

LOW LEVEL DESIGN - DOCUMENT

OVERVIEW — This Low-Level Design Document delineates the detailed architecture and configuration of a robust web application deployed on Amazon Web Services (AWS). The design focuses on achieving [high availability](#), [security](#), and efficient continuous integration and deployment (CI/CD) using [AWS CodePipeline](#). The system leverages multiple AWS services to create a scalable and fault-tolerant environment.

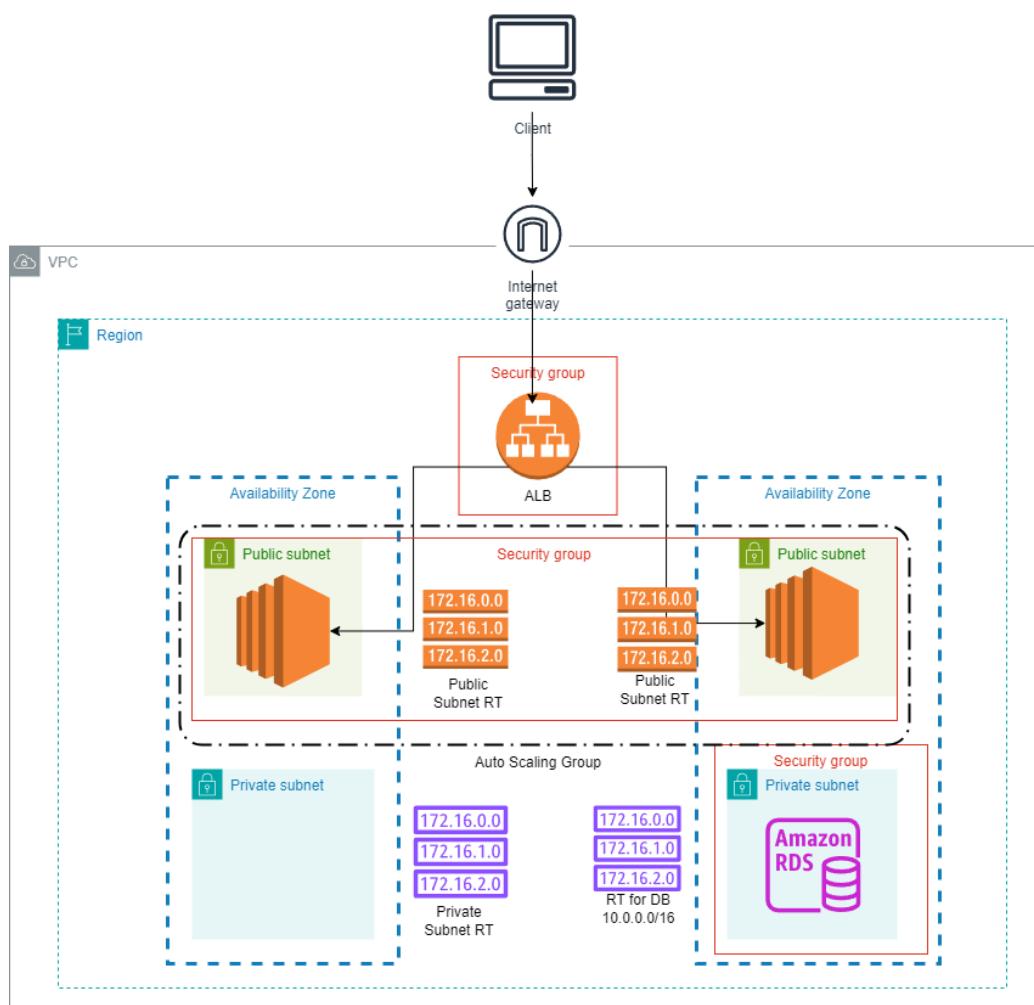
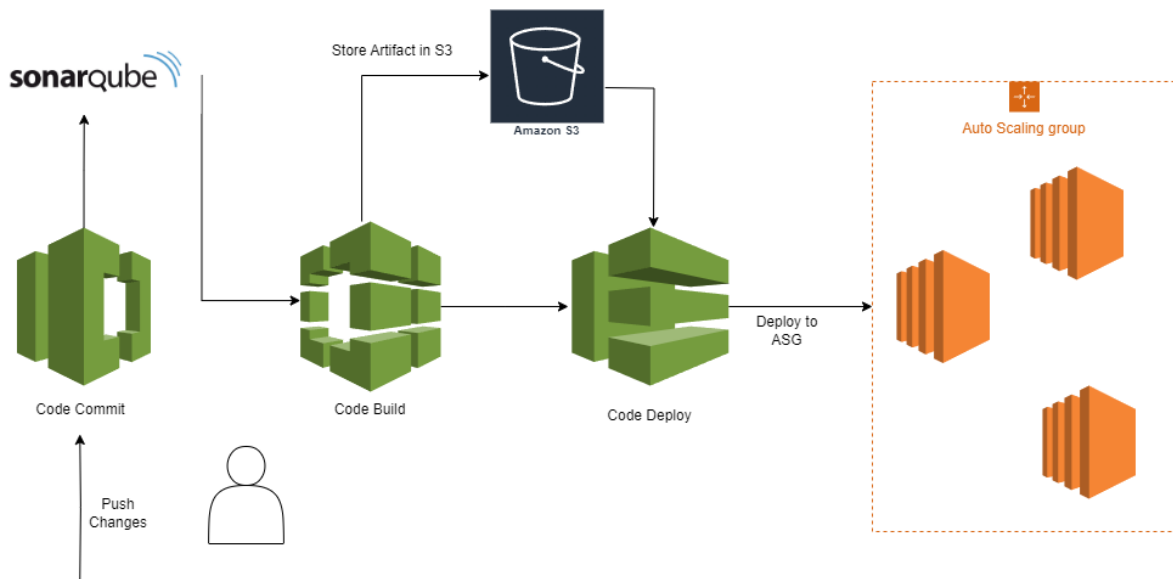
ARCHITECTURE COMPONENTS —

