**AWS ETL SETUP**

**Graphical user interface

Description automatically generated with medium confidence**

**1. S3 -> LAMBDA**

1. Create a S3 Bucket (link: <https://youtu.be/i4YFFWcyeFM>)

->must be publicly accessible

1. Create a Lambda function (link: <https://youtu.be/6RmCH6KOLpA>)

->add the required IAM role (lambda to glue and s3 full access)

->connect to s3 and connect to glue

->get the bucket and key name from the event

->invoke the glue job by passing the name

1. Trigger the Lambda when a file is uploaded in the S3 bucket

->In lambda console go to ADD TRIGGER

->select a source (S3 bucket)

->select bucket name, event type (all objects create events), acknowledge and add.

**2.LAMBDA -> GLUE**

1. Create a GLUE

* Choose a source (S3) and destination (choose later)
* Click on create then choose
  + S3 source type (S3 location)
  + Give S3 URL
  + Choose Data format
  + Choose an IAM role (permission of s3 and connecting db)
  + Language (python)
* After giving details Save the job
* In the script, add the mongo dB details as code

host ="3.84.239.167"

port ="27017"

database="mongo-data"

collection="first-collection"

mongo-URL = 'mongodb://'+host+':'+port

mongo-Client = MongoClient(mongo-URL)

mongo-dB= mongo-Client[database]

mongo-Collection = mongo-dB[collection]

data=[{"item":"lol","qty":8},{"item":"lolipop","qty":80}]

for i in data:

Save= mongo-Collection.insert\_one(i)

**3.S3 -> Glue**

s3 = boto3.resource('s3')

bucket = s3.Bucket('demoglueebucket')

for obj in bucket.objects.all():

key = obj.key

body = obj.get()['Body'].read()

print("Body : ",body)

**4.GLUE -> EC2 -> MONGODB**

1. Create an EC2 (link: <https://youtu.be/oqHfiRzxunY>)( https://awswithatiq.com/setup-mongodb-server-in-aws/)

* In the Instance Security Group add Inbound Rules (add rule for Custom TCP from port 27017, anywhere ipv4
* Go to the EC2 Dashboard, check the instance, and click on connect.
* Connect to instance will open, connect / connect using putty using ppk file
* create editor file using vi (command: vi /etc/yum.repos.d/mongodb-org-4.2.repo)
* An editor will open, press I to insert then paste the code.

[mongodb-org-4.2]

name=MongoDB Repository baseurl=https://repo.mongodb.org/yum/amazon/2013.03/mongodb-org/4.2/x86\_64/

gpgcheck=1

enabled=1 gpgkey=https://www.mongodb.org/static/pgp/server-4.2.asc

* Then press esc, then :wq
* Install mongodb using sudo yum install -y mongodb-org
* Start MongoDB service usingsudo service mongod start
* Start MongoDB on reboot using sudo chkconfig mongod on
* Connect to mongo shell using mongosh
* use demo-db to create db
* db.collection.insert({"SampleValue1" : 255, "SampleValue2" : "randomStringOfText"}) to create collection
* After these steps keep note of EC2 private key URL, database name, collection name, port of mongo
* sudo vi /etc/mongod.conf to change the bind ip to 0.0.0.0

**CDK**

Link: <https://www.youtube.com/watch?v=I2cXlYYoQqQ>

https://docs.aws.amazon.com/cdk/v2/guide/work-with-cdk-python.html

Steps to provide Access Key: -

1. Click on IAM Role
2. Create a User with its name and give full Administrator Access in policy
3. Click on Create and go to security credentials then create access key.

Prerequisite:

