ADL Prac manual

Exp1 (Setup AWS Cloud9 IDE)

1. Login with your AWS account.
2. Navigate to Cloud 9 service from Developer tools section.
3. Click on Create Enviornment
4. Provide name for the Enviornment (WebAppIDE) and click on next.
5. Keep all the Default seetings
6. Review the Enviornment name and Settings and click on Create Enviornment
7. Till that time open IAM Identity and Access Management in order to Add user In other tab.
8. Add user provide manual password if you want and click on Next permission tab.
9. Provide group name and click on create group.
10. Review for user settings and click on create user.
11. Navigate to user Groups from left pane in IAM.
12. click on your group name which you have created and navigate to permission tab
13. Now click on Add permission and select Attach Policy after that search for Cloud9 related policy and select Awscloud9EnviornmentMember policy and add it.
14. now we move towards our cloud9 IDE Enviornment tab it shows as shown
15. Enter commands on terminal:
    a. git –version
    b. aws iam get-user
16. create new file or choose from template, here m opting html file to collaborate.
17. Edit html file and save it
18. now in order to share this file to collaborate with other members of your team click on
19. Share option on Right Pane and username which you created in IAM before into Invite members and enable permission as RW (Read and Write) and click on Done..
20. Now Open your Browsers Incognito Window and login with IAM user which you configured before.
21. After Successful login with IAM user open Cloud9 service from dashboard services and click on shared with you enviornment to collaborate.
22. Click on Open IDE you will same interface as your other member
23. Use chat option to chat with members or edit the html file shared.

Exp2 (To Build Your Application using AWS CodeBuild and Deploy on a S3 bucket)

1. Login to AWS and go to AWS Elastic Beanstalk and click on create.
2. Name your web app and choose PHP from the drop-down menu(or any other language you are interested in) and then click Create Application.
3. Click on create Application,Note: This will take several minutes to complete.
4. To use Amazon S3 as your source, you will retrieve the sample code from the AWS GitHub repository, save it to your computer, and upload it to an Amazon S3 bucket Visit our GitHub repository containing the sample code at https://github.com/imoisharma/aws-codepipeline-s3-codedeploy-linux-2.0
5. open the Amazon S3 console and create your Amazon S3 bucket:
6. • Click Create Bucket
   • Bucket Name: type a unique name for your bucket, such as awscodepipeline-demobucket- variables. All bucket names in Amazon S3 must be unique, so use one of your own, not one with the name shown in the example.
   • Region: In the drop-down, select the region where you will create your pipeline such as ap- South-1
   • Click Create.
7. Goto Pipeline again and create it
8. give zip file name in S3 object Key and choose bucket name which you created
9. Deploy Stage:
   • Deployment provider: Click AWS Elastic Beanstalk.
   • Application name: MYEBS.
   • Environment name: Click Myebs-env.
10. Now go to your EBS environment and click on the URL to view the sample website you deployed.
    You have successfully created an automated software release pipeline using AWS CodePipeline!

Exp5 (To understand terraform lifecycle, core concepts/terminologies and install it on a Linux Machine.)

.

1. To download go to site https://www.terraform.io/downloads.html Select the appropriate package for your operating system and architecture.
2. unzip the archive by using below command:
   - Unzip terraform_1.0.3_linux_amd64.zip
   - cd terraform_1.0.3_linux_amd64/
   - sudo mv terraform /usr/local/bin
   - terraform -v

Exp 6 (To Build, change, and destroy AWS infrastructure Using Terraform)

1. Terminal Commands
    - sudo apt-get install curl
    - curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
    - sudo apt install unzip
    - sudo unzip awscliv2.zip
    - sudo ./aws/install
    - aws –version
2. Create a new access key if you don't have one.Login to AWS console, click on username and go to My security credentials.
   Continue on security credentials, click on access keys
3. Perform below commands in Linux where you have installed Terraform:
    - aws configure
    - cd ~
    - mkdir project-terraform
    - cd project-terraform
    - sudo nano variables.tf
4. Create Key Pair , Give name to key pair file as terraform
5. Use your Region and Key name in variable.tf as shown and provide instance type which you want to create in variables.tf

   Variables.tf:

```
variable "aws_region" {
description = "The AWS region to create things in."
default = "ap-south-1"
}
variable "key_name" {
description = "SSH keys to connect to ec2 instance"
default = "terraform"
}
variable "instance_type" {
description = "instance type for ec2"
default = "t2.micro"
}
```

6. Create main.tf file by **sudo nano main.tf**

```
provider "aws" {
region = var.aws_region
}

#Create security group with firewall rules
resource "aws_security_group" "security_jenkins_port" {
name = "security_jenkins_port"
```

```
description = "security group for jenkins"

ingress {
from_port = 8080
to_port = 8080
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}

ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
# outbound from jenkis server
```



```
egress {
from_port = 0
to_port = 65535
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}

tags= {
Name = "security_jenkins_port"
}
}

resource "aws_instance" "myFirstInstance" {
ami = "ami-0b9064170e32bde34"
key_name = var.key_name
instance_type = var.instance_type
security_groups= [ "security_jenkins_port"]
tags= {
Name = "jenkins_instance"
}
}
```

```
# Create Elastic IP address
resource "aws_eip" "myFirstInstance" {
vpc = true
instance = aws_instance.myFirstInstance.id

Department of Information Technology | APSIT

tags= {
Name = "jenkins_elstic_ip"
}
}
```

7. **Terminal:**
    - terraform init
    - terraform plan
    - terraform apply (Provide the value as Yes for applying terraform)
8. Now login to EC2 console, to see the new instances up and running, you can see Jenkins_instance is up and running which we deploy from terraform.
9. **terraform destroy (**in terminal, to destroy terraform**)**

Exp7 (To understand Static Analysis SAST process and learn to integrate Jenkins SAST to SonarQube/GitLab) .

