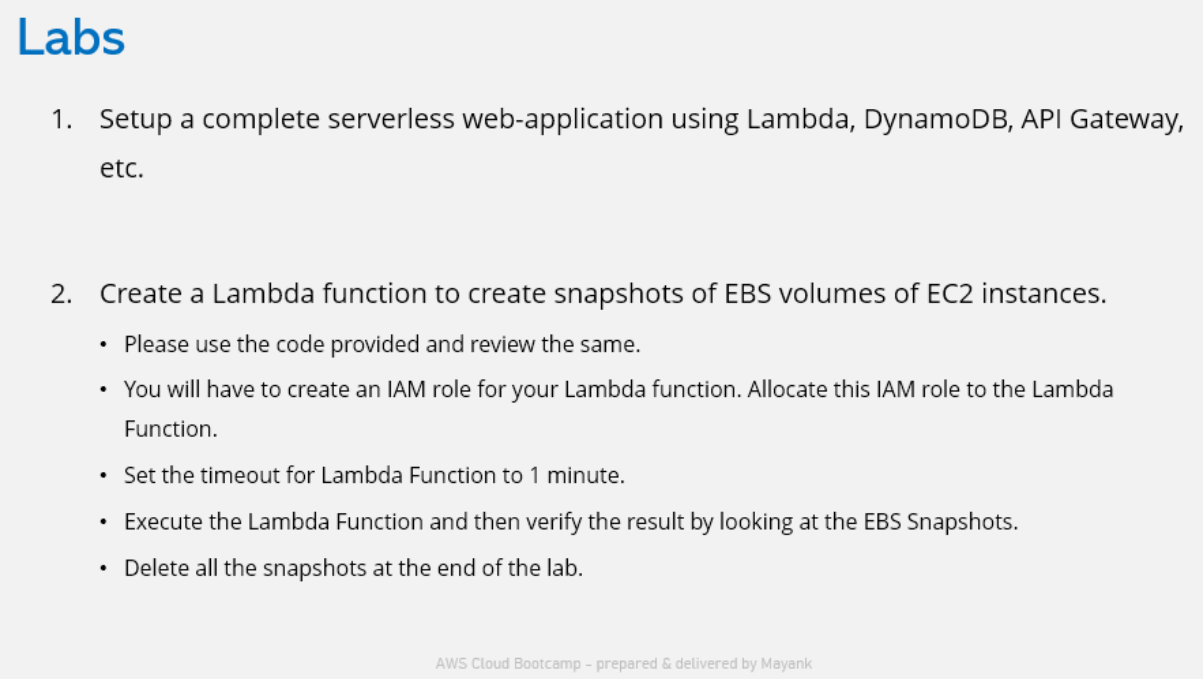
DELETE the ElastiCache Cluster after this. Also, delete the subnetgroup.

S3 Lab

1. Activate **CloudShell**
2. Execute “aws configure” and set the values:
    a. region = us-east-1
    b. Output format = json
3. Execute “aws sts get-caller-identity” and review the result.

4. Create new buckets and execute the following commands using AWS CLI - https://docs.aws.amazon.com/cli/latest/reference/s3/
5. Delete all the S3 buckets at the end of this lab.

Serverless Lab



Labs

1. Setup a complete serverless web-application using Lambda, DynamoDB, API Gateway, etc.

2. Create a Lambda function to create snapshots of EBS volumes of EC2 instances.
   - Please use the code provided and review the same.
   - You will have to create an IAM role for your Lambda function. Allocate this IAM role to the Lambda Function.
   - Set the timeout for Lambda Function to 1 minute.
   - Execute the Lambda Function and then verify the result by looking at the EBS Snapshots.
   - Delete all the snapshots at the end of the lab.

AWS Cloud Bootcamp - prepared & delivered by Mayank

Lab 1 - Detailed steps are given here. Make sure you delete all the resources after completing the lab.

