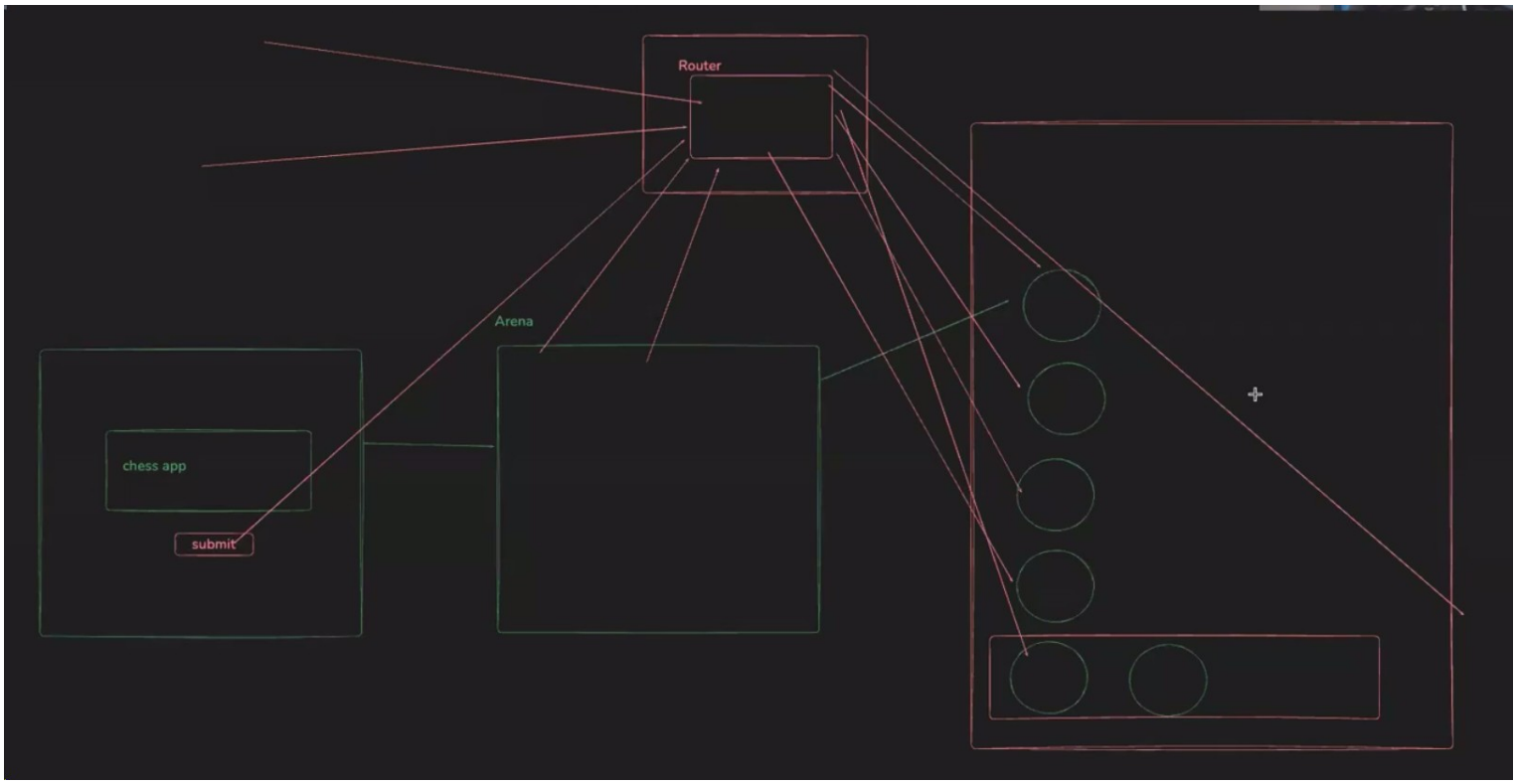
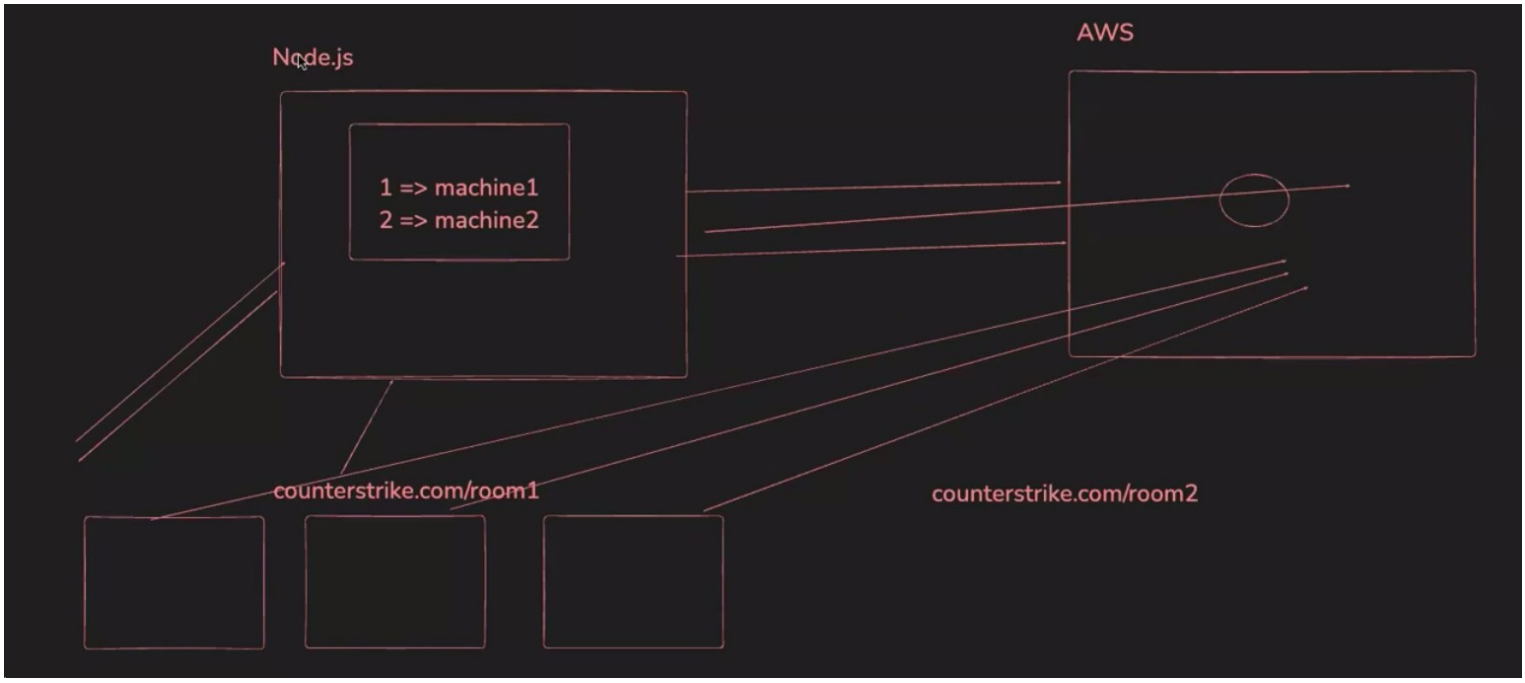
Like in the bolt and online application creator website whenever we write a application like chess app it starts to create the application on a vscode terminal on the side how that is done?
Its like whenever a request is made that pumps up a machine in which a vscode terminal runs
so we are creating a architecture which does this only using asg like based on the user usage is 100 users are hitting the website 100 machines will run if not then It will scale down to less machines based on the usage