1. Virtual Private Cloud (VPC) — The architecture is encapsulated within a Virtual Private Cloud (VPC) to establish a private network within the AWS Cloud. Spanning across two availability zones, [ap-south-1a and ap-south-1b](#), this setup ensures redundancy and high availability for the web application.
2. Subnet Configuration — Within each availability zone, a meticulously crafted configuration comprises one public subnet associated with a dedicated route table for outbound internet traffic and one private subnet associated with another route table. This subnet architecture ensures the segregation of public-facing resources and those requiring heightened security within the VPC.
3. Database Tier — A MySQL Relational Database Service (RDS) instance is housed in the private subnet. [Configured to only allow traffic within the VPC](#), the RDS instance ensures data privacy and security by restricting access solely to resources within the same virtual network.
4. Autoscaling Group — To bolster high availability, an Auto Scaling Group (ASG) is deployed in the public subnets. This ASG dynamically scales the number of Amazon Elastic Compute Cloud (EC2) instances based on traffic load, ensuring resilience and consistent application performance.

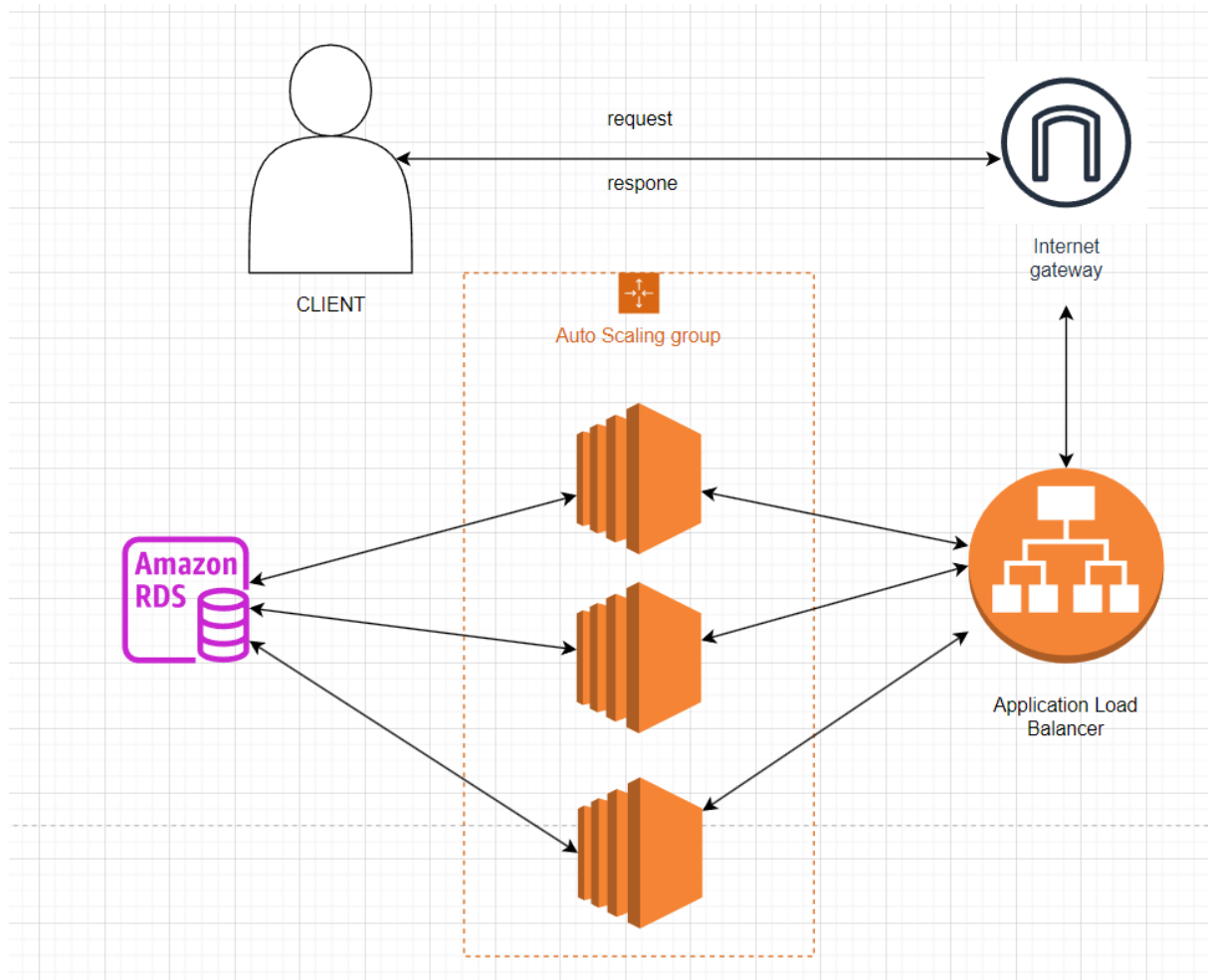
5. Load Balancing — An Application Load Balancer (ALB) stands at the forefront, listening on [port 80](#) to receive incoming user traffic. The ALB seamlessly redirects this traffic to [port 8080](#) on the EC2 instances within the ASG, where the web application runs, distributing the load evenly for optimal performance.
6. User Access and Gateway — User access into the VPC is channeled through an Internet Gateway. This gateway serves as a secure entry point, directing incoming traffic to the ALB, where it is further routed to the appropriate resources within the VPC.