Lab 2 -

1. Download these on your laptop:

https://usaa-aws-resources.s3.amazonaws.com/lambda-lab/lambdarole.json
https://usaa-aws-resources.s3.amazonaws.com/lambda-lab/automate.py

2. Create a new IAM policy using the json file downloaded above. Name it **SnapshotPolicy**. (If there is a policy with this name already, you can delete it).
3. Create a new IAM role. Name it **LambdaRole1**.
4. It should be assumed by **Lambda**.
5. It should be allocated **SnapshotPolicy** for permissions**.**
6. Also, select **DND_permission_boundary** as the Permissions Boundary.
7. This role will be used by the Lambda function.
8. Create a new Lambda function (Author from scratch) with **Python** runtime. Name it **SnapshotAutomation**. Press on the **Create Function** button.
9. Under the **Code** tab, Use the python code downloaded above and paste it there.
10. Press on the **Deploy** button.

11. Under the Configuration tab:
    a. Set the timeout period as 1 minute.
    b. Allocate **LambdaRole1** to this Lambda function.
12. Save and deploy it.
13. Verify the existing snapshots in EC2 dashboard.
14. **Test** the Lambda function and verify the created snapshots in the EC2 dashboard.
15. Delete the snapshots, IAM role, IAM policy & Lambda function at the end.

Terraform Lab

Labs

- AWS resource deployment using Terraform Code.
- Configure terraform in the Public EC2 instance and execute the steps.
- Verify the created resources in AWS Management Console.

1. Navigate to IAM Users > CloudLearner.
2. Navigate to the **Security Credentials** tab and click on the **Create access key** button (if you don't have it already with you).
3. Download and save the generated file on your laptop. This file contains secret access keys for your CloudLearner user.
4. Make sure you have removed the IAM role from your Public EC2 instance.
5. On the Public EC2 instance, configure your credentials for AWS CLI. Execute:

```
aws configure

Specify the Access Key ID & Secret Access Key from the file you
have.
Specify us-east-1 for region.
Specify json for output format.
```

6. Do the necessary setup on the Public EC2 instance to run Terraform code. Execute the following:

```
aws sts get-caller-identity

sudo yum install -y yum-utils
```

```
sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/
AmazonLinux/hashicorp.repo

sudo yum -y install terraform

terraform -help

terraform -install-autocomplete

source ~/.bashrc

mkdir tf-lab

cd tf-lab

wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/main.tf

ls

terraform init

terraform validate

terraform apply
```

7. Verify the created resource in the AWS management console.
8. Now destroy this resource. Execute:

```
terraform destroy
rm main.tf
```

---

2. Variables Testing
Delete the older main.tf file.

Download these files:

```
wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/02+-
+with+input+%26+output+file/variables.tf

wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/02+-
+with+input+%26+output+file/main.tf

wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/02+-
+with+input+%26+output+file/outputs.tf

wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/03+-
+with+tfvars/testing.tfvars
```

Execute: `terraform apply -var-file="testing.tfvars"`

---

3. Delete all the files from the tf-lab folder and then download this file.

`wget https://usaa-aws-resources.s3.amazonaws.com/tf-lab/vpc/main.tf`

Execute this once and create the VPC. Then, destroy it.

Afterwards, create a **variables.tf** file and a **test.tfvars** file.

You should specify the VPC CIDR and all the subnet CIDRs using these variable files.
At last, execute it by passing the CIDR values from the test.tfvars file.

Execute terraform destroy and delete the resources.

EKS Lab

Labs

- On your Public EC2 instance, configure AWS CLI with your credentials (generate it from IAM > CloudLearner).
- Install eksctl & kubectl
- Create an EKS Fargate cluster using eksctl.
- Create pods and target them to the correct Fargate profile.
- Access an AWS service from the pod.
- Delete the EKS cluster.

Important parts of this lab are:

- Creating the EKS cluster in the existing VPC.
- Creation of Fargate Profiles.
- Creation of Service Account to interact with AWS services.
- Launching a pod in the correct Fargate profile.
- Connecting to an AWS service from the pod.