now what are we making a system on the aws which will decide the machines based on the users and will also create a warm pool of machines which will be idle to reduce the waiting time for the user
in that way we can do that
in the warm pool of machines the number of machines are decided based on the users you have

the machine can destroy itself if not in use or self destruct that is done by a ROUTER which maintains all these stuffs



this assigns the user the machines and also if the user refresh or internet break it will not immediately remove the machine it will keep the machine or if for a longer period of time the machine is not accessed then it will be removed

this is also used in like youtube rendering, multiplayer games in which there is a router/middleman which assigns work to workers

like in counter strike the players are assigned to a websocket server which is used by the users and then they are connected until the server is not dead

ENDPOINTS
build a blog website with astro
why the name is kept router?
Bcoz it routes the request from browser to server based on the user request

now we are creating the router which can route the request and also can keep some empty machines to be fired based on the usage of the people

to do this we have to create a template in the aws like which will contain the boilerplate code and also the vscode terminal machine on top of that the ai will do its job

KUBERNETES HELPS IN SHARING SAME CPU BASED ON THE USAGE IF SOMEONE IS HEAVY USER GIVE THEM MORE COMPUTER AND IF SOMEONE IS NOT THEN GIVE THEM LESS COMPUTER
to do that first create a new instance
ssh -i pdkey.pem ubuntu@16.171.40.68

btw how a browser runs a vscode?
There is a project called code server on github which does this or use web containers

to run the code we need a vscode which we can get using the docker file from the code server website but the issue is we don't have node js we can get that from the mobile magic harkirat github and use the code server code and work on it

now trying the docker file locally using
cd code-server
docker build -t code-server-update .\

create a vscode base image
t2 medium is required for using the vscode from other places

then ssh using first
ssh ubuntu@ip
ssh -i name.pem ubuntu@ip

vscode in the browser
code-server
to use the project or something we need to need the node js also so for that we can create a docker file
docker build -t code-server-update .

docker run -p 8080:8080 code-server-update

now on localhost:8080 the code will be running

Dockerfile

```
FROM codercom/code-server:4.96.4

USER root
RUN apt-get update \
    && apt-get install -y curl \
    && curl -fsSL https:deb.nodesource.com/setup_22.x | bash - \
    && apt-get install -y nodejs \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

##Revert to the code-server default USER
USER coder
#Expose code-server's default port
EXPOSE 8000

RUN mkdir -p /tmp/bolty-worker

#Start code-server on container launch
CMD ["code-server","--auth","none","--bind-addr","0.0.0.0:8080","/tmp/bolty-worker"]
```

in this the first line imports the file from the github
to install the node we need the root access

then the run file install the node for the vscode
then we change back the access to the user only

the mkdir in this the p does a work it ensure it makes the folder recursively on the based of the url given

and the below command makes the command run and creates a bolty-worker folder on the vscode

now for the aws server I have to create a docker file then build the image and then run the image right

now a question arises like whenever there are like 100k users you will use 100k machines? Why not docker instead?
Good point but the point is if 100k people are using they all are not using for like 5 hrs straight and like only 5k-10k people are using extensively so that is down for that purpose only replit allows only 3 free machines for a id for this reason only

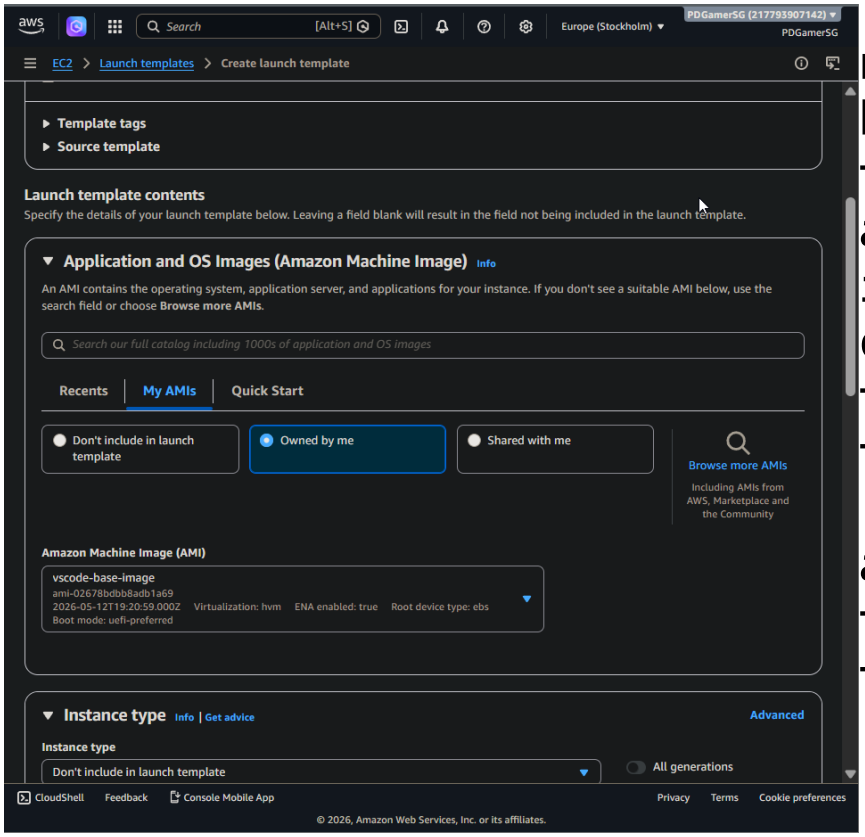
now on the system first install docker for ubuntu using the google commands and then using vi Dockerfile
copy the code and then run this command to build the file
sudo docker build -t code-server-with-node .