1.npm

2.python

3.aws cli

4.python -m ensurepip --upgrade

5.python -m pip install --upgrade pip

6.python -m pip install --upgrade virtualenv

7.npm install -g aws-cdk

8. pip install --upgrade aws-cdk.core

9. cdk init sample-app --language python

10. source .venv/bin/activate

11. python -m pip install -r requirements.txt

10.cdk ls

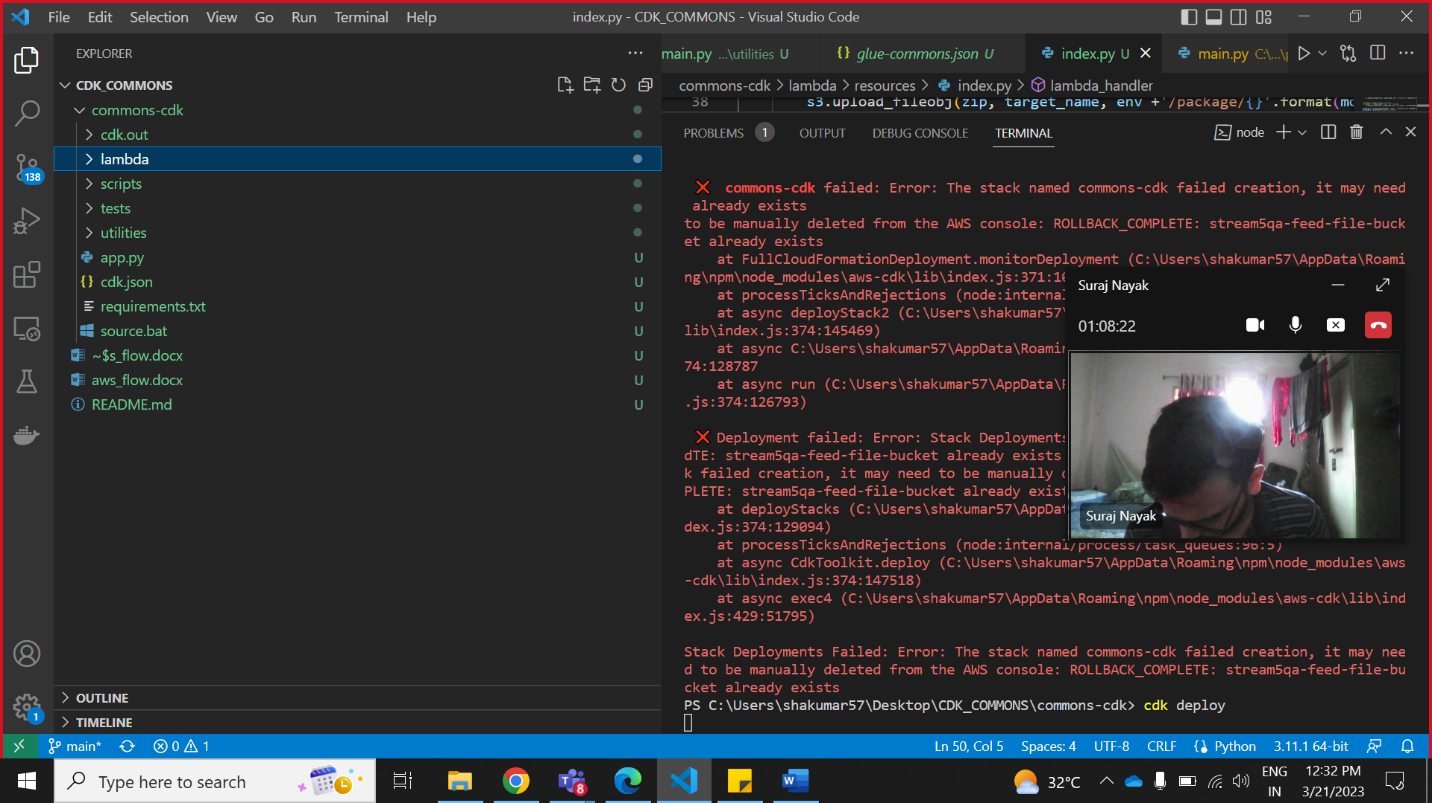
11.cdk synth

12.cdk bootstrap

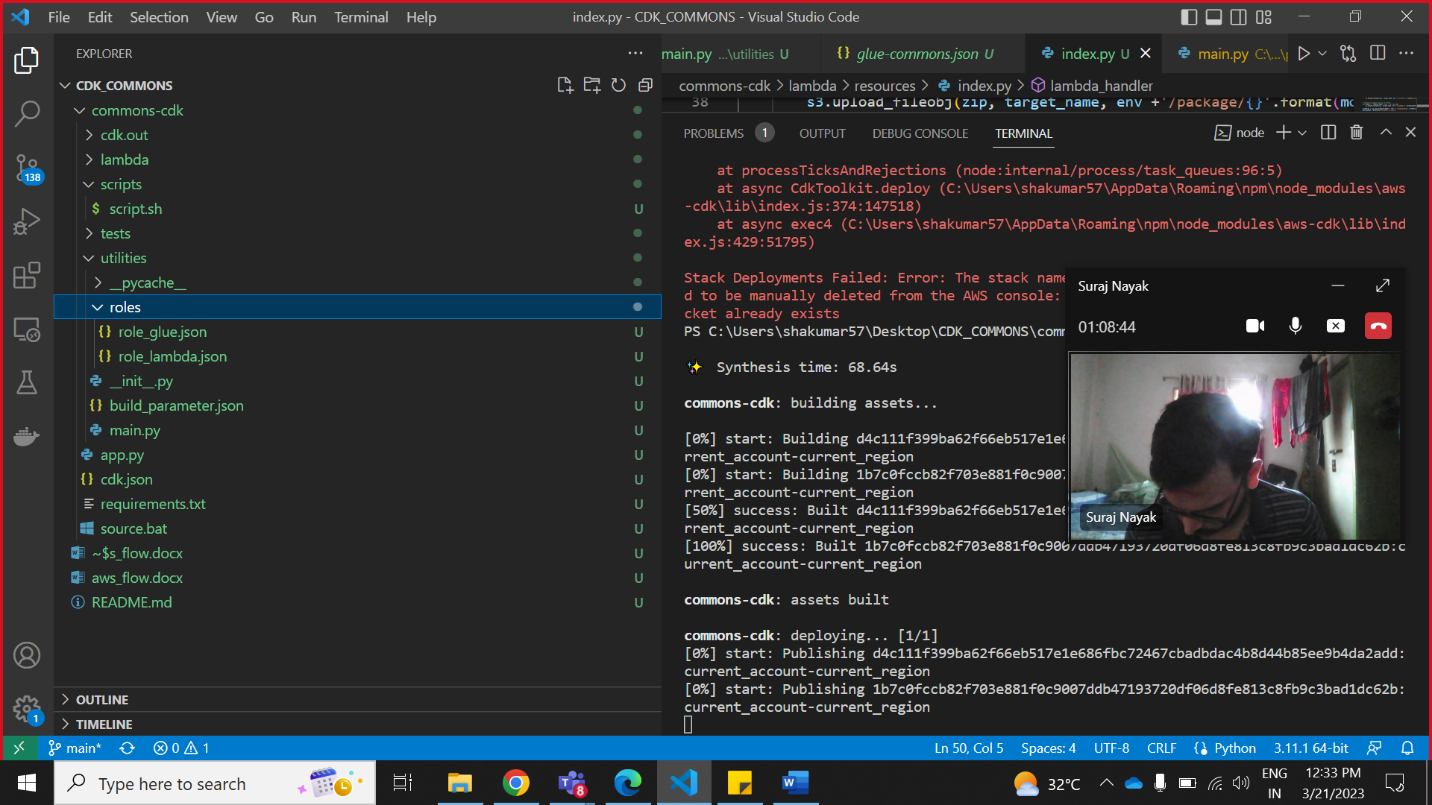
13.cdk deploy

**Folder Structure**

Here the all the folders are arranged in order

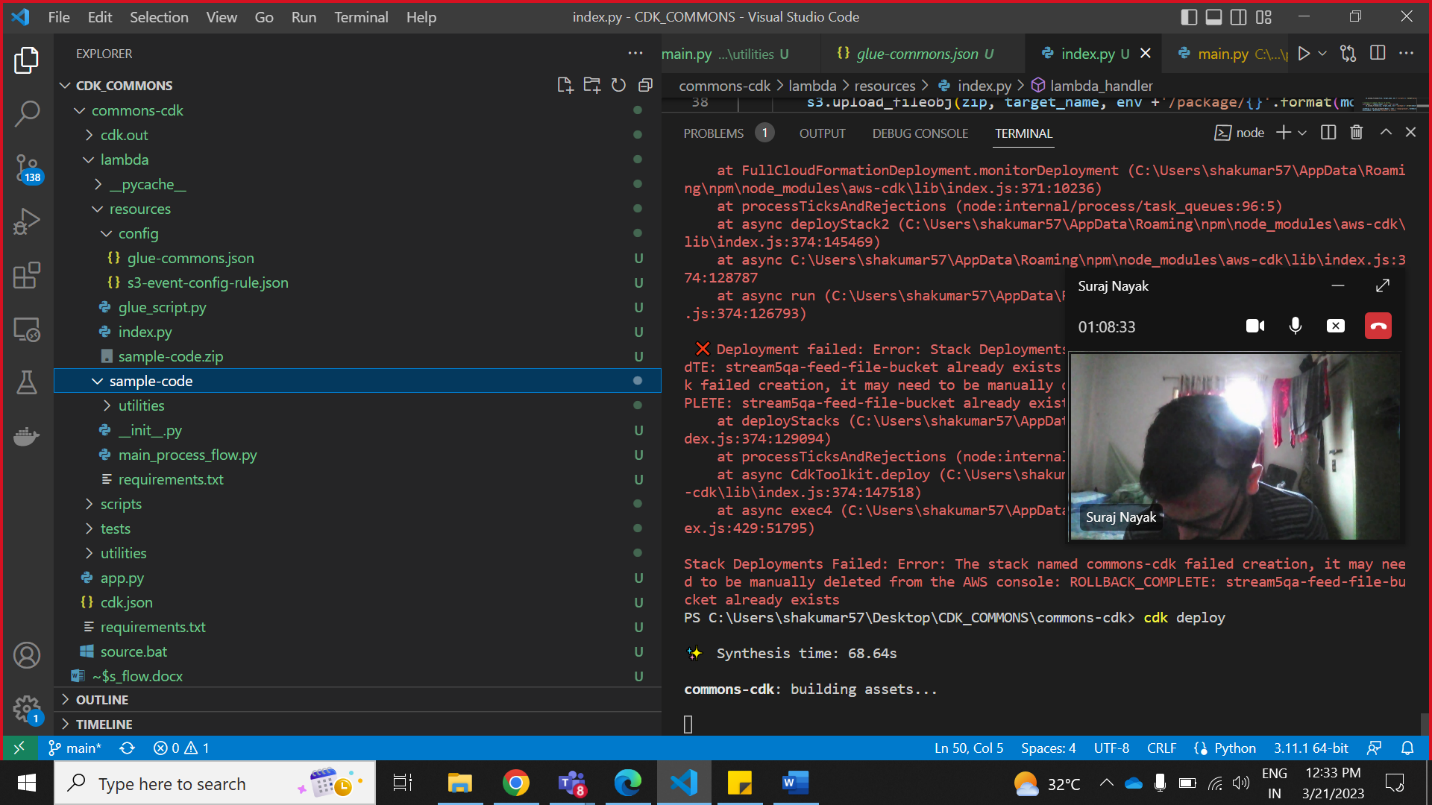
****

Here the scripts contains the script.sh where installation of mongo in ec2 is given and in utilities folder roles folder contain the roles used in aws services and build\_parameter.json contains all the values for parameter store

****

Here the lambda folder contains the resources folder which have config folder for validating eventkey and providing which database used.

It also contains glue\_script.py for performing glue job operation and index.py is the code for lambda

****

Creating S3 bucket

bucket\_script=s3.Bucket(self,"ScriptBucket",bucket\_name=env+"-"+script\_bucket)

Create lambda function

s3\_gluelambda = \_lambda.Function(self,

                                         "lambda\_func",

