```
[6:23 pm, 05/11/2024] Sarish: #!/bin/bash
# install httpd (Linux 2 version)
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
echo "Hello World from $(hostname -f)" > /var/www/html/index.html
```

[8:42 pm, 05/11/2024] Sujeeth STJCE Friends: "Aim:

Design a secure Multi-Tier VPC on AWS with NAT gateways and VPN connections, enabling communication between AWS resources and on-premises infrastructure.

Steps:

1. Create a VPC

Go to AWS Console > VPC Dashboard.

Create a VPC with CIDR block: 10.0.0.0/16.

2. Create Subnets

Create a Public Subnet (e.g., 10.0.1.0/24).

Create a Private Subnet (e.g., 10.0.2.0/24).

3. Internet Gateway

Create an Internet Gateway (IGW) and attach it to your VPC.

4. Route Tables

Public Route Table: Add a route to the Internet Gateway for public subnet internet access (0.0.0.0/0).

Private Route Table: Use a NAT gateway for private subnet internet access.

5. NAT Gateway

Create a NAT Gateway in the public subnet for private subnet internet access.

Update the private route table to use this NAT gateway.

6. VPN Gateway (VGW)

Create a VPN Gateway and attach it to your VPC.

7. Customer Gateway

Create a Customer Gateway using your on-premises router’s public IP.

8. VPN Connection

Create a VPN connection between the VPN Gateway and Customer Gateway.

Add routes for your on-premises network in the VPC route table.

9. Download Configuration

Download the VPN configuration file to set up VPN on your on-premises router.

10. Update Route Tables

Add a route in the private route table for the on-premises network.

11. Security Groups

Set up security groups for the public and private subnets to control traffic.

12. Test Connectivity

Launch instances and verify:

Public subnet can access the internet directly.

Private subnet accesses the internet via the NAT Gateway.

Private subnet communicates with on-premises via VPN."

[8:42 pm, 05/11/2024] Sujeeth STJCE Friends: Here's a simplified version of the CI/CD pipeline implementation using Jenkins, AWS CodePipeline, and Blue/Green deployment:

Aim:

To design and implement a CI/CD pipeline using Jenkins, AWS CodePipeline, and Blue/Green deployment. This ensures automated, fast, and reliable delivery of applications with minimal downtime and deployment errors.

Steps:

Step 1: Set Up Jenkins

1. *Install Jenkins on EC2* (Amazon Linux or Ubuntu):

  - For Amazon Linux:

```
    sudo yum update -y
    sudo amazon-linux-extras install java-openjdk11
    sudo yum install jenkins -y
    sudo systemctl start jenkins
```

- For Ubuntu:

```
  sudo apt update -y
  sudo apt install openjdk-11-jre jenkins -y
  sudo systemctl start jenkins
```

2. *Access Jenkins*:
   - Go to http://<public_IP>:8080 to complete the Jenkins setup.

3. *Install Plugins*:
   - Install AWS CodePipeline, Git, Pipeline, and Blue Ocean plugins.

Step 2: Configure AWS CodePipeline

1. *Create an S3 Bucket* for storing artifacts.
2. *Create CodePipeline*:
   - Source Stage: Choose GitHub or CodeCommit for source.
   - Build Stage: Select Jenkins as the build provider and specify the Jenkins project.
   - Deploy Stage: Choose AWS Elastic Beanstalk or ECS for deployment.

Step 3: Set Up Jenkins Job for Build

1. *Create Jenkins Pipeline Job*:
   - Go to Jenkins > New Item > Pipeline.
   - Configure the Git repository and add a Jenkinsfile in the repository for CI/CD.

2. *Sample Jenkinsfile*:

```
groovy
pipeline {
  agent any
  stages {
    stage('Build') {
        steps {
          sh 'npm install'
          sh 'npm run build'
        }
      }
      stage('Test') {
        steps {
          sh 'npm test'
        }
      }
      stage('Package') {
        steps {
          sh 'zip -r build.zip .'
          archiveArtifacts artifacts: 'build.zip'
        }
      }
    }
  }
```

Step 4: Blue/Green Deployment

1. *Create Elastic Beanstalk Application*:

  - Go to Elastic Beanstalk Dashboard > Create Application.
  - Select platform (e.g., Node.js, Python, or Docker).
  - Enable Blue/Green deployment for zero-downtime updates.

2. *Configure Load Balancer* (optional) to manage traffic between environments.

Step 5: Integrate Jenkins with CodePipeline

1. *IAM Roles*: Set up an IAM role for Jenkins to access S3, CodePipeline, and Elastic Beanstalk.

2. *Jenkinsfile with AWS Integration*:

```
 groovy
 pipeline {
  environment {
   S3_BUCKET = 'my-app-pipeline-artifacts'
  }
  stages {
   stage('Upload to S3') {
    steps {
     sh 'aws s3 cp build.zip s3://$S3_BUCKET/'
    }
   }
   stage('Deploy') {
    steps {
     sh 'aws elasticbeanstalk update-environment --application-name my-app --version-label v1 --environment-name my-app-env'
    }
   }
  }
 }
```

Step 6: Test and Monitor the Pipeline

1. *Trigger the Pipeline* by pushing code changes.

2. *Monitor Deployment* using Elastic Beanstalk and AWS CloudWatch.

3. *Monitor Traffic*: Check Blue/Green traffic switching between environments.

Optional Enhancements:

- *Add Rollback Mechanism* for failed deployments.