Now to use the ip for the vscode I have to expose a security group so go to security tabe and click on the security group and then click on the add rule to add 8080
this is just for testing to add 8080 first for ipv4 and then ipv6
sudo docker run -p 8080:8080 code-server-with-node

we need sudo for this because of the way we have installed the docker that’s why

now use the public dns and with that add :8080
now on that ip the vscode is running crazy its working damn bro
now we have to create asg to create that we have to create a image

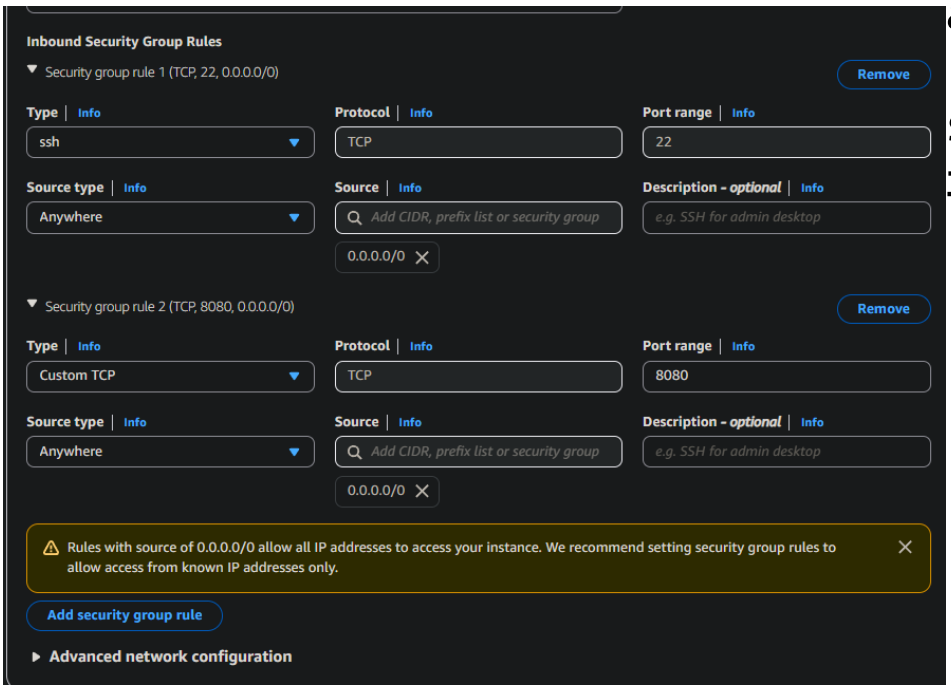
now to create a image select that instance and then select action and then image and then create image
now the image is created and which is called as a **AMI (amazon machine image)**



now to create a asg we have to create a template
and while creating the image select the my amis option and then choose the ami according to that

and then in the instance type choose the image t3.micro (free :))

then take the pair key login file as pdkey.pem
then in the security group create a security group and then add a security group rule



add these 2 rules

ssh and the 8080 rule

and then go to advanced details section and then add data to user data script and its like what commands should the system has to run while creating the system

sudo docker run -p 8080:8080 code-server-with-node
we are adding this but not sure will this run or not

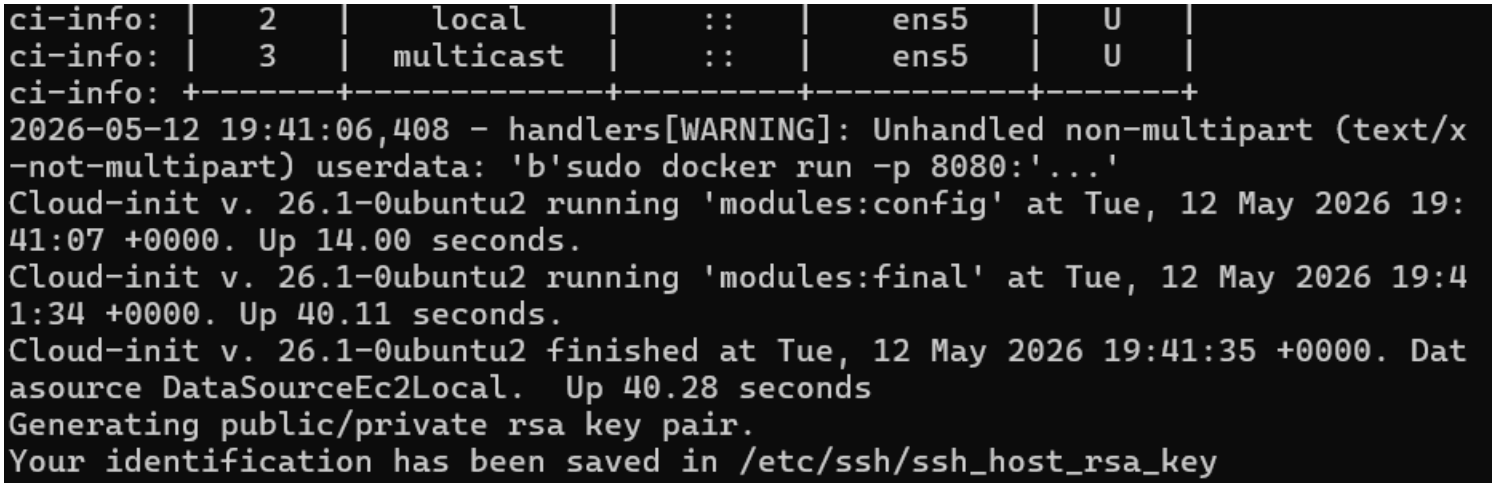
after this we used to create a target group and a load balancer
but we don’t need a load balancer
why?
Bcoz the we are giving direct access to the user the access of the machine so for that it does not require to go through the load balancer
so now we just need to create a **auto scaling group**
this can just start a bunch of these vscode machines

