This is like amazon answer to kubernetes
if you want similar benefits like kubernetes in aws use this
ECS = Elastic container service(Container orchestrator)
kubernetes is also a container orchestrator
ECS is more easier that kubernetes
ECS downside is it can only use the aws cloud but the kubernets can use any you can easily move from one to another
Containerized application means == a application that is dockerized, for something you have created a image for which you can push on dockerhub and place and run it
serverless env == you cannot see the servers(lamda,cloudfare servers)
ECR = elastic container registry(aws, google they all have there own registry)
Today, we'll learn how to deploy containerized applications to a serverless environment on AWS using ECS.

We'll also learn about ECR, AWS's container registry (similar to dockerhub)

What is ECR

ECR stands for **Elastic Container Registry**. It is a fully managed Docker container registry service provided by Amazon Web Services (AWS). ECR allows you to store, manage, and deploy Docker container images, making it easier for developers to work with containerized applications.

Key features of ECR include:

1. **Private repositories**: You can create private repositories to store your container images securely.
2. **Integration with ECS and EKS**: ECR is fully integrated with Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS), making it seamless to deploy containers from ECR to these services.
3. **Scalability**: As a managed service, ECR scales automatically to accommodate the growth of your container image repository.

What is ECS

elastic container service
in this the elastic means it can scale up and scale down based on the usage

What is ECS

ECS stands for **Elastic Container Service**. It is a fully managed container orchestration service provided by Amazon Web Services (AWS) that allows you to easily run, manage, and scale Docker containers on AWS infrastructure.

THIS IS SIMILAR TO THE ASG IN THAT ALSO CREATE A MACHINE WITH THE TEMPLATE AND THEN ADD THE DOCKER IMAGE IN THAT AND IT WILL ALSO RUN THAT AND BASED ON THE LOAD IT WILL SCALE UP AND DOWN
TO REMOVE THE HASLE OF MAINTAINING THE MACHINES YOU CAN USE THE KUBERNETES OR THIS ECS
Key features and concepts of ECS include:

1. **Container Orchestration**: ECS automates the deployment, scaling, and management of containerized applications. It allows you to run Docker containers without managing the underlying servers or infrastructure manually.
2. **Cluster Management**: ECS organizes resources into clusters, which are groups of EC2 instances (virtual servers) or AWS Fargate instances that run your containers. ECS automatically handles scheduling and managing the resources within these clusters.
3. **Task Definitions**: In ECS, you define your containerized applications in "task definitions." A task definition is a blueprint that describes which Docker containers to run, their settings (e.g., CPU, memory), and how they interact with other services.
4. **Service Management**: ECS allows you to define services that ensure your containers run continuously, are scaled according to demand, and are replaced automatically if they fail.

5. **Integration with Other AWS Services**: ECS integrates with various AWS services like Elastic Load Balancer (ELB), CloudWatch for monitoring, IAM for security, and ECR for container image storage, enabling a seamless environment for deploying containerized applications.
6. **Fargate**: ECS can run containers on EC2 instances, but it also supports **AWS Fargate**, a serverless compute engine that removes the need to manage the underlying infrastructure. With Fargate, you only specify the CPU and memory requirements, and AWS handles the rest.
7. **Scaling**: ECS can scale applications automatically based on demand, ensuring that your containers are appropriately distributed across resources.

IN THIS THE FARGATE WE WILL USE TO SCALE UP AND SCALE DOWN THE MACHINES IN THE AWS

Containerize a Node.js app

- Initialize an empty bun app

```
bun init
```

- Add express to it

```
bun add express @types/express
```

- Write a basic node.js app

```
import express from "express";

const app = express();

app.get("/cpu", (req, res) => {
        for (let i = 0; i < 100000000; i++) {
                Math.random();
        }
        res.send("Hello world");
});

app.listen(3000)
```

- Create the dockerfile

```
FROM oven/bun:alpine

WORKDIR /app
```

```
COPY . .

RUN bun install

CMD ["bun", "run", "index.ts"]
```

- Build the dockerfile

  ```
  docker build -t node-app .
  // mac M1 people should add --platform=linux/amd64
  ```