7. CI/CD Pipeline — For efficient software development and deployment, AWS CodePipeline is employed. The CI/CD process initiates with developers pushing changes to a [CodeCommit](#) repository. Subsequently, a [Code Analysis](#) and [CodeBuild stage](#) utilizes [Maven to build a Java WAR package](#). In the final [CodeDeploy stage](#), the artifact is deployed onto the EC2 instances within the Auto Scaling Group.
8. Infrastructure Provisioning with Terraform — The infrastructure provisioning process involves Terraform, orchestrating the creation of resources and [pre configured ec2 instances with code deploy agent and docker installed](#). The [Terraform script](#) installs the [CodeDeploy agent on EC2 instances](#) and establishes the necessary connections, including RDS connection configurations, to facilitate a seamless deployment process.
9. Docker Container — Docker Container in which [web application is containerize](#) is deployed on the ec2 instances.

ARCHITECTURE DIAGRAMS OF VPC AND CODE-PIPELINE.

1. Architecture of Code-Pipeline and VPC



2. Data / Network Flow Diagram



File Directory Structure —

```
CODE-DEPLOYMENT-FURTHER
├── ap.tmpl
├── c1-versions.tf
├── c2-generic-variables.tf
├── c3-local-values.tf
├── c4-01-vpc-variable.tf
├── c4-02-vpc-module.tf
├── c4-03-vpc-output.tf
├── c5-02-securitygroup-outputs.tf
├── c5-03-securitygroup-pubInstance.tf
├── c5-05-securitygroup-lb.tf
├── c5-06-securitygroup-db.tf
├── c6-01-datasource-ami.tf
├── c7-01-ec2instance-variables.tf
├── c8-04-ALB-main.tf
├── c9-02-asg-main.tf
├── c9-05-asg-launchconf.tf
├── c10-01-rds-main.tf
├── c10-02-rds-output.tf
├── env_variables.sh
├── IAMrole.tf
├── InstanceForAMI.pem
├── key
├── null-resource.tf
├── terraform.tfstate
├── terraform.tfstate.backup
└── terraform.tfvars
```

Note — The explanation of the code is going to be into sections.

