AWS 3-TIER ARCHITECTURE LAB

Silver Lining Cloud Operations Training

Session Date: *May 3rd, May 4th, 2021*

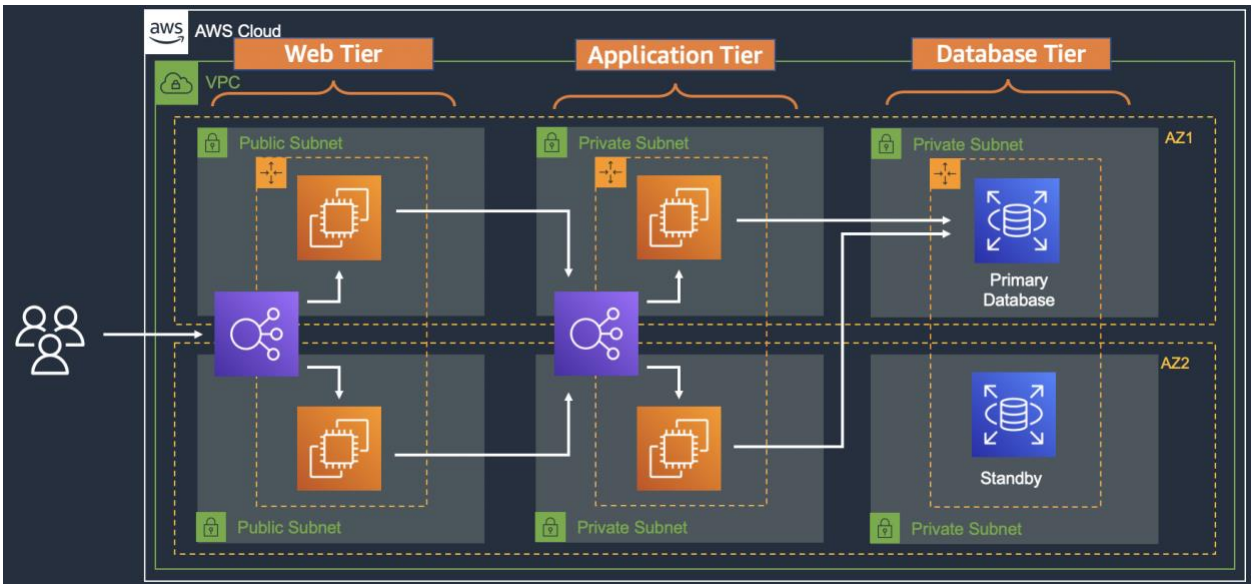
Instructors:
Shivani Sharma - Solutions Architect - awsshivs@amazon.com
David Kheyman - Solutions Architect - kheyman@amazon.com

Lab Description: This lab is a hands-on walk through of a three-tier web architecture in AWS. We will be creating the necessary network, security, app, and database components and configurations in order to run this architecture in an available and scalable manner.

Pre-requisites:

1. An AWS account. If you don't have an AWS account, follow the instructions here and click on "Create an AWS Account" button in the top right corner to create one.
2. IDE or text editor of your choice.

Architecture Overview:



In this architecture, a public-facing Application Load Balancer forwards client traffic to our web tier EC2 instances. The web tier is running Nginx webservers that are configured to serve a React.js website and redirects our API calls to the application tier's internal facing load balancer. The internal facing load balancer then forwards that traffic to the application tier, which is written in Node.js. The application tier manipulates data in an Aurora MySQL multi-AZ database and returns it to our web tier. Load balancing, health checks and autoscaling groups are created at each layer to maintain the availability of this architecture.

PART 0: SETUP

LEARNING OBJECTIVES:

- S3 Bucket Creation
- IAM EC2 Instance Role Creation

INSTRUCTIONS:

1. **S3 Bucket Creation**
    a. Navigate to the S3 service in the AWS console and create a new S3 bucket. You can also reuse any bucket, but make sure it is in the same region that you intend to use for this whole lab. This is where we will upload our code later.

