Deploy Java Application on AWS 3-Tier Architecture

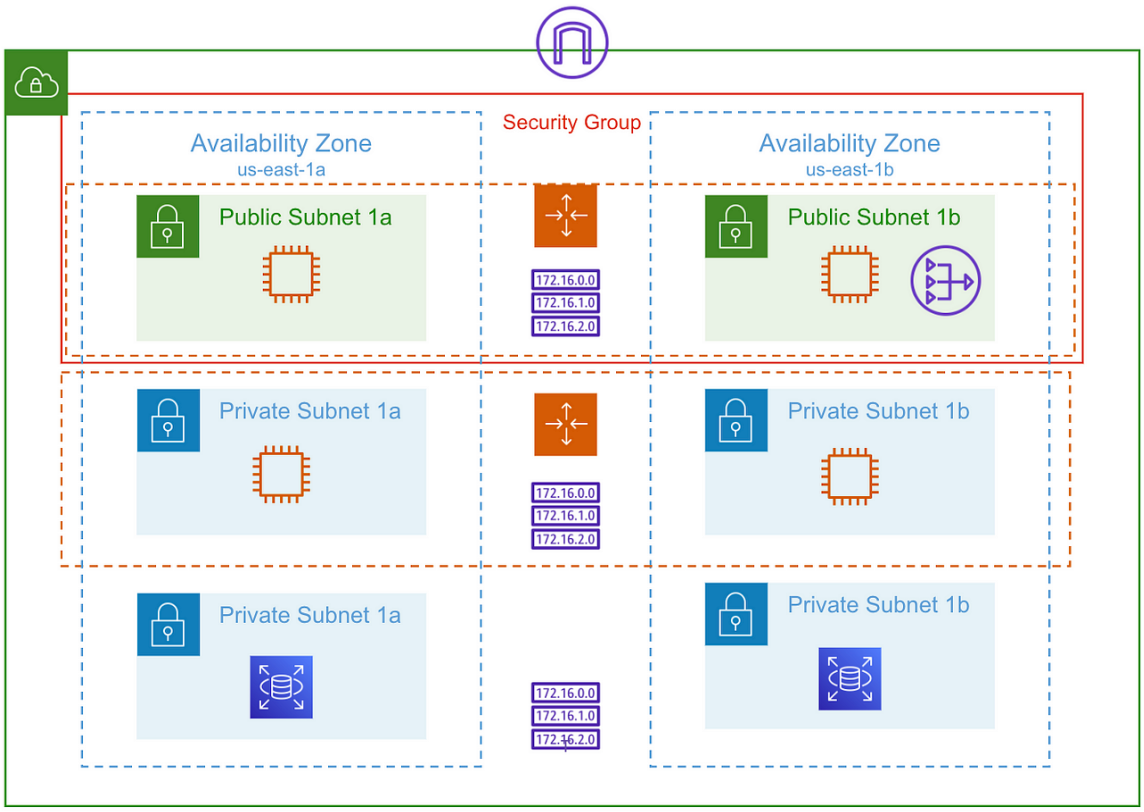
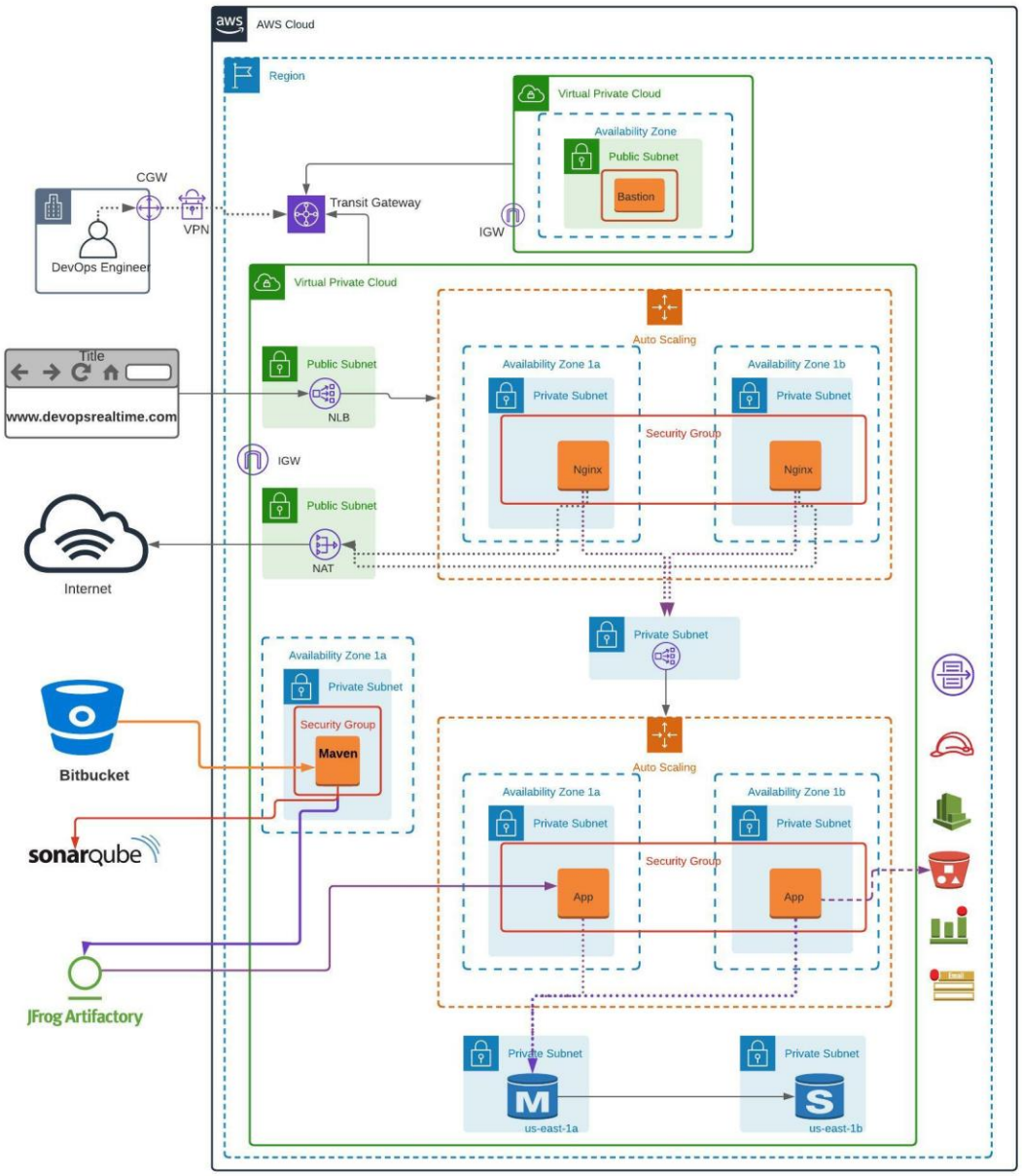