Section 1: Providers and VPC —

1. c1-versions.tf

```
terraform {
  required_version = ">= 0.13"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 5.6" # which means any version equal & above
    }
    null = {
      source = "hashicorp/null"
      version = "~> 3.0"
    }
  }
}

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

- The Amazon Web Services provider is used to interact with the many resources supported by AWS.
- During terraform init, terraform first downloads all the provider modules that are going to be used in the code.

2. c2-generic-variables.tf

```
# AWS Region
variable "aws_region" {
  description = "Region in which AWS Resource to be created"
  type = string
  default = "ap-south-1"
}

# Environment Variable
variable "environment" {
  description = "Environment Variable used as a prefix"
  type = string
  default = "dev"
}

# Business Divison
variable "intern" {
  description = "Who will use this infra"
  type = string
  default = "intern"
}
```

- In generic variables file all the most used variables are declared.

3. c3-local-values.tf

```
# Define Local values in terraform
locals {
  owners = var.intern
  environment = var.environment
  name = "${var.intern}-${var.environment}"
  common_tags = {
    owners = local.owners
    environment = local.environment
  }
}
```

- Local block define set of tags and variables that can be used together.

4. c4-01-vpc-variable.tf

```
# CIDR Block
variable "cidr_block" {
  description = "cidr block range which constitute the vpc"
  type = string
  default = "10.0.0.0/16"
}

# vpc name
variable "vpc_name" {
  description = "VPC name"
  type = string
  default = "myvpc"
}

variable "availability_zones" {
  description = "AZs where infra will provision"
  type = list(string)
  default = ["ap-south-1a", "ap-south-1b"]
}

variable "public_subnet_cidrs" {
  description = "No. of public subnet with their cidr values"
  type = list(string)
  default = ["10.0.101.0/24", "10.0.102.0/24"]
}

variable "db_subnet_cidrs" {
  description = "No. of db subnet with their cidr values"
  type = list(string)
  default = ["10.0.1.0/24", "10.0.2.0/24"]
}

variable "enable_database_subnet_group" {
  description = "Allows creation of private subnet for database."
  type = bool
  default = true
}

variable "enable_database_subnet_RT" {
  description = "Allows creation of private subnet RT for database."
  type = bool
  default = true
}
```

- The vpc related variables are declared in this file. Importance are cidr block range, subnets range, AZs etc.

5. c4-02-vpc-module.tf

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "5.4.0"

  # VPC basic details
  name = "${local.name}-${var.vpc_name}"
  cidr = var.cidr_block
  azs   = var.availability_zones
  public_subnets = var.public_subnet_cidrs

  # database subnets
  create_database_subnet_group      = var.enable_database_subnet_group
  create_database_subnet_route_table = var.enable_database_subnet_RT
  #create_database_nat_gateway_route = true
  #create_database_internet_gateway_route = true
  database_subnets = var.db_subnet_cidrs

  # Vpc DNS parameters.
  enable_dns_hostnames = true
  enable_dns_support   = true

  # add tags for subnet
  public_subnet_tags = {
    Type = "public-subnets"
  }
  database_subnet_tags = {
    Type = "database-subnets"
  }
  tags = local.common_tags
  vpc_tags = local.common_tags
}
```

- Configurations of vpc is written in this file. Vpc module by hashicorp is used to save time and increase readability.

6. c4-03-vpc-output.tf

```
# VPC ID
output "vpc_id" {
  description = "The ID of the VPC"
  value       = module.vpc.vpc_id
}

# VPC CIDR blocks
output "vpc_cidr_block" {
  description = "The CIDR block of the VPC"
  value       = module.vpc.vpc_cidr_block
}

# VPC Public Subnets
output "public_subnets" {
  description = "List of IDs of public subnets"
  value       = module.vpc.public_subnets
}

# VPC AZs
output "azs" {
  description = "A list of availability zones specified as argument to this module"
  value       = module.vpc.azs
}
```

- All the outputs related to vpc is declared in this file.

Section 2: Security Groups and Data Source —

- This section will be related to coding all the required security groups and data source to fetch the latest ubuntu ami.
1. c5-03-securitygroup-pubInstance.tf —

```
module "public_sg" {  
  source = "terraform-aws-modules/security-group/aws"  
  version = "5.1.0"  
  name     = "ec2_eg2"  
  description = "Security group for public subnet bastion host ssh port"  
  vpc_id    = module.vpc.vpc_id  
  
  tags = local.common_tags  
  # Ingress Rules and Cidr Blocks  
  ingress_rules = ["ssh-tcp", "http-80-tcp", "http-8080-tcp"]  
  ingress_cidr_blocks = ["0.0.0.0/0"]  
  # Egress Rule open all  
  egress_rules = ["all-all"]  
}
```

- Here we are defining the security group for ec2 instance that will be created by auto scaling group and lives in public subnet.
- Here we are allowing traffic at port 22, 80 and 8080 of ec2 anywhere from internet.