2. **IAM EC2 Instance Role Creation**
    a. Navigate to the IAM dashboard in the AWS console
    b. Create an EC2 role with the following AWS managed policies. These policies will allow our instances to download our code from S3 and use Systems Manager Session Manager to securely connect to our instances without SSH keys through the AWS console.
        i. AmazonSSMManagedInstanceCore
        ii. CloudWatchLogsFullAccess
        iii. AmazonS3FullAccess

PART 1: NETWORKING AND SECURITY

LEARNING OBJECTIVES:

- Create an isolated network with the following components:
    - VPC
    - Subnets
    - Route Tables
    - Internet Gateway
    - NAT gateway

INSTRUCTIONS:

3. **VPC Creation**
    a. Navigate to the VPC dashboard in the AWS console.
    b. Create a new VPC and fill out the VPC Settings with a Name tag and a CIDR range of your choice.

    NOTE: *Make sure you pay attention to the region you're deploying all your resources in. You'll want to stay consistent for this lab.*

    NOTE: *Choose a CIDR range that will allow you to create at least 6 subnets.*

4. **Subnet Creation**
    a. Next, create your subnets by navigating to **Subnets** on the left side of the dashboard.
    b. We will need **six** subnets across **two** availability zones. That means that **three** subnets will be in one availability zone, and **three** subnets will be in another zone. Create each of the 6 subnets by specifying the VPC we created in part 1 and then choose a name, availability zone, and appropriate CIDR range for each of the subnets.

        NOTE: *It may be helpful to have a naming convention that will help you remember what each subnet is for. For example in one AZ you might have the following: Public-Subnet-AZ-1, Private-Subnet-AZ-1, Private-DB-Subnet-AZ-1*

        NOTE: *Remember, your CIDR range for the subnets will be subsets of your VPC CIDR range.*

5. **Internet Gateway**
    a. In order to give our VPC internet access we will have to create and attach an Internet Gateway. On the left hand side of the current dashboard, select **Internet Gateway**. Create your internet gateway by simply giving it a name.
    b. After creating the internet gateway, attach it to your VPC that you create in step 1.

6. **NAT Gateway**
    a. In order for our instances in the private subnet to be able to access the internet they will need to go through a NAT Gateway. For high availability, you'll deploy one NAT gateway in each of your **public** subnets.
    b. Navigate to **NAT Gateways** on the left side of the current dashboard. Click **Create NAT Gateway**, and fill in the **Name,** choose one of the **public subnets** you created in part 2, and then allocate an Elastic IP.
    c. Repeat step a in the other subnet.

7. **Route Tables**
    a. In order to control where our network traffic is directed, we need to create route tables. Navigate to **Route Tables** on the left side of the VPC dashboard.
    b. First, let's create one route table for the *public subnets*. After creating the route table, add a route that directs traffic from the VPC to the internet gateway. In other words, for all traffic *destined* for outside the VPC, add an entry that directs it to the internet gateway as a *target*.
    c. Edit the **Subnet Associations** of the route table and select the two public subnets you created in part 2.
    d. Now create 2 more route tables, one for each availability zone. These route tables will each route traffic destined for outside the VPC to one of the NAT gateways instead of the internet gateway.
    e. Once the route tables are created, add the appropriate subnet associations for each of the private subnets.

8. **Security Groups**
    a. Security groups will tighten the rules around which traffic will be allowed to our load balancers and instances. Navigate to **Security Groups** on the left side of the dashboard, under **Security**.
    b. The first security group you'll create is for the internet facing load balancer. Add an inbound rule to allow **HTTP** type traffic for **your IP**.
    c. The second security group you'll create is for the public instances in the web tier. After typing a name and description, add an inbound rule that allows **HTTP** type traffic from your internet facing load balancer security group you created in the previous step. This will allow traffic from your public facing load balancer to hit your instances. Then, add an additional rule that will allow **HTTP** type traffic for **your IP**. This will keep access to your app restricted while allowing you to test.
    d. The third security group will be for our internal load balancer. Create this new security group and add an inbound rule that allows **HTTP** type traffic from your public instance security group. This will allow traffic from your web tier instances to hit your internal load balancer.
    e. The fourth security group we'll configure is for our private instances. After typing a name and description, add an inbound rule that will allow **TCP** type traffic on port **4000** from the **internal load balancer security group** you created in the previous step. This is the port our web tier is running on and allows our internal load balancer to forward traffic to our private instances . You should also add another route for port 4000 that allows your IP for testing.
    f. The fifth security group we'll configure protects our private database instances. For this security group, add an inbound rule that will allow traffic from the private instance security group to the MYSQL/Aurora port (3306).