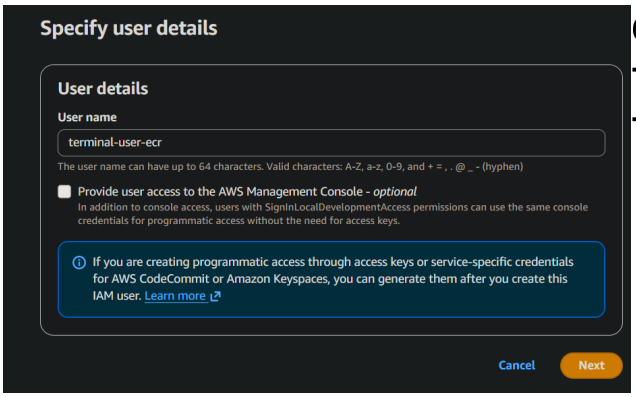
- Try running it locally

  ```
  docker run -p 3000:3000 node-app
  ```

now to send the data to dockerhub we have to do is
docker login
docker push 100xdevs/node-app

now login to aws and search for ecr
elastic container registry
create a repo
keep a name pd/nodejs-app
mutable, immutable you can change the image
we can keep a tag also for the image
now we created a repo now the challenge is how we will push to aws repo
now to push to aws repo we have to install the aws cli and then by clicking on the name of the repo click on the view push commands and use the commands
now type aws in the terminal to check if installed or not

now we have to put the creds of aws but its strongly suggested make a new id pass using the IAM
create a new user for the terminal and give the permissions as required



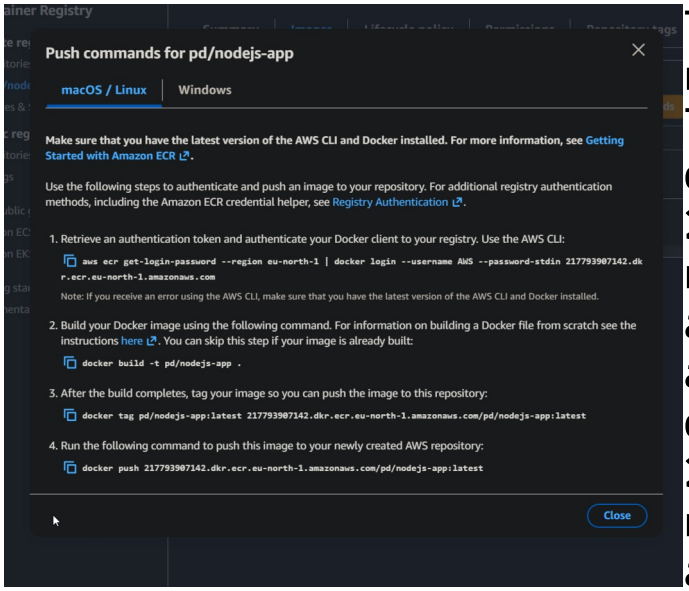
don’t check the below option as the terminal will never interact with the browser

this is like giving access like what all a devloper can access, what all a devops enginner can access like that
give this access for this
AmazonEC2ContainerRegistryFullAccess
now to get the user id and pass click on the name and go to security tab and create access key
select CLI
this we have created a SHADOW USER
now in the terminal use the aws configure and paste the user id and password in that

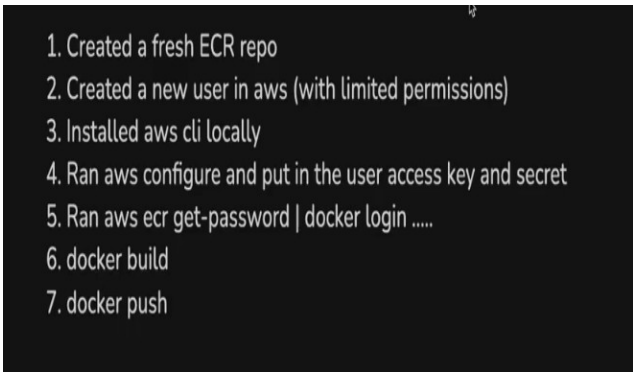
now the aws configure is done now do the view push commands which were present
use the macos/linux command only not the windows one as I have installed the aws cli

```
aws ecr get-login-password --region eu-north-1 | docker login --username AWS --password-stdin 217793907142.dkr.ecr.eu-north-1.amazonaws.com
```