2. c5-05-securitygroup-lb.tf —

```
module "loadbalancer_sg" {  
  source = "terraform-aws-modules/security-group/aws"  
  version = "5.1.0"  
  name     = "public-instance-sg"  
  description = "Security group for load balancer to allow traffic at port 80"  
  vpc_id    = module.vpc.vpc_id  
  
  tags = local.common_tags  
  # Ingress Rules and Cidr Blocks  
  ingress_rules = ["http-80-tcp"]  
  ingress_cidr_blocks = ["0.0.0.0/0"]  
  # Egress Rule open all  
  egress_rules = ["all-all"]  
}
```

- Here we are creating a security group for application load balancer.

- The application load balancer will listen at port 80 and redirects to ec2 instances at port 8080.

3. c5-06-securitygroup-db.tf

```
module "db-sg" {
  source      = "terraform-aws-modules/security-group/aws"
  version     = "5.1.0"
  name        = "public-instance-sg"
  description = "Security group for db to allow traffic at port 3306"
  vpc_id      = module.vpc.vpc_id

  tags = local.common_tags
  # Ingress Rules and Cidr Blocks
  #ingress_rules = ["http-80-tcp"]
  ingress_with_cidr_blocks = [
    {
      from_port = 3306
      to_port   = 3306
      protocol  = "tcp"
      cidr_blocks = "0.0.0.0/0"
    }
  ]
  ingress_cidr_blocks = ["0.0.0.0/0"]
  # Egress Rule open all
  egress_rules = ["all-all"]
}
```

- Here we are defining security group for RDS to allow traffic inside RDS from port 3306.

4. c5-02-securitygroup-outputs.tf

```
output "public_sg_id" {
  description = "The ID of the security group"
  value      = module.public_sg.security_group_id
}

output "public_sg_vpc_id" {
  description = "The VPC ID"
  value      = module.public_sg.security_group_vpc_id
}

output "public_sg_name" {
  description = "The name of the security group"
  value      = module.public_sg.security_group_name
}

output "public_sg_description" {
  description = "The description of the security group"
  value      = module.public_sg.security_group_description
}

#-----

output "loadbalancer_sg_id" {
  description = "The ID of the security group"
  value      = module.loadbalancer_sg.security_group_id
}

output "loadbalancer_sg_vpc_id" {
  description = "The VPC ID"
  value      = module.loadbalancer_sg.security_group_vpc_id
}

output "loadbalancer_sg_name" {
  description = "The name of the security group"
  value      = module.loadbalancer_sg.security_group_name
}
```

All the security groups related outputs are declared here.

5. c6-01-datasource-ami.tf

```
data "aws_ami" "amzlinux2" {
  most_recent = true
  owners     = ["amazon"]

  filter {
    name = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20231207"]
  }

  filter {
    name = "root-device-type"
    values = ["ebs"]
  }

  filter {
    name = "virtualization-type"
    values = ["hvm"]
  }

  filter {
    name = "architecture"
    values = ["x86_64"]
  }
}
```

- Here we are defining data source to fetch AMI id of latest Ubuntu. We are going to use this AMI to provision instances.

Section 3: ALB, ASG ,RDS and Roles

1. c8-04-ALB-main.tf

```
resource "aws_lb" "terramino" {
  name           = "learn-asg-terramino-lb"
  internal       = false
  load_balancer_type = "application"
  security_groups = [module.loadbalancer_sg.security_group_id]
  subnets       = module.vpc.public_subnets
}
```

- Here we are defining application load balancer using resource block.
- `Internal = false`, ensures that ALB is internet facing.
- Type of ALB is `application load balancer`

```
resource "aws_lb_listener" "terramino" {
  load_balancer_arn = aws_lb.terramino.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.terramino.arn
  }
}
```

- Here we are defining listeners for load balancer, it will keep listening at port 80 of load balancer.
- Type of action is forward : it will forward the request to alb target group that we are about to define.

```

resource "aws_lb_target_group" "terramino" {
  name      = "learn-asg-terramino"
  port      = 8080
  protocol  = "HTTP"
  vpc_id    = module.vpc.vpc_id
  health_check {
    enabled            = true
    interval           = 30
    path               = "/LoginWebApp"
    port               = "traffic-port"
    healthy_threshold  = 3
    unhealthy_threshold = 3
    timeout            = 6
    protocol           = "HTTP"
    matcher            = "200-399"
  }
}

```

- Here we are defining target group that will redirect the request to instances at port 8080.
- The [web application is running at path '/LoginWebApp'](#), so target group will check health at this path.
- If the instance will not respond at this path, the response comes in between 200- 399, then it will consider as unhealthy, and ASG will deregister it and launches a new instance.

```

resource "aws_autoscaling_attachment" "terramino" {
  autoscaling_group_name = aws_autoscaling_group.terramino.id
  lb_target_group_arn    = aws_lb_target_group.terramino.arn
}

```

- Here we are attaching the target group to AGS.
- That means target group will contain all the instances that are launched by ASG.