PART 2: DATABASE DEPLOYMENT

LEARNING OBJECTIVES:

- Deploy Database Layer

INSTRUCTIONS:

1. **Subnet Group Creation**
    a. Navigate to the RDS dashboard in the AWS console.
    b. Create a subnet group containing the **database private subnets** that you created in part 1, step 3.

2. **Database Creation**
    a. Create a **privately accessible**, **provisioned Amazon Aurora MYSQL** database with **Multi-AZ deployment**. Make sure to use the VPC, subnet group and database security group that we created earlier. We will be using password authentication, so note down the username and password for your database.
    b. When your database is provisioned, you should see a reader and writer instance. Note down the writer endpoint for your database for use later as well.

PART 3: APP TIER INSTANCE DEPLOYMENT

LEARNING OBJECTIVES:

- Create App Tier Instance
- Configure Software Stack
- Configure Database Schema
- Test DB connectivity

INSTRUCTIONS:

1. **Create App Layer Instance**
    a. Navigate to the EC2 console and launch a single **t.2 micro Amazon Linux 2** instance into one of the **private instance subnets** that we created in part 1 step 3. Configure the network, IAM role, and security group that we have created. You can keep the default storage amount. For ease, add a Name tag so you can identify that this is the App layer instance.
    b. Review your configurations before you launch the instance, and then choose to proceed without a key pair. We will be using Systems Manager Session Manager for instance access.

2. **Connect to App Layer Instance**
    a. Navigate to your list of running Ec2 Instances. When the instance state is running, connect to your instance by clicking the checkmark box to the left of the instance, and click the connect button on the top right corner of the dashboard.
    Select the Session Manager tab, and click connect.

    NOTE: *If you get a message saying that you cannot connect via session manager, then check that your instances can route to your NAT gateways and verify that you gave the necessary permissions on the IAM role for the Ec2 instance.*

    b. When you first connect to your instance like this, you will be logged in as ssm-user. For simplicity, let's switch to ec2-user by executing the following command:

    ```
    sudo -su ec2-user
    ```

    c. Let's take this moment to make sure that we are able to reach the internet via our NAT gateways. If your network is configured correctly up till this point, you should be able to ping the google DNS servers:

    ```
    ping 8.8.8.8
    ```

    You should see a transmission of packets. Stop it by pressing cntrl c.

NOTE: *If you can't reach the internet then you need to double check your route tables and subnet associations to verify if traffic is being routed to your NAT gateway!*

3. **Test Database Connectivity**
   a. Start by downloading the MySQL CLI:

```
sudo yum install mysql
```

   b. Initiate your DB connection with your Aurora RDS writer endpoint:

```
mysql -h change-to-your-rds-endpoint.rds.amazonaws.com -u
uname -p
```

   Type in your password when prompted. You should now be connected to your database.

   NOTE: *If you cannot reach your database, check your credentials and security groups.*

4. **Create Data**
   a. Create a database called **webappdb** with the following command using the MySQL CLI:

```
CREATE DATABASE webappdb;
```

   You can verify that it was created correctly with the following command:

```
SHOW DATABASES;
```

   b. Create a data table by first navigating to the database we just created:

```
USE webappdb;
```

   Then, create the following **transactions** table by executing this create table command:

```
CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL
AUTO_INCREMENT, amount DECIMAL(10,2), description
VARCHAR(100), PRIMARY KEY(id));
```

   Verify the table was created:

```
SHOW TABLES;
```

   c. Insert data into table for use/testing later:

```
INSERT INTO transactions (amount,description) VALUES
('400','groceries');
```

Verify that your data was added by executing the following command:

```
SELECT * FROM transactions;
```

When finished, just type **exit** and hit enter to exit the MySQL client.