now in that select the template
now with experience there is a thing like
WHENEVER LIKE YOU CHOOSE MULTIPLE ZONES FOR THE WORK IT TRIES TO BALANCE THE LOAD BY CLOSING AND OPENING THE MACHINES IN DIFFERENT ZONES WHICH CREATES A PROBLEM FOR THE USER IF ITS USING THAT MACHINE THAT MACHINE WILL SHUT DOWN IN BETWEEN TO FIX THAT USE JUST ONE AVAILABLE ZONE

use no load balancer not required
and for now keep the machines min max as 1 only
later we will scale this using the router
now for the first time the system it made it will not work now lets find the issue by ssh into the machine

we did ls
dockerfile is present
now we did docker ps
docker file is not running that means either docker not worked or maybe sudo not worked lets fix that

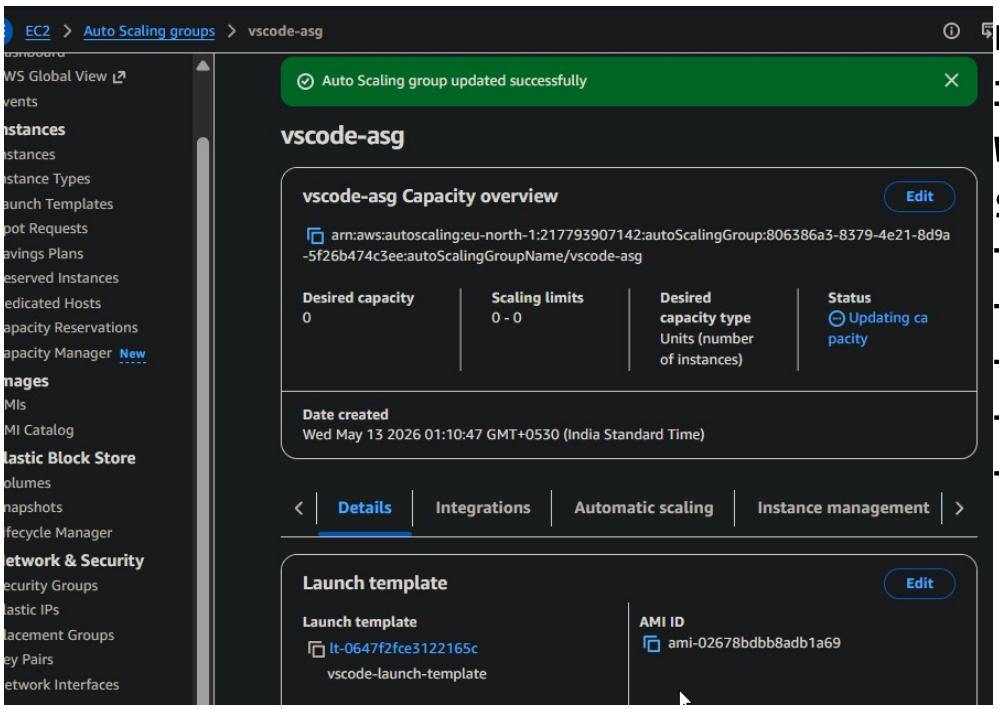
now we need to check the logs for that
so we can google how to find the logs for user data for aws
just do vi /var/log/cloud-init-output.log