docker images to see all the images
now what we are doing is we have to change the name of the docker file according to the repo name so we are tagging that
```
docker tag pd/nodejs-app:latest 217793907142.dkr.ecr.eu-north-1.amazonaws.com/pd/nodejs-app:latest
```
or what we can do is just(mention the linux speciafically)
```
docker buildx build --platform linux/amd64 -t pd/nodejs-app .
```



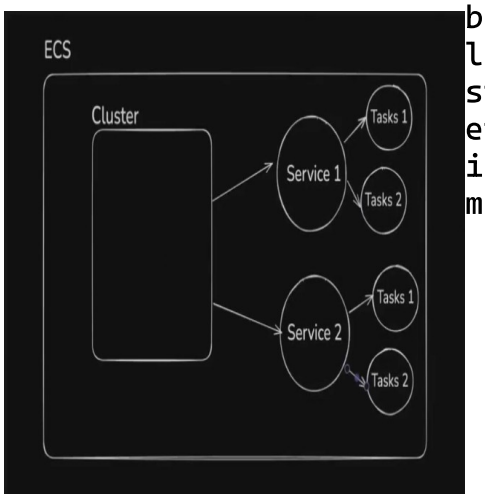
This commands works perfectly now for some changes just change latest to v2
docker tag pd/nodejs-app:latest 217793907142.dkr.ecr.eu-north-1.amazonaws.com/pd/nodejs-app:v2
and then
docker push 217793907142.dkr.ecr.eu-north-1.amazonaws.com/pd/nodejs-app:v2



HOW THE USERS ARE DIVIDED?
For backend we give perms like
ec2, ecr ,ecs
For Frontend
s3, cloudfront

NOW THE IMAGE HAS REACHED THE AWS NOW FROM THERE WHERE TO TAKE THE IMAGE DO YOU TAKE IT TO AWS EC2 MACHINE, KUBERNETES CLUSTER, ECS??

GOAL OF ECS ARCHITECTURE



basic understanding
like in this the cluster is like the head which can start a ws server, a backend server, a frontend server etc
in this the server is like a asg which can start multiple machines which is considered as tasks

Cluster

A **Cluster** is a logical grouping of resources used by ECS to manage and run containerized applications. A cluster is essentially a pool of computing resources (e.g., EC2 instances or Fargate) where your tasks (containers) run.

- **EC2 Cluster**: When you use EC2 launch type, the cluster contains EC2 instances that are registered to ECS.
- **Fargate Cluster**: When you use Fargate, the cluster is virtualized, and ECS manages the infrastructure for you (you don't manage the underlying EC2 instances).

Task

A **Task** is a running instance of a containerized application in ECS. It's the basic unit of work in ECS. A task is defined by a **Task Definition**, which specifies the Docker image to use, resource requirements (like CPU and memory), networking configurations, environment variables, and more.

- **Task Definition (similar to launch templates of aws)**: The blueprint for your task, describing which containers to run, their configuration, and the resources they need.
- **Running Task**: Once ECS launches a task from a task definition, it becomes a running task, executing your containerized application.

Service

A **Service** is a higher-level abstraction on top of tasks in ECS. A service allows you to maintain and scale a specified number of task instances running and ensures that the desired number of tasks are continuously running.

- **Scaling**: Services allow automatic scaling of tasks up or down based on demand (e.g., adding more task instances when traffic increases).
- **Load Balancing**: ECS services can also be associated with a **load balancer** (like an ELB) to distribute incoming traffic across running tasks.