5. **Configure App Layer Instance**
   a. The first thing we will do is update our database credentials for the app tier. To do this, open the **CloudOpsCharity3TierLab/cloud-ops-charity-app-layer/DbConfig.js** file that was provided to you in your favorite text editor. You'll see empty strings for the hostname, user, and password. Fill this in with the credentials you configured for your database, and save the file. Make sure you are using the **writer** endpoint of your database as the hostname.

   ***NOTE:** This is NOT considered a best practice, and is done for the simplicity of the lab. Moving these credentials to a more suitable place like Secrets Manager is left as an extension for this lab.*

   b. Upload the **cloud-ops-charity-app-layer** folder to the S3 bucket that you created in part 0.
   c. Go back to your SSM session. Now we need to install all of the necessary components to run our backend application. Start by installing NVM (node version manager)

```
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.38.0/install.sh | bash

source ~/.bashrc
```

   d. Next, install Node.js

```
nvm install node
```

   e. PM2 is a daemon process manager that will keep our node.js app running.

```
npm install -g pm2
```

   f. Now we need to download our code from our s3 buckets onto our instance:

```
cd ~/

aws s3 cp s3://BUCKET_NAME/cloud-ops-charity-app-layer/
app-layer --recursive
```

   g. Navigate to the app directory, install dependencies, and start the app with pm2.

```
cd ~/app-layer

npm install

pm2 start index.js
```

To make sure the app is running correctly run the following:

```
pm2 list
```

If you see a status of online, the app is running. If you see errored, then you need to do some troubleshooting. To look at the latest errors, use this command:

```
pm2 logs
```

NOTE: *If you're having issues, check your configuration file for any typos, and double check that you have followed all installation commands till now.*

h. Right now, pm2 is just making sure our app stays running when we leave the SSM session. However, if the server is interrupted for some reason, we still want the app to start and keep running. This is also important for the AMI we will create:

```
pm2 startup
```

After running this you will see a message similar to this:

```
[PM2] To setup the Startup Script, copy/paste the following command:
sudo env PATH=$PATH:/home/ec2-user/.nvm/versions/node/v16.0.0/bin
/home/ec2-user/.nvm/versions/node/v16.0.0/lib/node_modules/pm2/bin/pm2
startup systemd -u ec2-user –hp /home/ec2-user
```

Copy and paste the command you are given. After you run it, save the current list of node processes with the following command:

```
pm2 save
```

i. In order to test that your app tier works, you can temporarily associate this instance's subnet with the public route table, and then attach an Elastic IP to the instance since it has no public IP associated with it. An Elastic IP is a public IPv4 address that you can associate with your private instance in order to enable internet access.
   i. Navigate to the EC2 Dashboard and select **Elastic IP** on the left. Allocate another Elastic IP and name it **AppTierTest** so we can identify it.
   ii. Select the elastic IP, and select **Actions → Associate Elastic IP address**. Choose the app tier instance.
   iii. Navigate to the VPC dashboard. Select subnets on the left, and then select the private subnet your app tier instance is currently deployed in. Under actions click on edit route table associations and associate the subnet with the public route table.

iv. Navigate back to your Ec2 instances, select your app tier instance and in the details you'll see a public IP you can use to test. Plug the following in your browser:

```
http://[YOUR PUBLIC IP]:4000/transaction
```

OR

```
http://[YOUR PUBLIC IP]:4000/health
```

If you see data from your database, then everything is set up correctly.

v. Now, reset your route table association so this instance's private subnet is associated with the correct private route table. **This step is important to do or you may run into issues later.**

j. Congrats! Your app layer is fully configured and ready to go.

PART 4: INTERNAL LOADBALANCING AND AUTOSCALING

LEARNING OBJECTIVES:

- Create an AMI of our App Tier
- Create a Launch Configuration
- Configure Autoscaling
- Deploy Internal Load Balancer

INSTRUCTIONS:

1. **Create an AMI of the App Instance**
    a. Navigate to the EC2 dashboard in the AWS console.
    b. Select your App Tier instance and under **Actions** select **Image and templates → Create Image**. This will take a few minutes.