#!/bin/bash
this error is caused because we have not specified which shell to use btw #! this is called shabang

too add this to user data go to launch template and

add this data and choose the older ami only and all the pdkey and all will be there only
now to use this change in asg also
change the template version from 1 to 2



now in this top right edit it in which change the systems to 0 0 0 to scale down and then change back to 1 1 1 to get the latest data in the system

now use the public ip and with :8080
and then you can see the vscode is visible

NOW TILL THIS WHAT ALL THINGS WE HAVE DONE IS
1. STARTED EC2 MACHINE WITH CODE SERVER RUNNING
2. CREATED A IMAGE FROM IT
3. CREATED A LAUNCH TEMPLATE FROM IT
4. CREATED A ASG FROM IT

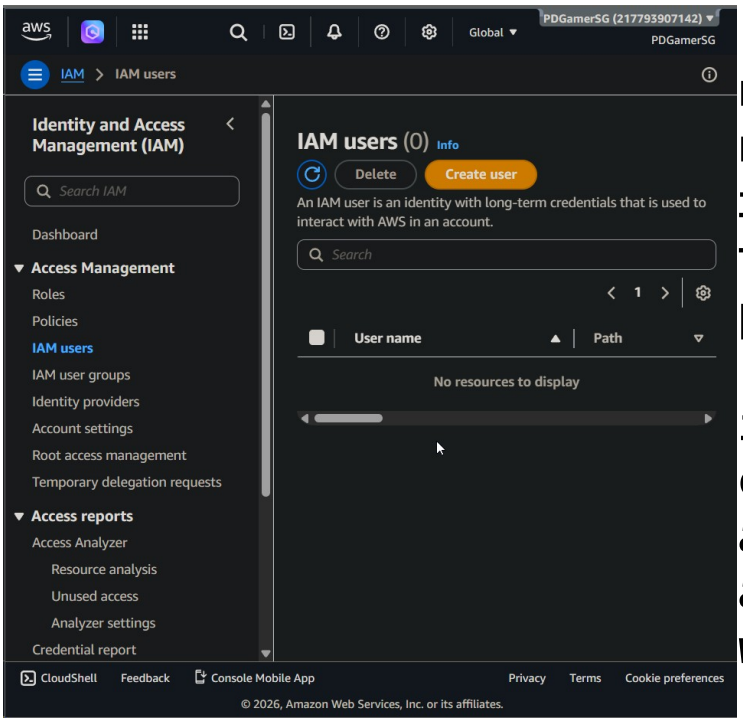
NOW WE HAVE TO JUST WRITE THE ROUTER

NOW WHAT WE HAVE TO DO IS CREATE A ORCHESTRATOR IN WHICH CURRENTLY THERE IS A EMPTY FOLDER RIGHT NOW
IT SHOULD TALK TO ASGS AND THEN SCALE THEM UP/DRAIN THEM DOWN WHEN THE WORKER BECOMES IDLE

now if the user became idle and comes back tmrw to access the data how he will get that?
He will get that back bcoz the data is stored in the s3 machine and while again taking the system the system will take the data from the s3 and show you the all code and the changes you made

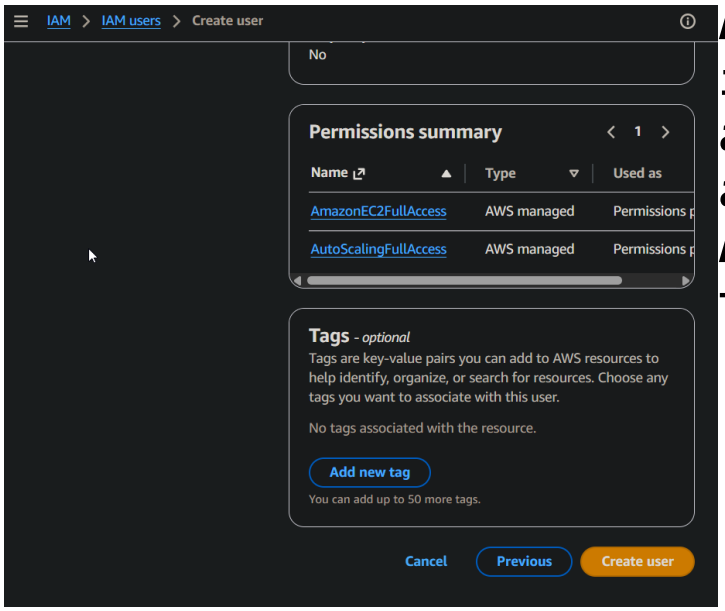
there are various types of shells:
zsh, bash, cmd
that #!/bin/bash specifics run the code on the bash shell only

now we will create a router which will connect to the aws using a api key called in aws as
IAM
which will allow the router code to access the asg group
in that we have to create a user



now create the user
now in that give name
router-user
then tick the attach policies directly

in this we can give like certain dept certain thing access based on the user and the dept and right now we are giving like