2. C9-02-asg-main.tf

```
resource "aws_autoscaling_group" "terramino" {
  min_size           = 1
  max_size           = 3
  desired_capacity    = 2
  launch_configuration = aws_launch_configuration.terramino.name
  # launch_template {
  #   id      = aws_launch_template.my_launch_template.id
  #   version = "$Latest"
  # }
  vpc_zone_identifier = module.vpc.public_subnets
}
```

- Here we are defining a ASG block named terramino, with desired capacity 2
- It will use launch configuration (that we are about to declare) to launch the ec2.

3. C9-05-asg-launchconf.tf

```
resource "aws_launch_configuration" "terramino" {
  name_prefix     = "learn-terraform-aws-asg-"
  image_id        = data.aws_ami.amzlinux2.id
  instance_type   = "t2.micro"
  security_groups = [module.public_sg.security_group_id]
  associate_public_ip_address = true
  key_name        = var.instance_keypair
  #user_data = filebase64("${path.module}/app2-simpleapp.sh")
  user_data = templatefile("${path.module}/ap.tpl", { HOST = aws_db_instance.default.address })
  iam_instance_profile = aws_iam_instance_profile.ec2_profile.name
  lifecycle {
    create_before_destroy = true
  }
  depends_on = [ aws_db_instance.default, null_resource.generate_env_file ]
}

resource "aws_iam_instance_profile" "ec2_profile" {
  name = "ec2_profile"
  role = aws_iam_role.EC2-S3Role.name
}
```

- This launch configuration will define the outline of ec2 that will be launched by ASG.

Some important points to highlight.

- Image_id is referring to the data source that we already defined.
- Templatefile function is used to give script with HOST variable.
- In the script we need a HOST value, that is why template file function is used.

- depends_on , this launch configuration will be created after the creation of rds end point. Because we need rds endpoint in the script.
- IAM role is created and assign to launch configuration to allow read and write permissions to S3 bucket
- This Role will be used in Code Deploy stage, when instance try to fetch artifact from S3.

4. C10-01-rds-main.tf

```
resource "aws_db_instance" "default" {
  allocated_storage    = 10
  db_name              = "mydb"
  engine               = "mysql"
  engine_version       = "5.7"
  instance_class       = "db.t3.micro"
  username             = "root"
  password             = "root12345"
  parameter_group_name = "default.mysql5.7"
  skip_final_snapshot = true
  vpc_security_group_ids = [module.db-sg.security_group_id]
  db_subnet_group_name = module.vpc.database_subnet_group
}
```

- Here we are defining rds resource block, it will create a [mysql 5.7](#) rds with [db.t3.micro](#) computing power.

5.IAMrole.tf

```
# Create an IAM Role for S3 Access.

resource "aws_iam_role" "EC2-S3Role" {
  name = "EC2-S3Role"
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      }
    ]
  })
}
```

- We are creating this role to for ec2 instance to allow access to artifacts stored in the s3 bucket.

```
# Create an S3 access policy to the above role.

resource "aws_iam_policy" "WebAppS3" {
  name       = "WebAppS3"
  description = "Policy for accessing S3 bucket"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = [
          "s3:GetObject",
          "s3:PutObject",
          "s3:DeleteObject"
        ]
        Effect = "Allow"
        Resource = ["*"]
      }
    ]
  })
}
```

```
# Attach the policy to the created role.

resource "aws_iam_role_policy_attachment" "s3_access_role_attachment" {
  policy_arn = aws_iam_policy.WebAppS3.arn
  role       = aws_iam_role.EC2-S3Role.name
}
```

- Here we are attaching policy with the role.

6. Null-resource.tf

```
output "rds_host" {
  value = aws_db_instance.default.address
}

# Generate the env_variables.sh file after obtaining RDS details
resource "null_resource" "generate_env_file" {
  provisioner "local-exec" {
    command = <<EOT
echo "export HOST=${aws_db_instance.default.address}" > env_variables.sh
EOT
  }
  # Run the provisioner when RDS details are available
  depends_on = [aws_db_instance.default]
}
```

- This is secondary thing to ensure that we will get the rds endpoint even if templatefile function fails to pass the rds end point into script.

