

Deploying a Simple Node.js Application on AWS Elastic Beanstalk through AWS CodePipeline using Terraform (Task 11)

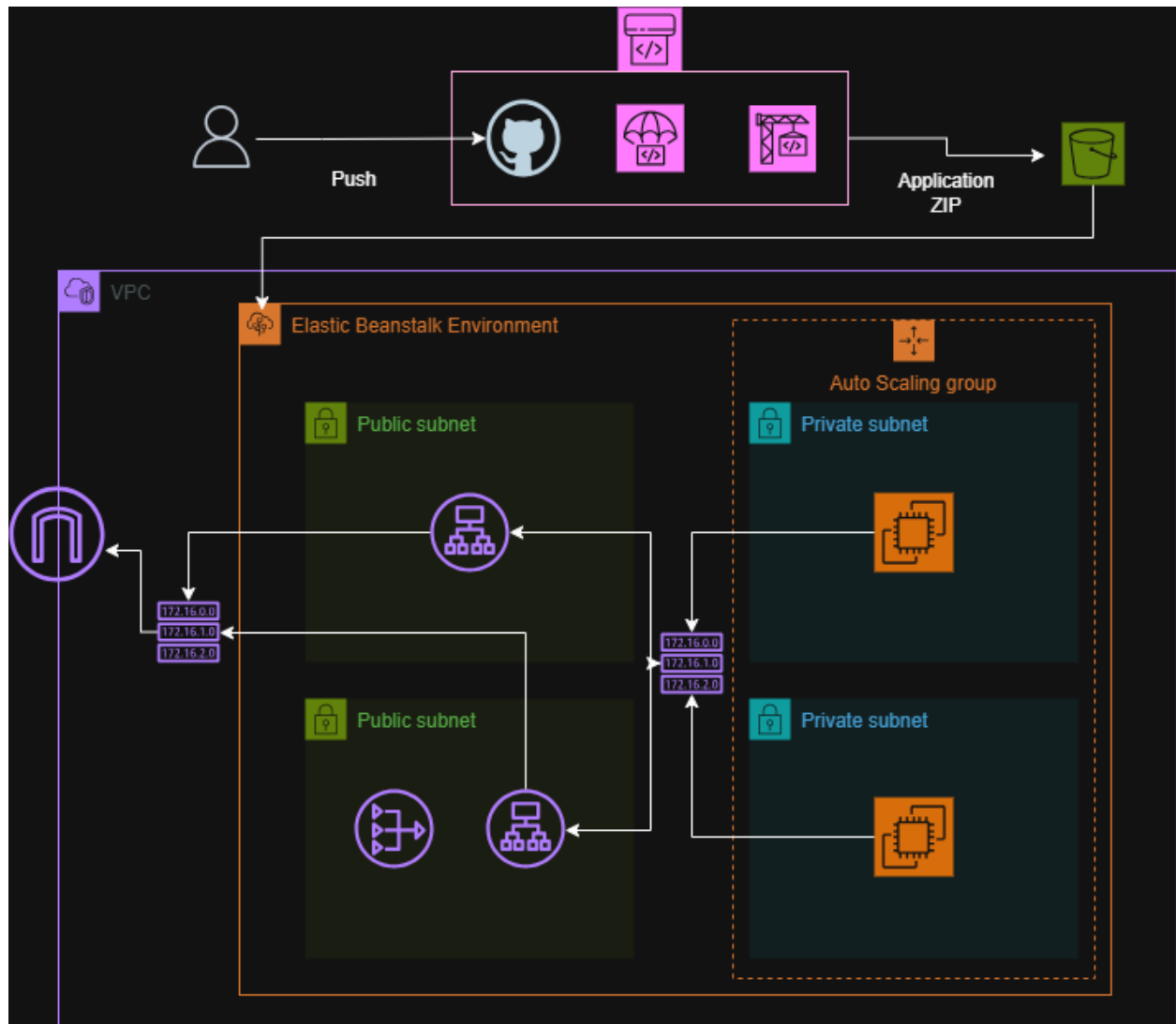


Zaeem Attique Ashar
Cloud Intern

Task Description:

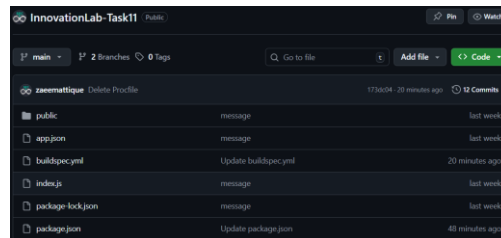
This project involves deploying a simple Node.js application using AWS Elastic Beanstalk. The application is built with the required dependencies, packaged into a ZIP file, and uploaded to an S3 bucket as the output artifact. Elastic Beanstalk automatically handles provisioning, deployment, and scaling. The environment is monitored through the dashboard, where logs, health, and configuration settings can be managed. The deployment is verified through the provided application URL, with updates deployed as needed.