Table of Contents



Project Overview

Goal

The primary objective of this project is to deploy a scalable, highly available, and secure Java application using a 3-tier architecture. The application will be hosted on AWS,

utilizing various services like EC2, RDS, and VPC to ensure its availability, scalability, and security. The application will be accessible to end-users via the public internet.

Pre-Requisites

Before starting the deployment, ensure you have the following:

1. **AWS Free Tier Account**:
    a. Sign up for an Amazon Web Services (AWS) Free Tier account.
    b. Set up the AWS CLI on your local machine.
    c. Configure your CLI with `aws configure`.
2. **GitHub Account and Repository**:
    a. Create a GitHub account if you don't have one.
    b. Fork the Java source code from the Java-Login-App repository to your GitHub account.
3. **SonarCloud Account**:
    a. Create an account on SonarCloud for static code analysis and code quality checks.
    b. Generate a SonarCloud token for integration with your GitHub repository.
4. **JFrog Cloud Account**:
    a. Sign up for a JFrog Cloud account.
    b. Set up a Maven repository in JFrog to store your build artifacts.

Pre-Deployment

Create Global AMI (Amazon Machine Image)

Creating a Global AMI involves installing necessary agents and software on an EC2 instance, which will be used to create custom AMIs for different components.

1. **Install AWS CLI**:
    a. Install AWS CLI by following the instructions here.

```
curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "AWSCLIV2.pkg"
sudo installer -pkg AWSCLIV2.pkg -target /
aws --version
```

2. **Install CloudWatch Agent**:

a. Install CloudWatch Agent to monitor your EC2 instances.

```
sudo yum install amazon-cloudwatch-agent
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a start
```

3. **Install AWS Systems Manager (SSM) Agent**:
    a. The SSM Agent is necessary for managing EC2 instances via AWS Systems Manager.

```
sudo yum install amazon-ssm-agent
sudo systemctl start amazon-ssm-agent
sudo systemctl enable amazon-ssm-agent
```

Create Golden AMIs

Golden AMIs will be created for the different tiers (Nginx, Tomcat, Maven) of the architecture.

