AWS Mini Projects Portfolio

Name: Aanand M

AWS Mini Projects Portfolio

Created by: Aanand M

Date: April 2025

## Project 1: Hosting a Static Website on EC2 (Ubuntu)

Objective: Host a static HTML website using an EC2 instance running Ubuntu.

### Steps Taken:

- Launched an EC2 instance using Ubuntu as the OS.

- Connected to the EC2 instance using SSH.

- Installed Apache server using 'sudo apt-get install apache2'.

- Uploaded HTML files to /var/www/html/ directory.

- Configured the security group to allow HTTP (port 80) traffic.

### Why I Did It:

To understand EC2 setup, web server configuration, and security group management.

### Challenges Faced:

- Initially, server wasn't reachable due to closed port. Resolved by opening port 80.

- Had to manually start Apache server using 'sudo service apache2 start'.

### What I Learned:

- Basics of EC2, security group configuration, and networking.

- How to configure Apache on Ubuntu and manage a static website.

- Importance of security groups in cloud infrastructure.

## Project 2: IAM User Creation and Security Groups

Objective: Create IAM users and configure security groups to secure EC2 instances.

### Steps Taken:

- Created multiple IAM users with different roles and policies.

- Configured Security Groups to control access to EC2 instances.

- Tested access using IAM users.

### Why I Did It:

To learn how IAM helps manage access to AWS resources and to understand the role of security groups in securing resources.

### Challenges Faced:

- Confusion around user roles and permissions.

- Had to recheck policies multiple times for correct access rights.

### What I Learned:

- IAM policies can be very granular to control user access.

- Security groups are like firewalls for your EC2 instances.

- IAM roles are crucial for defining what users can and cannot do.

## Project 3: Creating Custom AMIs and Templates

Objective: Create custom Amazon Machine Images (AMIs) and launch templates for easier management.

### Steps Taken:

- Created an EC2 instance with specific software pre-installed.

- Created an AMI of the configured instance.

- Used AMI to launch multiple instances.

- Created a launch template for standardized EC2 setup.

### Why I Did It:

To understand how AMIs can be used for quick replication and scalability.

### Challenges Faced:

- Had to carefully configure AMI to ensure all required software was included.

- Found out AMIs have storage limitations.

### What I Learned:

- AMIs are useful for replication and quick scaling.

- Launch templates allow easier and more consistent EC2 deployments.

## Project 4: Working with S3 (Upload, Versioning, Static Site Hosting)

Objective: Upload objects to S3, enable versioning, and host a static website.

### Steps Taken:

- Created an S3 bucket and uploaded static files.

- Enabled versioning on the bucket for file management.

- Configured the bucket to host a static website.

### Why I Did It:

To get hands-on experience with S3's storage capabilities, versioning, and website hosting.

### Challenges Faced:

- Encountered permission issues when trying to access files.

- Had to set bucket policy to allow public read access.

### What I Learned:

- S3's versioning feature is very useful for managing file changes.

- Static site hosting on S3 is simple and cost-effective.

## Project 5: Lambda for Automated Notifications with SNS

Objective: Trigger an SNS email notification whenever a new object is uploaded to S3.

### Steps Taken:

- Created an S3 bucket and an SNS topic.

- Created a Lambda function that gets triggered by S3 upload events.

- Wrote Python code in Lambda to send a message to the SNS topic.

- Subscribed an email to the SNS topic to receive notifications.

### Why I Did It:

To learn about serverless computing and event-driven architecture in AWS.

### Challenges Faced:

- Lambda permissions issues initially prevented SNS publishing.

- Had to manually confirm SNS subscription via email.

### What I Learned:

- Lambda functions are powerful for automation and event handling.

- SNS is an effective tool for sending notifications across different platforms.

## Project 6: Load Balancer and Auto Scaling

Objective: Set up a load balancer and configure auto scaling for an EC2 fleet.

### Steps Taken:

- Created an EC2 instance and launched it in an Auto Scaling group.

- Set up an Elastic Load Balancer to distribute traffic across instances.

- Configured Auto Scaling policies to add instances during traffic spikes.

### Why I Did It:

To understand how AWS scales applications and maintains high availability.

### Challenges Faced:

- Configuring Auto Scaling policies for optimal instance scaling.

- Testing load balancer with traffic simulation tools.

### What I Learned:

- Auto Scaling allows EC2 instances to be added/removed automatically based on demand.

- Elastic Load Balancer ensures high availability by distributing traffic across instances.

## Project 7: Elastic Beanstalk for Hosting a Static JS Website

Objective: Deploy a static JS website using Elastic Beanstalk.

### Steps Taken:

- Created an Elastic Beanstalk environment for static website deployment.

- Deployed a static JS website with simple HTML and JS files.

- Monitored the app's health and performance through Elastic Beanstalk's dashboard.

### Why I Did It:

To simplify deployment using Elastic Beanstalk and learn about managed services in AWS.

### Challenges Faced:

- Initially faced issues with environment configurations, which required adjustments.

- Learning how to configure environment variables for JS apps.

### What I Learned:

- Elastic Beanstalk automates app deployment and environment management.

- It's great for quick prototypes and small applications.

## Project 8: Creating VPC with Public and Private Subnets

Objective: Create a VPC with a public subnet connected to the internet and a private subnet for secure instances.

### Steps Taken:

- Created a VPC with custom CIDR block.

- Created both public and private subnets.

- Set up routing tables for internet access for public subnet.

- Launched EC2 instances in both public and private subnets.

### Why I Did It:

To understand VPC networking and how public/private subnets work in AWS.

### Challenges Faced:

- Setting up routing tables and ensuring proper access between subnets.

- Testing connectivity between public and private instances.

### What I Learned:

- VPC allows you to create isolated networks for different parts of your application.

- Understanding routing tables is key for managing traffic flow.