**AWS Full Stack Deployment With Kafka**

Note -- Make sure to test everything running in backend successfully check on postman than go with frontend deployment.

Step 1. Create a S3 Bucket and set permission as

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "PublicReadObject",

"Effect": "Allow",

"Principal": "\*",

"Action": "S3:GetObject",

"Resource": "arn:aws:s3:::ctsnewbucket1/\*"

}

]

}

Step 2. Create RDS Database in AWS. Try to configure port other than your local MySQL port 3306.

And connect it with your local like MySQL Workbench.

Step 3. Make a suitable EC2 instance according to your backend need.

(i) Key Pair creation

(ii) Network Settings

- Enable IP

- Security groups

select your Security groups> inbound rules> edit inbound rules>

Add rule

- All HTTP

- All HTTPS

- All TCP

- All Traffic

Step 4. Allocate an Elastic IP Address if it can be possible. Because there are limited number of EIP

Step 5. Install Java in EC2 Instance

===================================

1. To get the most recent archive, go to the JDK 17 releases page.

Wget https://download.java.net/java/GA/jdk17.0.2/dfd4a8d0985749f896bed50d7138ee7f/8/GPL/openjdk-17.0.2\_linux-x64\_bin.tar.gz

b. Using the tar command, extract the downloaded OpenJDK 17 archive file.

tar xvf openjdk-17.0.2\_linux-x64\_bin.tar.gz

c. Place the resultant folder in the /opt directory.

sudo mv jdk-17.0.2/ /opt/jdk-17

d. Now, configure Java environment:

sudo tee /etc/profile.d/jdk.sh <<EOF

export JAVA\_HOME=/opt/jdk-17

export PATH=\$PATH:\$JAVA\_HOME/bin

EOF

e. Source your profile file:

source /etc/profile.d/jdk.sh

f. Check the java command JAVA\_HOME:

$ echo $JAVA\_HOME

/opt/jdk-17

g. Confirm Java version:

$ java -version

openjdk version "17.0.2" 2022-01-18

OpenJDK Runtime Environment (build 17.0.2+8-86)

OpenJDK 64-Bit Server VM (build 17.0.2+8-86, mixed mode, sharing)

===============================================================================

Step 6. Install Kafka in EC2

wget https://dlcdn.apache.org/kafka/3.6.1/kafka\_2.13-3.6.1.tgz

tar xvf kafka\_2.13-3.6.1.tgz

cd ./kafka\_2.13-3.6.1/config/

vim server.properties

listeners=PLAINTEXT://:9092

advertised.listeners=PLAINTEXT://hot-name:9092 (change host-name with your EC2 DNS)

cd ./bin

vim kafka-server-start.sh

export KAFKA\_HEAP\_OPTS="-Xmx512M -Xms512M" (If giving some memory related error, change this accordingly)

./kafka\_2.13-3.6.1/bin/zookeeper-server-start.sh ./kafka\_2.13-3.6.1/config/zookeeper.properties (to run Zookeeper)

./kafka\_2.13-3.6.1/bin/kafka-server-start.sh ./kafka\_2.13-3.6.1/config/server.properties( to run Kafka Server)

===========================================================================================================

Step 7: Now to create the jar file of those service, mention the EC2 DNS in application.properties or .yml file where localhost is mention.

And also mention the RDS URL and Port .

Step 8: Create the jar file and also make sure that Zookeeper and Kafka Server are running in EC2 then create the jar file.

Step 9: Upload the jar files in S3 Bucket.

Step 10. Use wget command to get jars from s3 bucket in EC2 instance

Step 11. Run the jar file java -jar name.jar

step 12. Create API Gateway REST API

Make requests according to your needs

If you are using @Pathvariable

Make proxyresource using {proxy+} <--- IMPORTANT

NOTE VERY IMPORTANT: HTTPS URL's must be same as your local's URL

Step 13. After creating all REST API's, Enable CORS and Deploy the API's

Step 14. Change the service layer of Angular with Deployed HTTPS API's

Step 15. Build the Angular project using ng build

Note: Sometimes it may give error like `link.parentnode.insertBefore is not a function'

Solution:

- npm install

- ng build --configuration=development

Step 16. Upload the dist folder in S3 bucket.

Step 17. Go to S3 Bucket and make index.html as static hosting.Go to properties tab and enable static hosting but it will give you HTTP.

Step 18. To get HTTPS add the below bucket policy.

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::bustertimes/\*",

"Condition": {

"Bool": {

"aws:SecureTransport": true

}

}

}

]

}

Step 19. Create a cloud front distribution select your s3 bucket use WAF(Web Application Firewall), in document root enter your filename(index.html)

Step 20. After that HTTPS URL of deployed application can be used.

Now after following above steps, we can successfully deploy the application in AWS Cloud.