1. **For Nginx**:
    a. Launch an EC2 instance and install Nginx:

```
sudo amazon-linux-extras install nginx1.12
sudo systemctl start nginx
sudo systemctl enable nginx
```

b. Create a custom CloudWatch metric for memory usage:

```
#!/bin/bash
while true; do
  memory_usage=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
  aws cloudwatch put-metric-data --metric-name MemoryUsage --namespace Custom --value
$memory_usage --dimensions InstanceId=$(curl http://169.254.169.254/latest/meta-
data/instance-id)
  sleep 60
done &
```

2. **For Apache Tomcat**:
    a. Launch another EC2 instance and install Apache Tomcat:

```
sudo yum install java-11-openjdk-devel
wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.53/bin/apache-tomcat-9.0.53.tar.gz
sudo tar -xvzf apache-tomcat-9.0.53.tar.gz -C /opt/
sudo ln -s /opt/apache-tomcat-9.0.53 /opt/tomcat
sudo sh /opt/tomcat/bin/startup.sh
```

b. Configure Tomcat as a systemd service:

```
sudo nano /etc/systemd/system/tomcat.service
```

Add the following content:

```
[Unit]
Description=Apache Tomcat Web Application Container
After=network.target

[Service]
Type=forking

Environment=JAVA_HOME=/usr/lib/jvm/jre
Environment=CATALINA_PID=/opt/tomcat/temp/tomcat.pid
Environment=CATALINA_HOME=/opt/tomcat
Environment=CATALINA_BASE=/opt/tomcat
Environment='CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC'
Environment='JAVA_OPTS=-Djava.awt.headless=true -
Djava.security.egd=file:/dev/./urandom'

ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

User=tomcat
Group=tomcat
UMask=0007
RestartSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

c. Enable and start the Tomcat service:

```
sudo systemctl daemon-reload
sudo systemctl start tomcat
sudo systemctl enable tomcat
```

3. **For Apache Maven Build Tool**:

a. Install Maven, Git, and JDK 11 on a separate EC2 instance:

```
sudo yum install git
sudo yum install java-11-openjdk-devel
wget https://mirrors.ocf.berkeley.edu/apache/maven/maven-3/3.8.4/binaries/apache-maven-3.8.4-bin.tar.gz
sudo tar -xvzf apache-maven-3.8.4-bin.tar.gz -C /opt/
sudo ln -s /opt/apache-maven-3.8.4 /opt/maven
```

b. Update the system PATH:

```
echo "export M2_HOME=/opt/maven" | sudo tee -a /etc/profile.d/maven.sh
echo "export PATH=\$M2_HOME/bin:\$PATH" | sudo tee -a /etc/profile.d/maven.sh
source /etc/profile.d/maven.sh
```

c. Verify the installation:

```
mvn -version
```

VPC Deployment

Network Setup

1. **Create VPCs**:
    a. Create two VPCs for the application architecture:

```
aws ec2 create-vpc --cidr-block 192.168.0.0/16 --region us-east-1
aws ec2 create-vpc --cidr-block 172.32.0.0/16 --region us-east-1
```

2. **NAT Gateway and Internet Gateway**:
    a. Set up a NAT Gateway in the public subnet and an Internet Gateway for outbound access:

```
aws ec2 create-nat-gateway --subnet-id subnet-xxxx --allocation-id eipalloc-xxxx
aws ec2 create-internet-gateway
aws ec2 attach-internet-gateway --vpc-id vpc-xxxx --internet-gateway-id igw-xxxx
```

3. **Transit Gateway**:
    a. Create a Transit Gateway to allow communication between the VPCs:

```
aws ec2 create-transit-gateway
aws ec2 attach-transit-gateway-vpc-attachment --transit-gateway-id tgw-xxxx --vpc-id
vpc-xxxx
```

4. **Route Tables**:
    a. Update route tables for the private and public subnets to route traffic through the NAT Gateway and Internet Gateway:

```
aws ec2 create-route-table --vpc-id vpc-xxxx
aws ec2 create-route --route-table-id rtb-xxxx --destination-cidr-block 0.0.0.0/0 --
gateway-id igw-xxxx
```

Bastion Host Setup

1. **Deploy Bastion Host**:
    a. Launch an EC2 instance in the public subnet as a Bastion Host:

```
aws ec2 run-instances --image-id ami-xxxx --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids sg-xxxx --subnet-id subnet-xxxx --associate-public-ip-address
```

2. **Security Groups**:
    a. Create

a security group allowing SSH access from your IP:

```
aws ec2 create-security-group --group-name BastionSG --description "Security group for Bastion Host" --vpc-id vpc-xxxx
aws ec2 authorize-security-group-ingress --group-id sg-xxxx --protocol tcp --port 22 --cidr YOUR_IP/32
```

Maven (Build)

EC2 Instance for Maven Build

1. **Launch EC2 Instance Using Maven AMI**:
    a. Launch an EC2 instance using the Maven Golden AMI:

```
aws ec2 run-instances --image-id ami-maven --count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids sg-maven --subnet-id subnet-maven
```

2. **Clone the Repository**:
    a. Connect to the instance and clone your GitHub repository:

```
git clone https://github.com/YOUR_USERNAME/YOUR_REPO.git
cd YOUR_REPO
```

3. **Update `pom.xml`**:
    a. Update `pom.xml` with SonarCloud and JFrog deployment details.
4. **Build the Application**:
    a. Run the Maven build command to compile the project:

```
mvn clean install -s settings.xml
```

5. **SonarCloud Integration**:
    a. Integrate SonarCloud with Maven:

```
mvn sonar:sonar -Dsonar.projectKey=YOUR_PROJECT_KEY -Dsonar.organization=YOUR_ORGANIZATION -Dsonar.host.url=https://sonarcloud.io -Dsonar.login=YOUR_SONAR_TOKEN
```

3-Tier Architecture

Database Tier (RDS)

1. **Deploy MySQL RDS**:
    a. Create a MySQL RDS instance in the private subnet:

```
aws rds create-db-instance --db-instance-identifier mydbinstance --db-instance-class db.t2.micro --engine mysql --allocated-storage 20 --master-username admin --master-user-password password --vpc-security-group-ids sg-db --db-subnet-group-name mydbsubnetgroup
```

Backend Tier (Tomcat)

1. **Network Load Balancer**:
    a. Create a private-facing Network Load Balancer:

```
aws elbv2 create-load-balancer --name private-nlb --subnets subnet-xxxx subnet-yyyy --type network
```

2. **Auto Scaling Group**:
    a. Set up an Auto Scaling Group for the Tomcat instances:

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name tomcat-asg --launch-configuration-name TomcatLC --min-size 1 --max-size 3 --vpc-zone-identifier subnet-xxxx,subnet-yyyy
```

Frontend Tier (Nginx)

1. **Public Network Load Balancer**:
    a. Create a public-facing Network Load Balancer:

```
aws elbv2 create-load-balancer --name public-nlb --subnets subnet-xxxx subnet-yyyy --type network
```

2. **Auto Scaling Group**:
    a. Set up an Auto Scaling Group for the Nginx instances:

```
aws autoscaling create-auto-scaling-group --auto-scaling-group-name nginx-asg --launch-configuration-name NginxLC --min-size 1 --max-size 3 --vpc-zone-identifier subnet-xxxx,subnet-yyyy
```

Application Deployment

Deploying Artifacts

1. **User Data Scripts**:
    a. Use user data scripts to deploy artifacts during the EC2 instance launch. Ensure the script fetches the `.war` file from JFrog and places it in the Tomcat `webapps` directory.
2. **Database Setup**:
    a. Log into the MySQL database from the Tomcat server and create the database schema:

```
mysql -h mydbinstance.123456789012.us-east-1.rds.amazonaws.com -u admin -p
CREATE DATABASE mydatabase;
USE mydatabase;
SOURCE /path/to/schema.sql;
```

Post-Deployment

Logging and Monitoring

1. **Cron Job for Logs**:
    a. Set up a Cron job to push Tomcat logs to an S3 bucket:

```
crontab -e
```

Add the following entry:

```
0 0 * * * /usr/bin/aws s3 cp /path/to/tomcat/logs s3://mybucket/tomcat-logs/ --
recursive --region us-east-1 && rm -rf /path/to/tomcat/logs/*
```

2. **CloudWatch Alarms**:
    a. Set up CloudWatch alarms to monitor database connections:

```
aws cloudwatch put-metric-alarm --alarm-name "DBConnectionsHigh" --metric-name
DatabaseConnections --namespace AWS/RDS --statistic Average --period 300 --threshold
100 --comparison-operator GreaterThanOrEqualToThreshold --evaluation-periods 1 --
alarm-actions arn:aws:sns:us-east-1:123456789012:MySNSTopic
```

Validation

1. **EC2 Access**:

    a. Verify SSH access to EC2 instances via the Bastion Host and Systems Manager.
2. **Application Accessibility**:
    a. Ensure the application is accessible from a public internet browser.