MY LEARNINGS

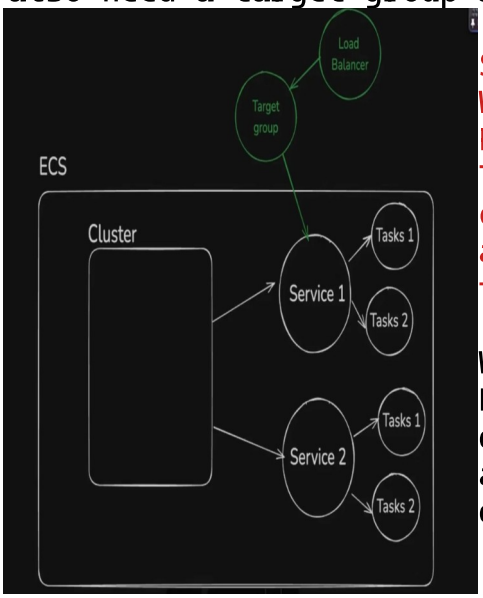
WHAT IS FARGATE CLUSTER?
Serverless containers and deploy container without managing the machines you want someone else do that for me
it should automatically do scale up and scale down of machines

Task Definition (similar to launch templates of aws)

CLUSTER KO YOU GIVE A SERVICE
SERVICE KO YOU GIVE TASK OR TASK DEFINITION
THEN THE SERVICE ACTUALLY RUNS
THE THING ACTUALLY RUNNING IS CALLED A TASK
AND THE BLUEPRINT IS CALLED AS TASK DEFINITION

WHAT DOES SERVICES DO?
Creates task

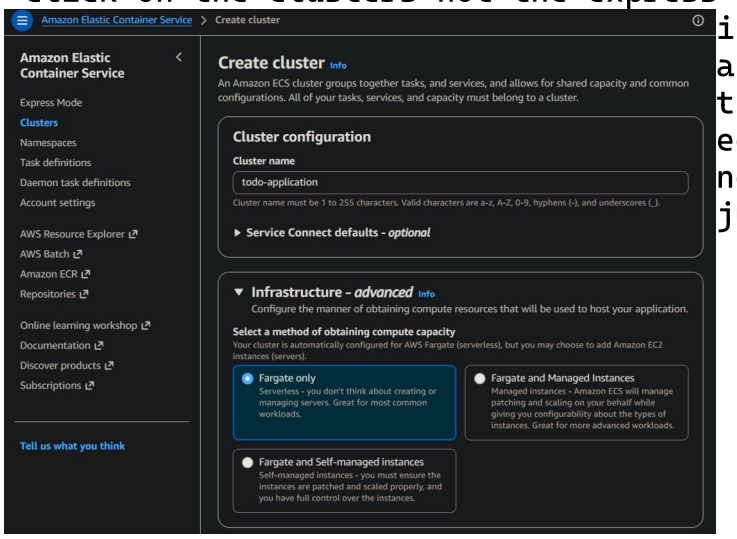
now what we will do?
First we will create a clusters called node js application then we will create a task definition and then we will create a service attached to the server then we also need a target group and load balancer attached to the services



STUPID QUESTION
WHY NOT JUST GIT CLONE RATHER THAN MAKING A DOCKER FILE?
The thing is git only gives you the code back not the other things but the docker file gives you the code but also the node js, docker install etc which is required to run that git repo files so yaa