2. **Create Internal Facing Load Balancer**
    a. On the EC2 dashboard, click on **load balancers**. Create an internal facing application load balancer in the two availability zones that your **private** subnets are in. Use the security group we configured for your internal load balancer. This load balancer will listen on port 80.
    b. Create a new target group and set the port to **4000**. This is the port our node.js app is running on.
    c. For the health check, change the path to **/health.** In advanced health check settings override the traffic port and set it to **4000**.
    d. **Do not** register any targets for now and complete the creation of the load balancer.

3. **Create Launch Configuration**
    a. On the left side of the EC2 dashboard navigate to **Launch Configurations**.
    b. Create a Launch Configuration with the AMI you created in step 1, and specify the same details as your original EC2 instance. I.e., the same security group, EC2 role, instance type, etc.

4. **Create Autoscaling Group**
    a. On the left side of the Ec2 dashboard navigate to Autoscaling Groups.
    b. After giving your group a name, under Launch Template switch to **launch configuration** so we can use the configuration we just created in step 3.
    c. Under Configure Settings, configure the network with the VPC you created, along with both of the **private** subnets.
    d. Under advanced options we will attach our existing internal load balancer and target group.
    e. We want have 2 instances in our app tier running, so choose 2 for desired, min and max capacity. Skip the notifications for now, this can be configured later.

You should now have your internal load balancer and autoscaling groups configured correctly. You should see the autoscaling group spinning up 2 new app tier instances. If you wanted to test if this is working correctly, you can delete one of your new instances manually and wait to see if a new instance is booted up to replace it.

NOTE: *Your original app tier instance is excluded from the ASG so you will see 3 instances. You can delete your original instance that you took an AMI of.*

PART 5: WEB TIER INSTANCE DEPLOYMENT

LEARNING OBJECTIVES:

- Create Web Tier Instance
- Configure Software Stack

INSTRUCTIONS:

1. **Update Configs and Upload Code**
    a. Before we start configuring the web instances, open up the **nginx.conf** file that was provided to you in your favorite text editor. Scroll down to **line 58** and replace **[INTERNAL-LOADBALANCER-DNS]** with your internal load balancer's DNS entry. Then, upload this file and the **cloud-ops-charity-web-layer** folder to the s3 bucket you created for this lab.

2. **Create Web Tier Instance**
    a. Follow the same instance creation instructions we used for the App Tier instance, with the exception of the subnet. We will be provisioning this instance in one of our **public subnets.** Again, select the necessary network configuration, security groups, and same IAM role. Remember to auto-assign a public ip. We will also proceed without a key pair for this instance as well.

3. **Connect to Web Tier Instance**
    a. Follow the same steps you used to connect to the app instance and change the user to **ec2-user**. Test connectivity here via ping as well, though this instance should have regular internet connectivity:

```
sudo -su ec2-user

ping 8.8.8.8
```

4. **Configure Web Tier Instance**
    a. We now need to install all of the necessary components needed to run our front-end application. Again, start by installing NVM and node :

```
curl -o- https://raw.githubusercontent.com/nvm-
sh/nvm/v0.38.0/install.sh | bash

source ~/.bashrc

nvm install node
```

    b. Now we need to download our web tier code from our s3 bucket:

```
cd ~/

aws s3 cp s3://BUCKET_NAME/cloud-ops-charity-web-layer/
web-layer --recursive
```

    Navigate to the web-layer folder and create the build folder for the react app so we can serve our production code:

```
cd ~/web-layer

npm install

npm run build
```

    c. Nginx can do all sorts of things like load balancing, content caching etc, but we will be using it as a web server that we will configure to serve our application on port 80, as well as help direct our API calls to the internal load balancer.

```
sudo amazon-linux-extras install nginx1
```

    d. We will now have to configure Nginx. Navigate to the Nginx configuration file with the following commands:

```
cd /etc/nginx

ls
```

You should see an **nginx.conf** file. We're going to delete this file and use the one we uploaded to s3:

```
sudo rm nginx.conf

sudo aws s3 cp s3://BUCKET_NAME/nginx.conf .
```

Then, restart Nginx with the following command:

```
sudo service nginx restart
```

To make sure Nginx has permission to access our files execute this command:

```
chmod -R 755 /home/ec2-user
```

And then to make sure the service starts on boot, run this command:

```
sudo chkconfig nginx on
```

e. Now when you plug in the public IP of your web tier instance, you should see your website. If you have the database connected and working correctly, then you will also see the database working. You'll be able to add data. Careful with the delete button, that will clear all the entries in your database.