Section 3: User data script (game changer) —

1. Code Deploy agent

```
#!/bin/bash
# This installs the CodeDeploy agent and its prerequisites on Ubuntu 22.04.
sudo apt-get update
sudo apt-get install ruby-full ruby-webrick wget -y
cd /tmp
wget https://aws-codedeploy-us-east-1.s3.us-east-1.amazonaws.com/releases/codedeploy-agent_1.3.2-1902_all.deb
mkdir codedeploy-agent_1.3.2-1902_ubuntu22
dpkg-deb -R codedeploy-agent_1.3.2-1902_all.deb codedeploy-agent_1.3.2-1902_ubuntu22
sed 's/Depends:.*Depends:ruby3.0/' -i ./codedeploy-agent_1.3.2-1902_ubuntu22/DEBIAN/control
dpkg-deb -b codedeploy-agent_1.3.2-1902_ubuntu22/
sudo dpkg -i codedeploy-agent_1.3.2-1902_ubuntu22.deb
systemctl list-units --type=service | grep codedeploy
sudo service codedeploy-agent status
```

- This part of the script will install code deploy agent into the instances that will help in code deploy stage.

2. Docker setup and installation.

```
# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# Add the Docker repository to Apt sources
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker packages
sudo apt-get update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io
```

- This is setting up docker repo and install the docker engine and docker compose.

3. My Sql RDS setup and database initialization.

```
sudo apt install -y mysql-server

# MySQL setup - Replace 'root' and 'root12345' with more secure credentials
sudo mysql -e "CREATE USER 'root'@'%' IDENTIFIED BY 'root12345';"
sudo mysql -e "GRANT ALL PRIVILEGES ON *.* TO 'root'@'%';"
sudo mysql -e "FLUSH PRIVILEGES;"

# Set MySQL credentials in a secure way
echo -e "[client]\nuser=root\npassword=root12345" > ~/.my.cnf
chmod 600 ~/.my.cnf

# Set the MySQL password for the current session
export MYSQL_PWD='root12345'

git clone https://github.com/Mohammedaslaan/LoginWebApp.git
cd LoginWebApp/

mysql -h ${HOST} -u root --ssl-mode=DISABLED < database.sql
```


- Here we are installing mysql and granting root permissions
- Setting up username and password as a environment variable to ensure no security warning will be there while executing the script in the instance.
- Using the passed variable HOST to get rds endpoint to configure the database in the rds.

AWS CICD-PIPELINE —

These are the some pre-steps that need to be done in order to configure code pipeline.

1. Create a Role with [code deploy full access](#) that will be used in code deploy stage.

Code Commit —

I have created a repo on code commit named “Java-webapp”, this will contain source code of application and some dependencies.


SonarQube Analysis —


1. Credentials of SonarCloud.io for project repo.
 2. Token = cea5b42661c436990326425d908c7544f169aebe
 3. Host.url = https://sonarcloud.io
 4. ProjectKey = recruitcrmintern_loginwebapp
 5. Organization = recruitcrmintern
- I have Integrated SonarQube analysis which is a code quality analysis, to do sonarQube analysis code build service of aws need buildspec.yml file.
 - Here is the [buildspec.yml file for sonarQube Analysis](#)

```
version: 0.2






phases:
  install:
    runtime-versions:
      java: corretto17
  pre_build:
    commands:
      - echo "Setting up environment variables"
      - export SONAR_TOKEN=cea5b42661c436990326425d908c7544f169aebe
  build:
    commands:
      - echo "Running SonarQube analysis"
      - mvn clean verify sonar:sonar -Dsonar.token=cea5b42661c436990326425d908c7544f169aebe \
        -Dsonar.host.url=https://sonarcloud.io -Dsonar.projectKey=recruitcrmintern_loginwebapp \
        -Dsonar.organization=recruitcrmintern
```

Result of SonarQube Analysis.

 [LoginWebApp Maven Webapp](#) NEW PUBLIC

 Passed

Last analysis: 1/2/2024, 1:43 AM • 137 Lines of Code • JSP, XML

 6	 0	 100%	 6	 0.0%
Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Duplications

Code Build

1. Code Build Create a separate environment in which it builds the application.
2. I have created a new project and define the configuration of that environment in which the source code will be build.
3. Select a Source Code Commit.
4. Environment

Environment

Environment image

☒ **Managed image**
Use an image managed by AWS CodeBuild

☐ **Custom image**
Specify a Docker image

Compute

☒ **EC2**
Optimized for flexibility during action runs

☐ **Lambda**
Optimized for speed and minimizes the start up time of workflow actions

Operating system

Ubuntu ▼

Runtime(s)

Standard ▼

Image

aws/codebuild/standard:6.0 ▼

Image version

Always use the latest image for this runtime version ▼

Privileged

☐ Enable this flag if you want to build Docker images or want your builds to get elevated privileges