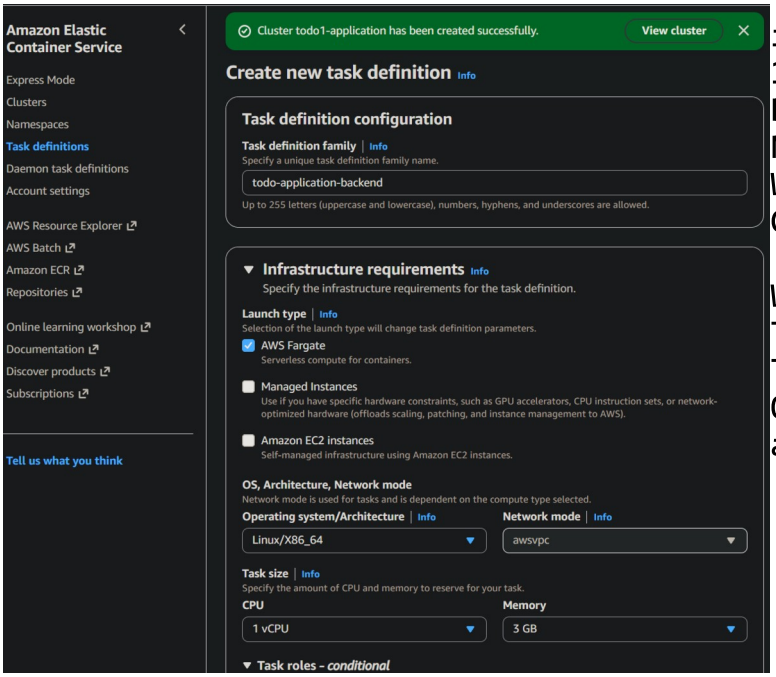
WHY WE ARE USING ECS WHEN WE HAVE ASG IN AWS?
Because we don't have serveless autocalling that just do the autoscalling part
asg can deploy anything but the ecr does only the dockerfile only

till now we have pushed our code to ecr and now we have to connect that to ecs elastic container service
click on the clusters not the express mode



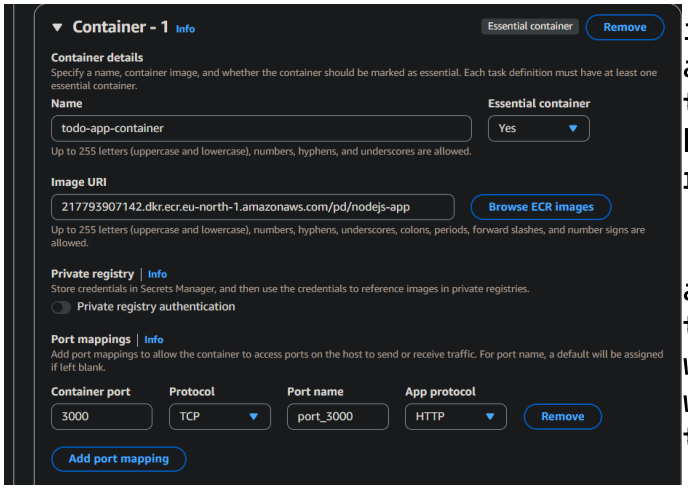
in this we can also do like both fargate and ec2 machines for the infra for like 30% time use the fargate and 70% time use the ec2 machines like that we can do but for now we are just using the fargate only
just add the name and fargate

Now create a new task definition



in this we should choose how many cpus 16 WILL THAT GIVES PERFORMANCE BENEFITES?
NO MAN NODE JS IS SINGLE THREADED AND WE HAVE NOT WRITTEN ANY MUTLITHREADED CODE

WHAT ALL THINGS WE CHOOSE?
Task name
Task launch type
OS name
and now the container type



in this give a name
and the code was export on the port 3000 so this port and give that a name also and also below you can add some env variables for redis, mongo etc

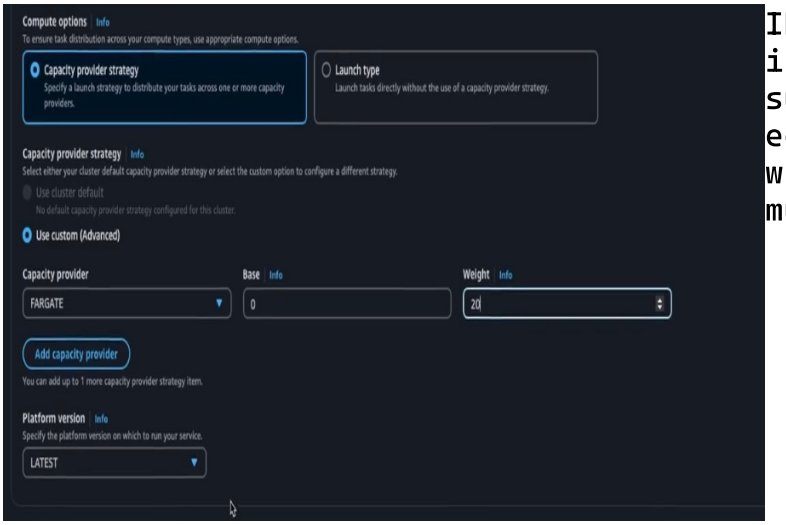
also get the url from the ecr homepage for the image we create paste that here and also we can use docker image also but this will will be more fast as compare to docker for the ecs