Architecture Diagram:



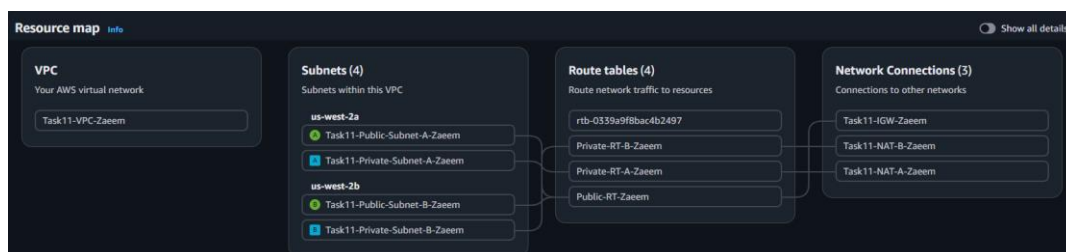
Task11.1: Prepare Node.js application code

- The files should be uploaded to a GitHub repository where they can be fetched by the AWS CodePipeline.
- The app.js file and the package.json file should be located at the root of the directory to be read by the AWS CodePipeline.



Task11.2: Create Networking infrastructure to be used by EB

- Create and configure a VPC
 - CIDR Block: 10.0.0.0/16
- Create and configure Subnets
 - Public Subnet A (us-west-2a), CIDR: 10.0.1.0/24
 - Private Subnet A (us-west-2a), CIDR: 10.0.2.0/24
 - Public Subnet B (us-west-2b), CIDR: 10.0.3.0/24
 - Private Subnet A (us-west-2a), CIDR: 10.0.4.0/24
- Create and configure NAT Gateways
 - NAT Gateway A in Public Subnet A
 - NAT Gateway B in Public Subnet B
- Create and configure Internet Gateway
 - Create and attach to the project's VPC
- Create and configure Route Tables
 - Public Route Table, Outbound rule: 0.0.0.0/0 -> IGW, attach to Public SN A&B
 - Private Route Table A, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN A
 - Private Route Table B, Outbound Rule: 0.0.0.0/0 -> NGW attach to Private SN B



Task 11.3: Set up an Elastic Beanstalk application and environment in AWS console

- Create an elastic beanstalk application:
 - Name: Task11-EB-App-Zaeem
- Create an elastic beanstalk environment:
 - Name: Task11-EB-App-Env-Zaeem
 - Application: Task11-EB-App-Zaeem
 - Tier: Webserver
 - Solution stack name: "64bit Amazon Linux 2023 v6.7.0 running Node.js 20"
 - Setting to set an EB env service role
 - Setting to set a VPC ID
 - Setting to set subnets inside VPC
 - Setting to set Load Balancer subnets
 - Setting to set an ALB Security Group ID
 - Setting to enable a listener
 - Setting to set Protocol HTTP
 - Setting to set EB Env Process port as 5000
 - Setting to set EB Env Process protocol HTTP
 - Setting to set a Health Check path "/"
 - Setting to set a Health Check interval of 30 sec
 - Setting to set a Health Check Timeout of 5 min
 - Setting to set ASG Launch Config to launch t3.micro
 - Setting to set Min Cluster Size of 2 instances
 - Setting to set Max Cluster Size of 4 instances
 - Setting to set an EC2 instance profile
 - Setting to set an EC2 Security Group ID
 - Setting to enable Rolling Updates based on %
 - Setting to set Stream Logs as true

```

5 resource "aws_elastic_beanstalk_environment" "Task11-EB-App-Env-Zaeem" {
6   name = "Task11-EB-App-Env-Zaeem"
7   application = aws_elastic_beanstalk_application.Task11-EB-App-Zaeem.name
8
9   tier = "WebServer"
10  solution_stack_name = "64bit Amazon Linux 2023 v6.7.0 running Node.js 20"
11
12  # Service Role for Elastic Beanstalk
13
14
15  setting {
16    namespace = "aws:elasticbeanstalk:environment"
17    name      = "ServiceRole"
18    value     = var.eb_service_role_arn
19  }
20
21  # VPC Configuration
22  setting {
23    namespace = "aws:ec2:vpc"
24    name      = "VPCId"
25    value     = var.vpc_id
26  }
27
28  setting {
29    namespace = "aws:ec2:vpc"
30    name      = "Subnets"
31    value     = join(",", var.private_sn_ids)
32  }
33
34  setting {
35    namespace = "aws:ec2:vpc"
36    name      = "ELBSubnets"
37    value     = join(",", var.public_sn_ids)
38  }
39
40  setting {
41    namespace = "aws:elasticbeanstalk:environment"
42    name      = "LoadBalancerType"
43    value     = "application"
44  }
45