                                         function\_name=env + "\_etl\_func",

                                         runtime=\_lambda.Runtime.PYTHON\_3\_7,

                                         handler="index.lambda\_handler",

                                         code=\_lambda.InlineCode(s3func\_code),

                                         timeout=Duration.seconds(60),

                                         environment=dict(

                                             BUCKET=bucket\_feed.bucket\_name),

                                         role=lambda\_job\_role

                                         )

Put data in lamda

try:

            with open("serverless\_stacks/lambda\_src/s3togluefunc.py", mode="r") as f:

                s3func\_code = f.read()

        except OSError:

            print("Unable to Read")

IN the s3togluefunct.py the lambda code is present

Create trigger to lambda through s3

 bucket.add\_object\_created\_notification(

            notify.LambdaDestination(s3\_gluelambda)

        )

Create glue job

glue\_job = \_glue.CfnJob(self,'demoGlueJob',role=glue\_job\_role.role\_name,glue\_version="2.0",

name=env+"\_demoGlueJob",max\_retries=0,no\_of\_workers=10,worker\_type="G.1X",

default\_arguments={'--extra-py-files': "s3://"+bucket\_script.bucket\_name+"/commons/pace-glue-commons.zip"},

command=\_glue.CfnJob.JobCommandProperty(name='glueetl',

python\_version=os.getenv('PYTHON\_VERSION',"3"),

script\_location=f"s3://"+bucket\_script.bucket\_name+"/glue\_script.py"))

Adding iam role to lambda

with open(roles[0], mode="r") as f:

            lambda\_policy = json.load(f)

        poli1 = list(lambda\_policy["Action"])

        rsc1 = list(lambda\_policy["Resource"])

        policy\_statement1 = iam.PolicyStatement(actions=poli1, resources=rsc1)

        lambda\_job\_role = iam.Role(

            self,

            env+'-lambda-job-role',

            assumed\_by=iam.ServicePrincipal('lambda.amazonaws.com'))

        lambda\_job\_role.add\_to\_policy(policy\_statement1)

Adding iam role to glue

with open(roles[1], mode="r") as f:

            glue\_policy = json.load(f)

        policy = list(glue\_policy["Action"])

        rsc = list(glue\_policy["Resource"])

        policy\_statement = iam.PolicyStatement(

            actions=policy,

            resources=rsc)

        glue\_job\_role = iam.Role(self,

                                 f'{env}-glue-job-role',

                                 assumed\_by=iam.ServicePrincipal('glue.amazonaws.com'))

        glue\_job\_role.add\_to\_policy(policy\_statement)