now we created a cluster its empty right now
LAST STEP (CREATING A SERVICE IN A CLUSTER)
we have not created a task we have just created a blueprint for the task
now we have to create a service inside the cluster and attaching the task to the service and giving the service a desired capacity such as 2 machines

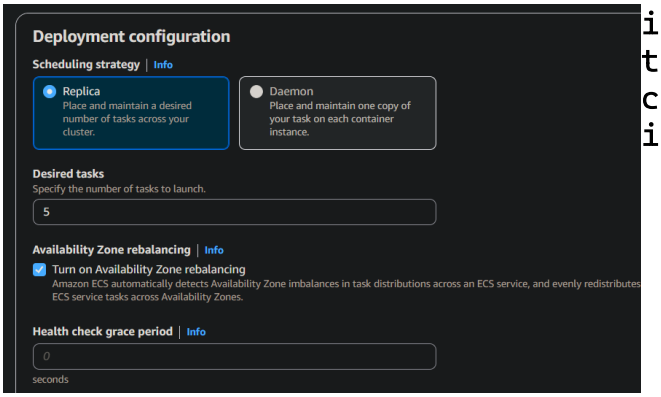
now in the clusters we will have a service section through which create a service which will create a task

WHAT IS FARGATE_SPOT?, SPOT INSTANCES
ITS A MACHINE WHICH SOMEONE WAS RUNNING BUT NO ONE IS NOT USING THAT CURRENTLY SO THE AWS USES THAT TEMPORALLY IF REQ
THE THING IS THEY ARE CHEAP BUT THEY CAN SHUT DOWN ANY TIME SO DONT USE THAT MUCH

WHILE CREATING THE SERVICE FIRST CHOOSE THE TASK DEFINITION THEN CHOOSE THIS IN THE PROVIDER STRATERGY



in this we can use multiple providers such as fargate, fargate_spot, ec2, ec2_spot and we can provide weight also which thing should be assigned to how much amount of machines



in this we have two things we can run a single task only or we can also run a service which can run multiple tasks which we will be using in this we have fixed that only 5 machines

AND AS OUR SYSTEM IS INTERNET FACING WE NEED A LOAD BALANCER
and for a load balancer you always need a target group so creating that also
because the load balancer only understand the target group that it should send 70% of data to this traffic group
below this only created the target group and gave the name

NOW WHAT WAS THE PROCESS
1. IN THIS CLUSTER WE WANT TO SEND ALL THE REQUEST TO FARGATE
2. WILL RUN THE TASK DEFINTION WE MADE EARLIER
3. FIXED TASKS WHICH WILL BE RUNNING
4. CREATING THE LOAD BALANCER AND THEN THE PORT ALSO MENTIONED WITH THE NAME , DEFAULT PORT OF 80 AND A TARGET GROUP BELOW ALSO

NOW WHAT WE HAVE A CLUSTER WHICH HAS ONE SERVICE WHICH IS ATTACHED TO A LOAD BALANCER THROUGH A TARGET GROUP AND THAT ONE SERVICE WILL START 5 TASKS OF THE TASK DEFINITION WE CREATED IN THE STARTING

THIS IS THE PROCESS OF
THIS IS HOW WE DEPLOY A CONTAINERIZED APPLICATION TO ECS

haam kabhi bhi load balancer ko machines nhi dete hai machines rehti hai sirf target group kae paas and load balancer command deta hai target group ko and target group fhir machines ko use karta hai based on the conditions

load balancer bss yeh bolta hai send 30% traffic to this traffic group and send 70% to this traffic group etc

if you want to check any issues in the code just run check the logs in the service section