```

```

46  setting {
47    namespace = "aws:elbv2:loadbalancer"
48    name      = "SecurityGroups"
49    value     = var.alb_sg_id
50  }
51
52  setting {
53    namespace = "aws:elbv2:listener:default"
54    name      = "ListenerEnabled"
55    value     = "true"
56  }
57
58  setting {
59    namespace = "aws:elbv2:listener:default"
60    name      = "Protocol"
61    value     = "HTTP"
62  }
63
64  setting {
65    namespace = "aws:elasticbeanstalk:environment:process:default"
66    name      = "Port"
67    value     = "5000"
68  }
69
70  setting {
71    namespace = "aws:elasticbeanstalk:environment:process:default"
72    name      = "Protocol"
73    value     = "HTTP"
74  }
75
76  setting {
77    namespace = "aws:elasticbeanstalk:environment:process:default"
78    name      = "HealthCheckPath"
79    value     = "/"
80  }
81
82  setting {
83    namespace = "aws:elasticbeanstalk:environment:process:default"
84    name      = "HealthCheckInterval"
85    value     = "30"
86  }

```


Task 11.4: Configure AWS CodePipeline with source stage pointing to the repository

- Create a CodeStar Connection with the GitHub repository for the pipeline
 - The connection has to be manually approved via console.

```
69 resource "aws_codestarconnections_connection" "Task11-Codestar-Connection-Zaeem" {
70     name           = "Task11-Codestar-Connection-Zaeem"
71     provider_type  = "GitHub"
72 }
```

- Create an AWS Code Pipeline as well as an artifact bucket

```
1 resource "aws_codepipeline" "Task11-CICD-Pipeline-Zaeem" {
2     name           = "Task11-CICD-Pipeline-Zaeem"
3     role_arn       = var.codepipeline_role_arn
4
5     artifact_store {
6         location = var.codepipeline_bucket
7         type     = "S3"
8     }
9 }
```

- Setup the source stage with the right codestar connection and the output artifact to be passed onto the next stage of the pipeline

```
11 stage {
12     name = "Source"
13
14     action {
15         name           = "Source"
16         category       = "Source"
17         owner          = "AWS"
18         provider       = "CodeStarSourceConnection"
19         version        = "1"
20         output_artifacts = ["source_output"]
21
22         configuration = {
23             ConnectionArn = aws_codestarconnections_connection.Task11-Codestar-Connection-Zaeem.arn
24             FullRepositoryId = "zaeemattique/InnovationLab-Task11"
25             BranchName      = "main"
26         }
27     }
28 }
```

Task 11.5: Configure AWS CodePipeline with source stage pointing to the repository

- Write a buildspec.yml file and upload it to the source repository in GitHub.
 - The build process involves installing all the required dependencies.
 - It Copies the application code into a zip file and uploads it to S3.

```
Code Blame 22 lines (18 loc) · 351 Bytes
1  version: 0.2
2  phases:
3    install:
4      runtime-versions:
5        nodejs: 18
6
7    pre_build:
8      commands:
9        - echo "Installing dependencies..."
10       - npm install --production
11
12    build:
13      commands:
14        - echo "Build completed on `date`"
15
16    post_build:
17      commands:
18        - echo "Preparing artifact..."
19
20  artifacts:
21    files:
22      - '**/*'
```

- Write the build stage of the AWS CodePipeline:

```
30 stage {
31   name = "Build"
32
33   action {
34     name          = "Build"
35     category      = "Build"
36     owner         = "AWS"
37     provider      = "CodeBuild"
38     input_artifacts = ["source_output"]
39     output_artifacts = ["build_output"]
40     version       = "1"
41
42     configuration = {
43       ProjectName = aws_codebuild_project.Task11-Build-Project-Zaeem.name
44     }
45   }
46 }
47
48 }
```

- We also must write the build project for the build stage to use in the pipeline

```

74 resource "aws_codebuild_project" "Task11-Build-Project-Zaeem" {
75     name           = "Task11-Build-Project-Zaeem"
76     description    = "Build project for Node.js application"
77     service_role   = var.codebuild_role_arn
78     build_timeout  = 20
79
80     artifacts {
81         type = "CODEPIPELINE"
82     }
83
84     environment {
85         compute_type      = "BUILD_GENERAL1_SMALL"
86         image              = "aws/codebuild/standard:7.0"
87         type              = "LINUX_CONTAINER"
88         image_pull_credentials_type = "CODEBUILD"
89     }
90
91     source {
92         type      = "CODEPIPELINE"
93         buildspec = "buildspec.yml"
94     }
95 }