5. Select a service Role that you have already created in pre steps.

Service role

☒ **New service role**
Create a service role in your account

☐ **Existing service role**
Choose an existing service role from your account

Role name

arn:aws:iam::634270476875:role/service-role/codebuild-new-build-mydiaries-service-

Type your service role name

6. Select a “Use a buildspec file” option.

- Here is the build spec file that i have created to use in this project.

```
! buildspec.yml > {} artifacts > [ ] files > [ ] 0
1  version: 0.1
2
3  phases:
4    build:
5      commands:
6        - echo Build completed on `date`
7        - mvn clean install
8
9  artifacts:
10   files:
11     - '**/*'
```

- Here we are only building the app using maven.

7. Use S3 in artifact option.

Artifact 1 - Primary

Type

Amazon S3

You might choose no artifacts if you are running tests or pushing a Docker image to Amazon ECR.

Bucket name

buildmydiaries9887

Name

The name of the folder or compressed file in the bucket that will contain your output artifacts. Use Artifacts packaging under Additional configuration to choose whether to use a folder or compressed file. If the name is not provided, defaults to project name.

javaapp.zip

☐ Enable semantic versioning

Use the artifact name specified in the buildspec file

8. Allow Service Role permission.

Service role permissions

- ☒ Allow AWS CodeBuild to modify this service role so it can be used with this build project
arn:aws:iam::634270476875:role/service-role/codebuild-new-build-mydiaries-service-role

9. Tick on Cloudwatch to see logs.

Code Deploy

1. Create new application. Select compute platform as ec2.
2. Create deployment group. Deployment group takes the ec2 instances and group them together to deploy artifacts in them at once.
3. Give the role that we have already created with code deploy full access.
4. There are two deployment types exists. I have chosen in-place.
5. Select the auto scaling group that is created by terraform in environment configuration option. Deployment group will deploy the artifacts on the all the ec2 that comes under auto scaling group.
6. Select deployment configuration settings as [OneAtTime](#). It will take one instance at a time to deploy artifact.
7. There is a Option of Load balancer. I have not chosen it, But if we choose this option, then a load balancer will first register the instance and block all the traffic, then deploy the artifact, then register it again.
8. Code Deploy uses a appspec.yml file in order to execute commands while deploying.

Our Appspec.yml file.

```
version: 0.0
os: linux
files:
  - source: /
    destination: /opt/myapp
    overwrite: true
fileExistsBehavior: OVERWRITE
hooks:

  AfterInstall:
    - location: scripts/build_docker_image.sh
    - location: scripts/start_docker_container.sh
      timeout: 300
    runas: root
```

1. First it will copy all the content into /opt/myapp directory of the instance. Overwrite: true, ensure the every time the whole files are copied.
2. We have two files that will do deployment.
Build_docker_images.sh and Start_docker_container.sh
3. Build_docker_images.sh

```
#!/bin/bash

# Container name
CONTAINER_NAME=myapp-container

# Check if the container is running
if docker container inspect "$CONTAINER_NAME" &> /dev/null; then
    echo "Stopping and removing existing $CONTAINER_NAME container..."
    docker stop "$CONTAINER_NAME"
    docker rm "$CONTAINER_NAME"
fi

# Check if the Docker image exists
if docker image inspect myapp-image:latest &> /dev/null; then
    echo "Removing existing myapp:latest Docker image..."
    docker rmi myapp-image:latest
fi

# # Change to the deployment directory and build the new Docker image
# cd /opt/myapp
# docker build -t myapp:latest .

# Ensure the directory exists
mkdir -p /opt/myapp

# Change into the directory
cd /opt/myapp || exit
```

```
# Change into the directory
cd /opt/myapp || exit
# Assuming Dockerfile is in the /opt/myapp directory
docker build -t myapp-image /opt/myapp
```

```
Dockerfile
1 FROM tomcat
2
3 ADD target/LoginWebApp.war /usr/local/tomcat/webapps/LoginWebApp.war
4
```

In build_docker_images.sh file

1. First it will check for existing docker container named "myapp-container".
2. If it is there then it will stop the container and remove it.
3. Then it will search for docker images "myapp-image:latest"
4. If it is there then it will remove the image.
5. These checks ensure the storage will not full, by keeping the necessary docker images only.
6. Then it will build the new docker image.

4. Start_docker_container.sh

```
#!/bin/bash

docker run -d -p 8080:8080 --name myapp-container myapp-image:latest
```

1. This script will simply start the container inside the ec2 instance and maps port 8080 of container to 8080 of ec2 instance.