HTTP GENERATION
TO ADD HTTPS TO THE WEBSITE JUST ADD A LISTENER IN THE LOAD BALANCER FOR THE HTTPS AND CHANGE SOME SETTINGS
need a certificate for the machine also using the acm
add the cname and all also and which should point to the load balancer

also for the load balancer add port 80 so that it should available to the world

run the website using the loadbalancer dns name
in the dns add a record for the website in the namecheap, square space etc

todo-app-loadbalancer-1962923628.eu-north-1.elb.amazonaws.com
just add this much the loadbalancer url not the https and the / part
and add the record and wait like 5 min

hosting you loadbalancer on your domain on http is easy but for https it gets tricky
now to do the https part just go to the certificate manager
THESE ARE THE STEPS
1. take the load balancer url and then put it in the namecheap with a cname record and then make it work wait 5 min
2. then go to the certificate manager and then add the load balancer url and make the certificate manager to the last step where you will again get a url and a value which again put in the namecheap
3. then go to load balancer and add a listener in which choose https at the top, add ur url of the website and then below the certificate from acm the previously generated certificate will be shown choose that and you will get your https
(BTW THE VALIDATION REQUIRES TIME TO APPROVE)

WE CAN USE THE CERTBOT ALSO THE NGINX AND CERTIFICATE IN THE AWS

NOW DOING THE HTTPS USING THE CERTBOT

create a ec2 machine
why not locally?
We need a public ip machine to make the certbot work
use the docs
then after installing do
sudo vi /etc/nginx/nginx.conf
BTW FOR EACH URL WE NEED A CERTIFICATE AND IF YOU WANT TO GET FOR ALL AT ONCE USING THE WILDCARD CERTIFICATE
to clean the whole file use :%d

```
events{
     #Event directives...
}
http{
server {
 listen 80;
 server_name todo-app-class.pallabdas.me;

 location / {
 proxy_pass http://127.0.0.1:8000;

 # Recommended headers for passing client info to backend
 proxy_set_header Host $host;
 proxy_set_header X-Real-IP $remote_addr;
 proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
 proxy_set_header X-Forwarded-Proto $scheme;
 }
}

}
```

add this and then run

```
sudo nginx -s reload
```

certbot helps in creating the certificate which we can later import in the aws
now install certbot

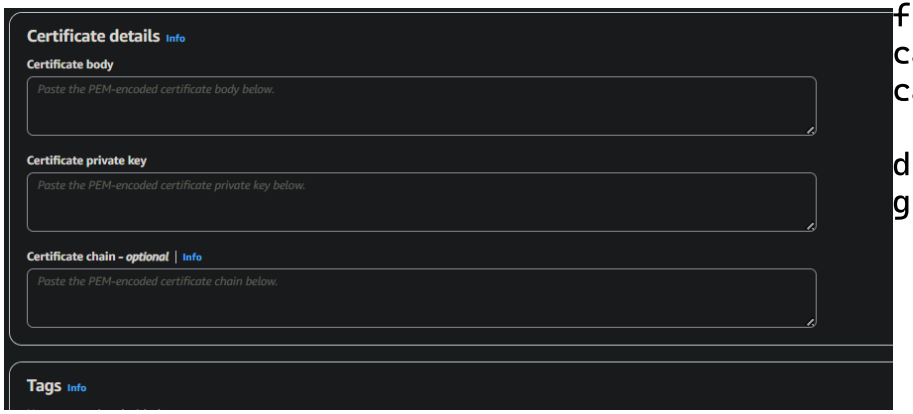
```
sudo snap install --classic certbot
```

set some yes and then add the
a name record for the machine with its public ip and website starting part and also make sure the port 80 is open in the security group add that port 80 and then check on the public url of the machine nginx is showing or not
http://todo-app-class.pallabdas.me/
and the url should also show that the nginx error
after doing all that then press 1 in the certbot

after doing that the location will be mentioned above for the files
first do sudo -s (for the root access)
using ls /etc/letsencrypt/live/todo-app-class.pallabdas.me/
check the files
we can get paid or free both certificates for the website

then just go to the location cd /etc/letsencrypt/live/todo-app-class.pallabdas.me/