```

Task 11.6: Define deploy stage in CodePipeline to deploy the app to Elastic Beanstalk

- Write the deploy stage for the AWS CodePipeline in the terraform file:
 - The deployment stage must include the names of the EB Application and Environment.
 - It also includes the build output, which will be a ZIP file in the artifact bucket.

```

50 stage {
51     name = "Deploy"
52
53     action {
54         name           = "Deploy"
55         category       = "Deploy"
56         owner          = "AWS"
57         provider       = "ElasticBeanstalk"
58         input_artifacts = ["build_output"]
59         version        = "1"
60
61         configuration = {
62             ApplicationName = var.eb_app_name
63             EnvironmentName = var.eb_env_name
64         }
65     }
66 }
67 }

```

Task 11.7: Set up IAM roles and permissions for CodePipeline and CodeBuild

1. CodePipeline Role

Permissions:

- Access to S3 bucket for artifacts
- Start/stop CodeBuild projects
- Deploy to Elastic Beanstalk
- Use CodeStar Connections (for GitHub)
- Create/manage CloudWatch Logs
- Manage ECS tasks and services

2. CodeBuild Role

Permissions:

- Read/write to S3 artifact bucket
- Push/pull from ECR (if using Docker)
- Create CloudWatch Logs
- Decrypt KMS keys (for encrypted artifacts)

3. Elastic Beanstalk Service Role

Permissions:

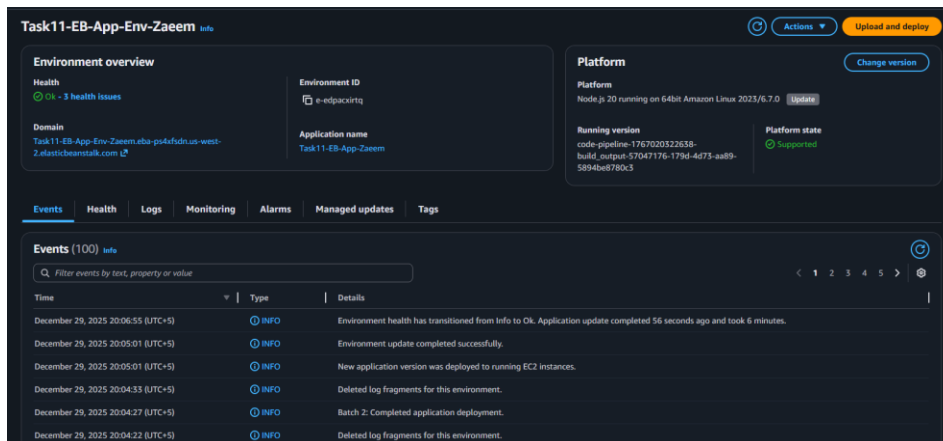
- Manage EC2 instances
- Configure Auto Scaling
- Manage Elastic Load Balancers
- Access CloudFormation
- Write CloudWatch Logs

4. EC2 Instance Profile

Permissions:

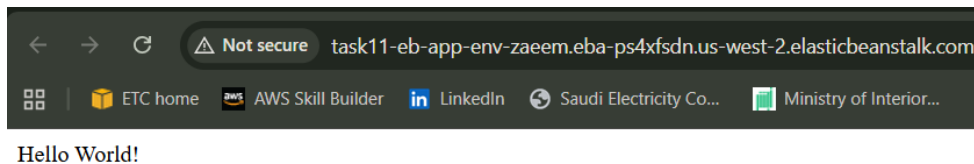
- Read from S3 (application artifacts)
- Write to CloudWatch Logs
- Pull from ECR (if needed)

- We can also check the deployment process in detail on the EB dashboard.



Task 11.9: Test the deployed Node.js application endpoint

- After the Pipeline has completed execution, we can head over to the domain provided by EB to access the webpage.



Task 11.10: Tips I have Learned.

- We do not need to configure a DockerFile for EB pipeline as we do not need to containerize the application. Hence, we do not need to push anything to ECR.
- We do not need to write an appspec file either because EB automatically handles deployment.
- The buildspec file only contains simple commands to install npm packages that are needed for the application to work.
- To update the node.js version used by the application, make changes to the package.json file and write the new version.
- If command execution fails on an instance, download the EB engine logs and find the error. Most of the time it is related to the proc file or a syntax error in the command.
- We can either use a Procfile to start the application or directly use the package.json file with the script to start the application.