---

1. Generate Access Keys for IAM user **CloudLearner** (if not done already).
2. Connect to your Public Instance and execute "aws configure"
3. Run "aws sts get-caller-identity" and verify that you are using IAM user CloudLearner.
4. Execute:

```
mkdir eks-lab

cd eks-lab/

wget https://usaa-aws-resources.s3.amazonaws.com/eks-lab/cluster-fargate-3.yaml

vi cluster-fargate-3.yaml
```

5. Update the VPC id and Private Subnet IDs & their **CIDRs** (**according to the VPC you have created in your account**) in this file and then save it. **Be careful about the AZs mentioned in the file.**
6. In a new tab, create a NAT Gateway in one of the Public Subnets. Assign Elastic IP to it as well.
7. Update the Private Route Table to have a route for this NAT Gateway (for target 0.0.0.0/0). All the routes in the private Route Table should show as **ACTIVE**. You can update the older 0.0.0.0/0 route itself.
8. Install kubectl - Installing kubectl - Amazon EKS (**look for Linux tab**)
9. Install eksctl - The eksctl command line utility - Amazon EKS (**look for Linux tab**)
10. Execute:

```
eksctl create cluster -f cluster-fargate-3.yaml
```

11. Open CloudFormation & EKS in new browser tabs to review the activities happening there.
12. Wait for the EKS cluster to be **ready**. Constantly review the progress in AWS Management Console. Check the **Fargate profiles** which get created as part of the EKS cluster. 2 Fargate Profiles should be created. (**Refer**: EKS Cluster > Configuration tab > Compute tab). It may take **20 to 30 minutes** for the complete cluster creation.
13. Once you get the success message for **Create Cluster** command, execute "`kubectl run nginx --image=nginx`" to create a pod.
14. Check the running pods using "`kubectl get pods --all-namespaces --output wide`"
15. Wait for this newly created pod to come to **Running** state. Execute above command again to check the progress. Once done, check the available **Nodes** in the EKS Cluster (in EKS Dashboard).
16. Execute "`kubectl create namespace dev`" to create a new namespace.

17. Execute "`kubectl run nginx-dev --image=nginx --namespace dev`" to create a new pod in the dev namespace
18. Check the running pods using "`kubectl get pods --all-namespaces --output wide`"
19. Wait for this newly created pod to come to **Running** state.
20. You can verify the running fargate nodes and their details by executing "`kubectl describe nodes`"
21. A **service account** needs to be created in EKS so that it could help pods to interact with AWS services (via an IAM role).
22. Execute:

```
eksctl utils associate-iam-oidc-provider --cluster fargate-cluster-3
--approve --region us-east-1

# make sure there are no additional spaces than what is mentioned
here
eksctl create iamserviceaccount --namespace dev --name ddb-sa-2 --
cluster fargate-cluster-3 --attach-policy-arn
arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess --approve --region
us-east-1
```

23. Check the details of the service account. Execute:

```
kubectl get sa ddb-sa-2 -o yaml --namespace dev
```

24. Execute following to create a new pod which uses the created service account:

```
wget https://usaa-aws-resources.s3.amazonaws.com/eks-lab/my-pod.yaml

#Open and review this yaml file. As this is part of dev namespace,
it will be placed in fp-dev fargate profile.

kubectl apply -f my-pod.yaml

kubectl get pods --all-namespaces --output wide
```

25. Verify the status of the pod using the above command. Once the pod is in **running** state, move ahead.
26. Execute "`kubectl exec -it my-pod --namespace dev -- bash`" to login to a pod.
27. In a new browser tab, navigate to **DynamoDB** dashboard and create 2 tables (names & attributes don't matter).

Following needs to be executed inside the pod.

1. Execute following:

```
apt-get update && apt-get install curl unzip less -y

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
```

```
unzip awscliv2.zip

./aws/install

aws sts get-caller-identity
# you should see the reference to the IAM role related to the
created service account.

aws dynamodb list-tables --region us-east-1
```

You should see the **names of DynamoDB tables** (which you created in the previous steps). The pod is able to interact with AWS service using the IAM role.

- You can now delete the EKS cluster.
- Execute "**`eksctl delete cluster --name fargate-cluster-3`**"

1. Delete the NAT Gateway.
2. Delete the RDS you had created.
3. Delete the ElastiCache cluster you had created.
4. Release all the Elastic IPs. (see VPC Dashboard > Elastic IP)

5. Delete the Public EC2 instance. Wait for 5 minutes.

6. **Delete the VPC** which you had created at the start of training.

7. Delete all the DynamoDB tables, S3 buckets, IAM roles, IAM users, IAM policies **you have created**.

8. Delete any other resource you had created in the AWS account.