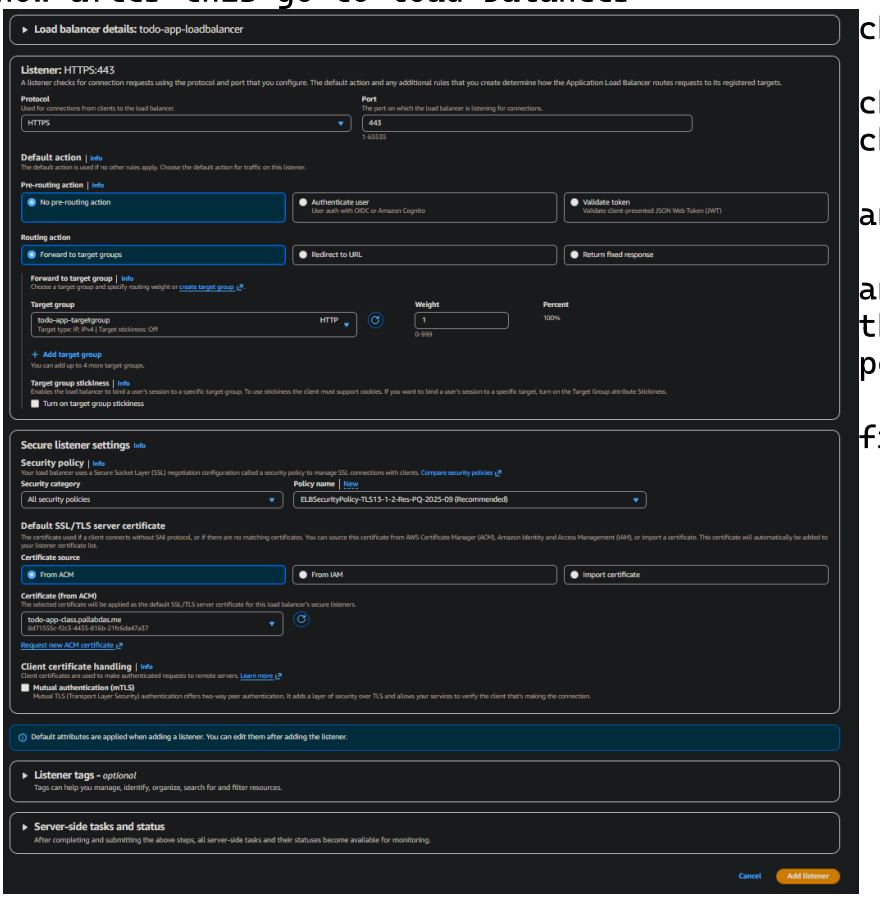
and then just get the keys data



for the first one do cat cert.pem
cat privkey.pem
cat fullchain.pem

do this and paste and you will get the certificates

now after this go to load balancer



change to https

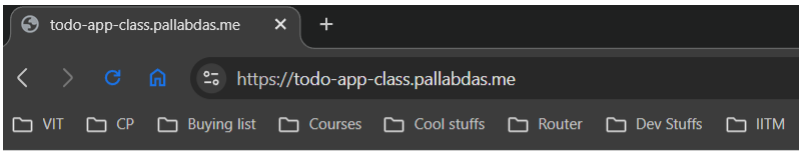
change the target group
check the certificate

and then add the listener

and also in the security group of the load balancer open the 443 port

from 0.0.0.0/0

LAST STEP IS TO TAKE THE DOMAIN NAME AND GO BACK TO NAMECHEAP
CHANGE THE A RECORD WE MADE EARLIER THAT WE MADE TO VERIFY THE NGNINX THING
CHANGE TO AGAIN CNAME WITH THE LOAD BALANCER URL
and delete all other now run the website on icognito and congo your website runs on https


todo-app-class.pallabdas.me
https://todo-app-class.pallabdas.me
VIT
CP
Buying list
Courses
Cool stuffs
Router
Dev Stuffs
IITM

hello wsup