AutoScalingFullAccess
its like you can give autoscaling group full access
AmazonEC2FullAccess
this access also

now in the user security creds create a access key
then use the connect using cli
no need to create a tag and just create you will get a user id and pass
now save the access key and secret in the env file in this variable

AWS_ACCESS_KEY
AWS_ACCESS_SECRET
now to connect to the aws asg
we have a npm package for that
@aws-sdk/client-auto-scaling

now initialize a project with bun in the folder

bun init -y
bun add @aws-sdk/client-auto-scaling
now I want to change the number of machines from the code file which is called as router for now

now read the docs first import and in the client just write the name of the location from where the system is made which is written on the top of the aws website eg eu-north-1 and also the secret key

then we can use functions like describeautoscalinginstances, describeloadbalancer, describescalingactiviites and many more

```ts
import { AutoScalingClient,SetDesiredCapacityCommand } from "@aws-sdk/client-auto-scaling";
const client = new AutoScalingClient({ region: "eu-north-1", credentials:{
    accessKeyId: process.env.AWS_ACCESS_KEY!,
    secretAccessKey: process.env.AWS_ACCESS_SECRET!,
} });

const command = new SetDesiredCapacityCommand({
    AutoScalingGroupName:"vscode-asg",
    DesiredCapacity:3
})
const data = await client.send(command);

console.Log(data);

```

now in this using the docs click on the function you want to use and then the input option to check what all inputs it can take and then just use the client.send to use the function

then run the file using bun index.ts

```
> worker-orchestrator git(main) bun .\index2.ts
{
  $metadata: {
    httpStatusCode: 200,
    requestId: "448ac922-f443-4d46-950c-e1afe805aaff",
    extendedRequestId: undefined,
    cfId: undefined,
    attempts: 1,
    totalRetryDelay: 0,
  },
}
> worker-orchestrator git(main)
```

this means it worked and changed the capacity to 3 based on the code

and the the max machines is 10 we cannot input above 10

now lets convert this into a express server

bun add express

bun add -d @types/express

now what we have to do is we have a project id now with that project it we have to find that machine or an idle
now we have to add some logic
now we have to use the cs brain for this for now many libraries helped us

promise means you have to async and await

ls -al to see all the hidden files in the folder like .env also as its not seen in the ls command

celenium script
this is like all the first principle approaches before the kubernetes just try to understand and just learn what all things are present in this

as in this the @aws-sdk/client-auto-scaling this library was not having the ip of the system so for that we required to use the another package called