Here are the sequential steps for installing and configuring a Jenkins and SonarQube CI/CD environment using Docker containers, along with configuring Jenkins with the SonarQube Scanner plugin:

Steps to Install and Configure Jenkins and SonarQube

1. Install Jenkins
   - Add the Jenkins GPG key:

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

- Append the Jenkins Debian package repository to sources.list.
- Update the package list:

sudo apt update

- Install Jenkins:

sudo apt install jenkins

- Start Jenkins:

  sudo systemctl start jenkins

- Verify Jenkins is running:

  sudo systemctl status jenkins

- Allow traffic on port 8080:

  sudo ufw allow 8080

2. Set Up Jenkins
   - Open Jenkins in a web browser: `http://your_server_ip_or_domain:8080`
   - Retrieve the initial password:

     sudo cat /var/lib/jenkins/secrets/initialAdminPassword

   - Paste the password to unlock Jenkins.
   - Install suggested plugins.
   - Set up the first administrative user and click Save and Finish.

3. Install and Configure SonarQube
   - Run SonarQube using Docker:

     sudo docker run -d -p 9000:9000 sonarqube

   - Access SonarQube in a web browser: `http://localhost:9000`
   - Log in using default credentials (admin:admin).

4. Generate SonarQube User Token
   - Navigate to Administration > User > My Account > Security.
   - Generate a new token and copy it.

5. Configure Jenkins with SonarQube Scanner Plugin
   - Install the SonarQube Scanner plugin:
     - Go to Manage Jenkins > Plugin Manager > Available.
     - Search for "SonarQube Scanner" and install it.
   - Configure SonarQube server:
     - Go to Manage Jenkins > Configure System > SonarQube Server.

- Add SonarQube with the Installation Name and Server URL.
  - Add the Authentication token in Jenkins Credential Manager.
- Set up SonarQube Scanner:
  - Go to Manage Jenkins > Global Tool Configuration > SonarQube Scanner.
  - Click "Add SonarQube Scanner" and choose the installer (e.g., from Maven Central).

6. Integrate SonarQube Scanner in Jenkins Pipeline
   - Modify the Jenkinsfile to include a new stage for SonarQube scanning.

7. Set Up GitHub Configuration in Jenkins
   - Configure Git cloning into Jenkins as needed.

8. Build the GitHub Repository in Jenkins
   - Execute the build and ensure successful integration with SonarQube.

Exp11 (To understand AWS Lambda, its workflow, various functions and create your first Lambda functions using Python / Java / Nodejs)

Here are the sequential steps for creating your first AWS Lambda function using Python:

Steps to Create Your First Lambda Function Using Python

1. Open AWS Console
   - Search for the Lambda service and open the home screen of Lambda.

2. **Choose Region**
   - Select the region in which you want to create the Lambda function (as it is region-specific).

3. Create Lambda Function
   - Name the function "sum."
   - Select the latest version of Python as the runtime.
   - Choose a role with basic Lambda permissions to allow CloudWatch for monitoring.

4. Verify Creation
   - Confirm that the Lambda "sum" function is created successfully.

5. Write Sample Python Code

- Implement the code for the sum of two numbers.

```
import json
def lambda_handler(event, context):
        first_number = 100
        second_number = 200
        sum = first_number + second_number
        return sum
```

6. Configure Test Event

- Create a test event in JSON format.

7. Write Second Sample Python Code

- Implement a second sample Python code that includes an if condition.

```
def lambda_handler (event, context):
    if event["name"] == "vishal":
    return "apsit"|
```

8. Configure Test Event for Second Code

- Set up a test event for the second sample code, ensuring it returns a value as "apsit" if conditions are met.

```
{
 “name”: "omkar"
}
```

Exp12 (To create a Lambda function which will log “An Image has been added” once you add an object to a specific bucket in S3)

Steps to Use AWS Lambda with Amazon S3

1. **Create S3 Bucket**
    - Go to AWS Services and select S3 in the storage section.
    - Click "Create bucket" to store uploaded files.
    - Enter the Bucket name and select the Region.
    - Click the "Create" button.
    - Click the bucket name to upload files.
2. **Create Role that Works with S3 and Lambda**
    - Go to AWS Services and select IAM.
    - Click on "Roles" and then "Create role."
    - Choose the service that will use this role (select Lambda) and click "Next: Permissions."
    - Add permissions: AmazonS3FullAccess, AWSLambdaFullAccess, CloudWatchFullAccess, and click "Next: Tags."
    - Enter the Role name and description, then click "Create Role."
3. **Create Lambda Function and Add S3 Trigger**
    - Go to AWS Services and select Lambda.
    - Click "Create function," enter a name, choose the Runtime, and select the Role.
    - Click "Create function."
    - Choose "Add trigger" and select S3.
    - Enter details, such as Prefix and File pattern (e.g., for .jpg files), then click "Add."
    - Use the online editor to add code to handle S3 events.
    - Save changes.
4. **Test the Lambda Function**
    - Open the S3 bucket created earlier.
    - Upload a file (e.g., an image) to the bucket.
    - To check the trigger details, go to AWS Services and select CloudWatch.
    - Open the logs for the Lambda function to see the output indicating the file was uploaded.