- *Enable Auto-Scaling* to handle traffic spikes.

- *Set Notifications* for pipeline status using SNS or Slack.

Result:

Successfully implemented a CI/CD pipeline that automates build, test, and deployment, ensuring minimal downtime and fast delivery of new features.

[8:42 pm, 05/11/2024] Sujeeth STJCE Friends: Here’s a simplified walkthrough for implementing a serverless architecture using AWS Lambda, DynamoDB, and API Gateway:

Aim:

To design a fully serverless architecture using Amazon API Gateway, AWS Lambda, and DynamoDB for a scalable, cost-effective, and highly available application. This eliminates the need for server management and enables automatic scaling and rapid deployment.

Steps:

Step 1: Create a DynamoDB Table

1. *Navigate to DynamoDB in the AWS Console*.

2. *Create a Table*:

   - Name: Users

   - Primary Key: userId (String)

   - (Optional) Add a Sort Key for complex access patterns.

3. Leave other settings as default and click *Create Table*.

Step 2: Create an AWS Lambda Function

1. *Navigate to Lambda in AWS Console*.

2. *Create a new …

[8:42 pm, 05/11/2024] Sujeeth STJCE Friends: Here’s a step-by-step guide for implementing a disaster recovery (DR) solution using Amazon Route 53 and S3 Cross-Region Replication.

Aim:

To design a highly available and fault-tolerant disaster recovery solution using Amazon Route 53 for DNS failover and S3 Cross-Region Replication (CRR) to replicate data across AWS regions. This setup will minimize downtime during disasters and ensure seamless failover between regions.

Steps:

Step 1: Set Up S3 Cross-Region Replication (CRR)

This step ensures that your data is replicated across AWS regions.

1. *Create Source and Destination S3 Buckets*:

 - *Source Bucket*:

   - Go to the S3 console and create a new bucket (e.g., source-bucket).

   - Choose a primary region (e.g., us-east-1).

   - Enable *Versioning* for the bucket.

 - *Destination Bucket*:

   - Create a second S3 bucket in a different region (e.g., us-west-2, named destination-bucket).

   - Enable *Versioning* for this bucket as well.

2. *Configure Cross-Region Replication*:

 - Go to the *Management* tab in the source bucket.

 - Create a *Replication Rule* (e.g., replicate-to-west).

 - Choose the entire bucket or specific prefixes/tags to replicate.

 - Set the destination to the previously created destination bucket (destination-bucket) and select the destination region.

 - Create a new IAM role that allows replication between the two buckets.

 - Save the replication rule.

3. *Test Cross-Region Replication*:

 - Upload objects to the source bucket.

 - Verify that the objects are automatically replicated to the destination bucket in the second region.

Step 2: Set Up Route 53 DNS Failover

Configure Amazon Route 53 to provide automatic DNS failover between your primary and secondary regions in the event of a failure.

1. *Create a Route 53 Hosted Zone*:

 - Navigate to the Route 53 console.

- Create a new *Hosted Zone* for your domain (e.g., example.com).

2. *Set Up Health Checks*:

  - Go to the *Health Checks* section in Route 53.

  - Create a health check for your primary server in Region A (e.g., us-east-1). Configure the endpoint with the server's public IP or domain.

  - Create another health check for the server in the secondary region (e.g., us-west-2).

3. *Create Route 53 DNS Records for Failover*:

  - Create a *Record Set* for the primary region (e.g., www.example.com or root domain).

    - Set the *Routing Policy* to *Failover*.

    - Set the record type as *Primary* and associate it with the primary health check.

  - Create another *Record Set* for the secondary region.

    - Set the *Routing Policy* to *Failover* and the type as *Secondary*.

    - Associate it with the health check of the secondary region.

Step 3: Test the Disaster Recovery Solution

To ensure the DR solution is correctly configured, perform a disaster recovery drill.

1. *Simulate a Failure*:

  - Stop or disable services in the primary region (EC2, RDS, etc.).

  - Monitor the Route 53 health checks; traffic should automatically failover to the secondary region once the primary region fails the health check.

2. *Verify Cross-Region Replication*:

  - Confirm that data replicated via S3 is accessible in the secondary region.

  - Ensure users can interact with the application hosted in the secondary region without experiencing downtime.

3. *Restore Normal Operations*:

  - Once the primary region is operational, Route 53 will automatically restore traffic to the primary region if it passes health checks again.

Step 4: Set Up Monitoring and Alerts

Monitoring the disaster recovery setup ensures that you are informed of any issues in real-time.

1. *CloudWatch Alarms*:

   - Create CloudWatch alarms to monitor Route 53 health checks and trigger alerts when failover occurs.

   - Set up alarms for S3 Cross-Region Replication failures.

2. *SNS Notifications*:

   - Set up Amazon SNS to receive notifications when failover happens or when replication issues occur.

Step 5: Optimize for Cost and Performance

1. *Auto Scaling*: Use auto-scaling to adjust EC2 resources dynamically in both regions based on traffic demands.

2. *Cost Optimization*: Implement lifecycle policies in S3 to delete old data, and clean up unnecessary snapshots or AMIs to reduce costs.

3. *Regular Testing*: Periodically test the disaster recovery system to ensure it's fully operational.

Result:

A highly available and fault-tolerant disaster recovery solution was successfully implemented using Route 53 DNS failover and S3 Cross-Region Replication. The solution ensures minimal downtime and seamless failover between AWS regions in the event of a disaster