@aws-sdk/client-ec2
this helps us to get the ip of the system

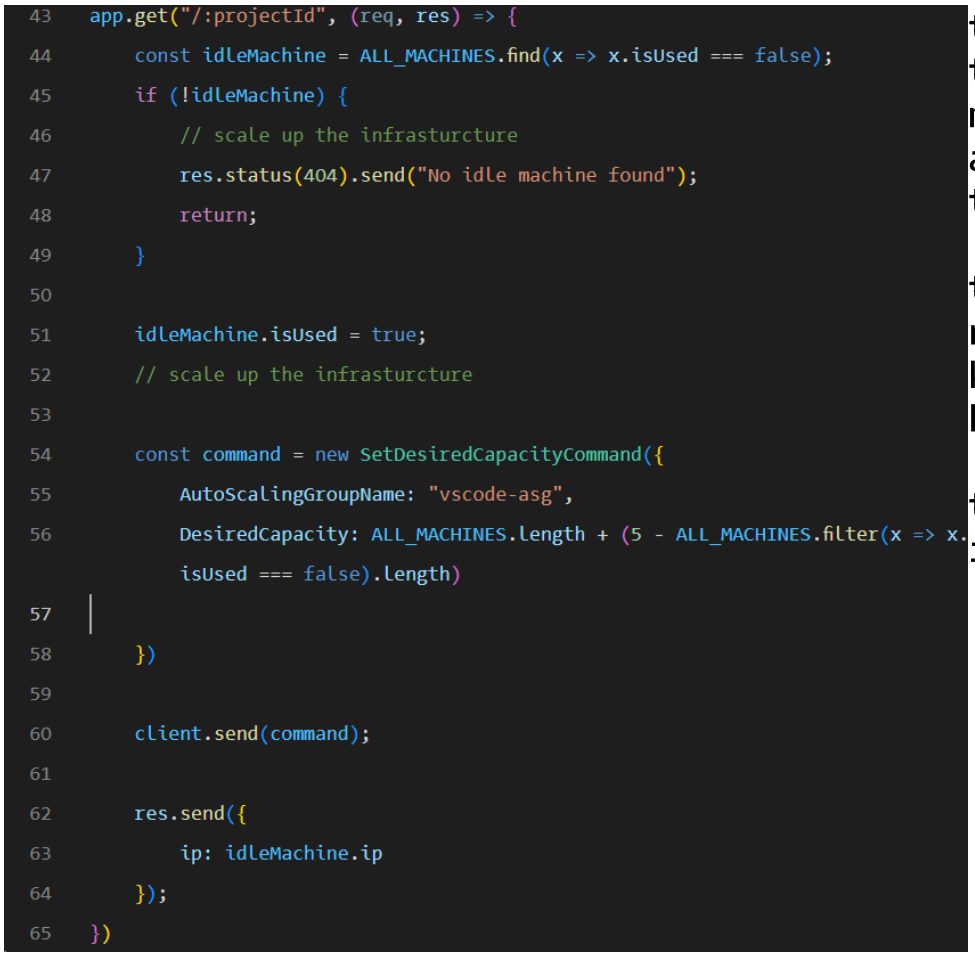
Code explanation

```ts
import express from "express";
import { AutoScalingClient, SetDesiredCapacityCommand,
DescribeAutoScalingInstancesCommand,
TerminateInstanceInAutoScalingGroupCommand } from "@aws-sdk/
client-auto-scaling";
const { EC2Client, DescribeInstancesCommand } = require("@aws-sdk/client-ec2");

const app = express();
const client = new AutoScalingClient({ region: "ap-south-1", credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY!,
    secretAccessKey: process.env.AWS_ACCESS_SECRET!,
} });

const ec2Client = new EC2Client({ region: "ap-south-1", credentials: {
    accessKeyId: process.env.AWS_ACCESS_KEY!,
    secretAccessKey: process.env.AWS_ACCESS_SECRET!,
}})

type Machine = {
    ip: string;
    isUsed: boolean;
    assignedProject?: string;
}

```

in this nothing was there just defining the things and creating a machine struct type which will have all the things for a machine

```ts
async function refershInstances() {
    const command = new DescribeAutoScalingInstancesCommand();
    const data = await client.send(command);

    const ec2InstanceCommand = new DescribeInstancesCommand({
        InstanceIds: data.AutoScalingInstances?.map(x => x.InstanceId)
    })

    const ec2Response = await ec2Client.send(ec2InstanceCommand);
    // console.log(JSON.stringify(ec2Response.Reservations[0].Instances[0].
    PublicDnsName))
    // TODO: Enrich the ALL_MACHINES array with the new instances, and remove
    the instances that have died
}

refershInstances();

setInterval(() => {
    refershInstances();
}, 10 * 1000);
```

in this we are trying to get the status of all the systems and then try to ping them every 10 seconds to get the status of the system if its available or being used



```
app.get("/:projectId", (req, res) => {
    const idleMachine = ALL_MACHINES.find(x => x.isUsed === false);
    if (!idleMachine) {
        // scale up the infrasturcture
        res.status(404).send("No idle machine found");
        return;
    }

    idleMachine.isUsed = true;
    // scale up the infrasturcture

    const command = new SetDesiredCapacityCommand({
        AutoScalingGroupName: "vscode-asg",
        DesiredCapacity: ALL_MACHINES.length + (5 - ALL_MACHINES.filter(x => x.
        isUsed === false).length)

    })

    client.send(command);

    res.send({
        ip: idleMachine.ip
    });
})
```

this helps to identify the machines based on the machines available and we add and remove based on the number of machines

this also require the asg name and kuch toh khatarnak sa logic likha hai Idk what

then just we return the ip

```
app.post("/destroy", (req, res) => {
    const machineId: string = req.body.machineId;

    const command = new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: machineId,
        ShouldDecrementDesiredCapacity: true
    })

    client.send(command);
})

app.listen(9092);
```

this is just used to terminate the session

IN THIS LESSON WE LEARNT HOW CAN WE ACCESS THE AGS USING THE TERMINAL AND THEN INCREASE OR DECREASE THE LOAD BASED ON THE USERS AND WORKERS