PART 6: EXTERNAL LOADBALANACER AND AUTOSCALING

LEARNING OBJECTIVES:

- Create an AMI of our Web Tier
- Deploy Internet Facing Load Balancer
- Create a Launch Configuration
- Configure Autoscaling

INSTRUCTIONS:

1. **Create an AMI of the Web Instance**
   a. Navigate to the EC2 dashboard in the AWS console.
   b. Select your Web Tier instance and under **Actions** select **Image and templates → Create Image**. This may take a few minutes.

2. **Create Internet Facing load balancer**
   a. Navigate to Load Balancers on the left side of the EC2 dashboard.
   b. Create an **internet facing application load balancer** with an **HTTP** listener on **port 80** in the two **public** subnets we created. Skip step 2 for now and configure your security groups. Select the security group we created for the internet facing load balancer.
   c. Create a new target group that directs traffic to port 80**.**
   d. Configure the health check path with **/health**.

e. **Do not** register any targets for now, and complete the creation of the load balancer.

3. **Create Launch Configuration**
    a. On the left side of the EC2 dashboard navigate to **Launch Configurations**.
    b. Create a Launch Configuration with the Web Tier AMI and specify the same details as your original EC2 instance. I.e., the same security group, EC2 role, instance type, etc. Make sure you enable public IPs for these instances.

4. **Create Autoscaling Group**
    a. On the left side of the Ec2 dashboard navigate to Autoscaling Groups.
    b. After giving your group a name, under Launch Template switch to **launch configuration** so we can use the configuration we just created in step 2.
    c. Under Configure Settings, configure the Network with the VPC you created, along with both of the **public** subnets.
    d. Under advanced options we will attach our existing **internet facing** load balancer and target group.
    e. We want have 2 instances in our app tier running, so choose 2 for desired, min and max capacity. Skip notifications for now- this can be configured later.

You should now have your internet facing load balancer and autoscaling groups configured. You should see the autoscaling group spinning up 2 new web tier instances. Plug in your internet facing load balancer DNS into your browser to see if your web tier in action. You should also be able to interact with your database.

NOTE: *Your original web tier instance is excluded from the ASG so you will see 3 instances. You can delete that instance if everything is working correctly.*

Congrats! You've Implemented a 3 Tier Web Architecture!

PRESENTATION PRACTICE

Being able to talk about the technology you used and architecture you built is a key technical skill. Practice walking through your demo out loud by yourself or present to a peer, and explain the architecture you created as well as the key advantages of this architecture like scalability and availability.

BONUS CHALLENGES

There are a few ways that we can extend this lab by doing the following:

1. Creating an alias for your internet facing load balancer so you can use a human friendly DNS name for your website.
2. Implementing a secure connection to the internet facing load balancer with SSL/TSL certificates and HTTPS listener, and additionally creating a secure end to end connection.
3. Add security services like AWS Web Application Firewall, or implement AWS Network Firewall for traffic inspection.
4. Move credentials from the configuration file to AWS Secrets Manager.
5. Configure Cloud Watch metrics.
6. Implement Multi-region failover.

Challenge yourself and take this lab one step further!

APPENDIX

DOCUMENTATION:

- NVM (Node Version Manager)
  - https://github.com/nvm-sh/nvm
- Node.js
  - https://nodejs.org
- PM2
  - https://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/
- NGINX
  - https://docs.nginx.com/?_ga=2.4975283.1227355032.1619285306-501395053.1619109877
- Amazon Linux Extras
  - https://aws.amazon.com/amazon-linux-2/faqs/