VPC

ETL\_VPC = ec2.Vpc(self, "ETLVpc", max\_azs=3)

SECURITY GROUPS

        ETL\_SG = ec2.SecurityGroup(self, 'web-server-sg',

                                   vpc=ETL\_VPC,

                                   allow\_all\_outbound=True,

                                   description='security group for a web server'

                                   )

ADDING INBOUND PERMISSIONS

ETL\_SG.add\_ingress\_rule(

            ec2.Peer.any\_ipv4(),

            ec2.Port.tcp(22),

            description='allow SSH access from anywhere',

        )

MACHINE IMAGE

amz\_linux\_ami = ec2.MachineImage.latest\_amazon\_linux(

            generation=ec2.AmazonLinuxGeneration.AMAZON\_LINUX\_2,

            edition=ec2.AmazonLinuxEdition.STANDARD,

            storage=ec2.AmazonLinuxStorage.EBS,

            virtualization=ec2.AmazonLinuxVirt.HVM

        )

KEY PAIR

cfn\_key\_pair = ec2.CfnKeyPair(self, "MyCfnKeyPair",

                                      key\_name=env + "\_etl\_key"

                                      )

EC2 INSTANCE LAUNCH

web\_server = ec2.Instance(

                self,

                "myInstance1",

                instance\_type=ec2.InstanceType(

                    instance\_type\_identifier="t2.micro"),

                instance\_name=env + "\_etl\_instance",

                machine\_image=amz\_linux\_ami,

                vpc=ETL\_VPC,

                security\_group=ETL\_SG,

                key\_name=env + "\_etl\_key",

                vpc\_subnets=ec2.SubnetSelection(

                    subnet\_type=ec2.SubnetType.PUBLIC

                ),

                user\_data=ec2.UserData.custom(user\_data)

            )

Points to remember For the use of User data while launching: -

1. #!/bin/bash- is mandatory for automation
2. Don’t use sudo because these are executed in root.
3. Use -y to avoid user interference
4. As it is executed in root we can’t find the user data execution in non-root site as we have to enter to the root by cmd- “sudo -i” then the execution can be find.
5. Open vi editor and write data

echo"data needed to write"> /etc/yum.repos.d/mongodb-org-6.0.repo

PARAMETER STORE

 publicIPParam = ssm.StringParameter(self, 'ec2publicip',

 parameter\_name='/'+env + '/myapp/ec2cdk',

 string\_value=web\_server.instance\_public\_ip,

 description='this is used to get the public ip of ec2',

 type=ssm.ParameterType.STRING,

 tier=ssm.ParameterTier.STANDARD )

CALLING PARAMETER STORE

parameter = ssm.get\_parameter(Name='/'+env+'/myapp/gluejob',WithDecryption=True)

glueJobName= parameter ['Parameter']['Value']

IAM ROLE FILE FORMAT

{

    "Effect": "Allow",

    "Action": [],

    "Resource": [],

}

LOCAL SETUP FILE

[

    {

        "ParameterKey": "env",

        "ParameterValue": "stream5qa"

    }

]

SETUP FUNCTION

def read\_setup\_file():

->local setup file has parameter key and value mentioned.

->the file is read by read setup function.

->it loads all the data in list and separately send to the main file

ADD GITIGNORE FILE

->touch .gitignore

->inside that file add the files u need to ignore and then push the code again

->If u need to ignore a file add file.extension and if directory then add dir/ or if a extension then \*.extension

IMPLEMENTING PYTHON LIB PATH

🡪 This code used to add the python lib path through cdk in aws glue

default\_arguments={'--extra-py-files': “give the s3 bucket location of your file”}

example:-default\_arguments={'--extra-py-files': "s3://"+bucket\_script.bucket\_name+"/commons/pace-glue-commons.zip"}

🡪 Import the .py file to be used inside the glue\_script.py

🡪 Call the function you want to implement inside the .py file

🡪 If your library only consists of a single Python module in one .py file, you do not need to place it in a .zip file.

ADD FILES TO A FOLDER IN S3 BUCKET

with open("path of file to be uploaded", "rb") as file:

     client.upload\_fileobj(file,bucket\_name,file\_name)

if u need to create directory then in the file\_name then add the path and then the file\_name

example: '/folders/{}'.format("file\_name"))

HOW TO AUTOMATICALLY ZIP A FOLDER

shutil.make\_archive("output\_file\_name","extension","input\_file\_name")

## pip install -U -r requirements.txt