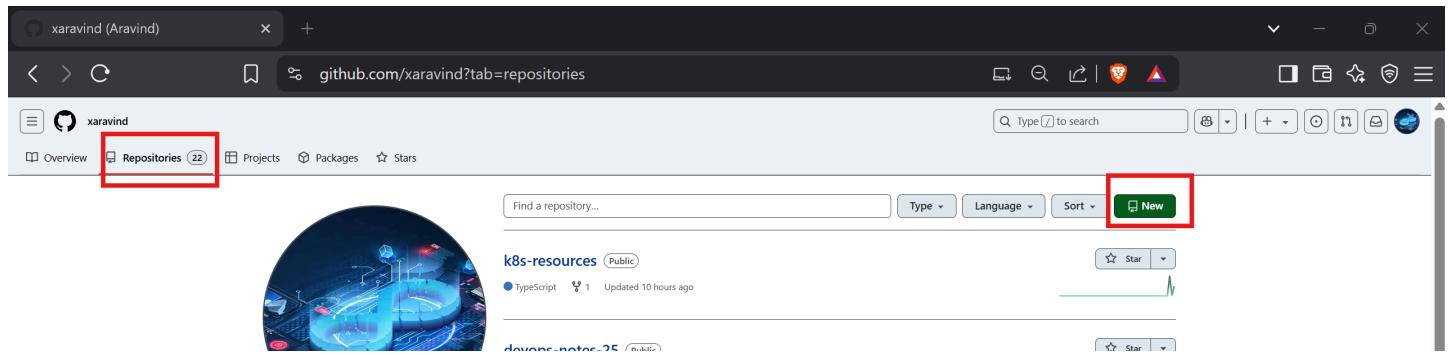


Full Project Setup Guide (GitHub → Docker → EC2 → EKS)

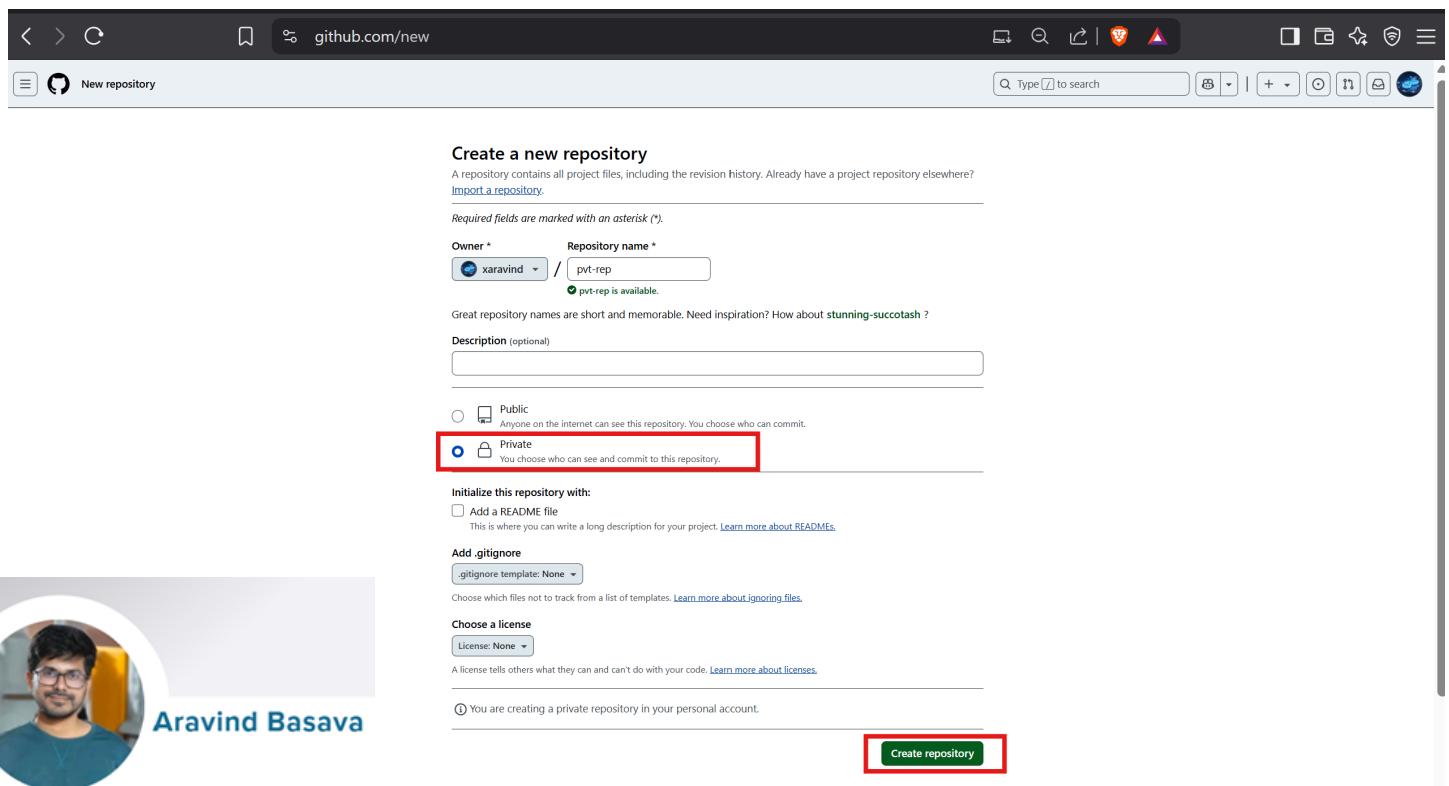
This guide walks through a real-world Kubernetes project from scratch — covering GitHub, Docker, EC2, private registry usage, and deployment on EKS using Kubernetes manifests.

✓ First Stage: GitHub Setup

1. Login to your GitHub account.



2. Create a private repository.



3. Go to the top right of your profile → Settings → Developer Settings → Personal Access Tokens (classic) .

- Click **Generate new token (classic)**.
- Select expiry, **grant only repo access**, and generate the token.
- **⚠ Note:** Never share your credentials or tokens with anyone.

The screenshot shows the GitHub interface for the user 'xaravind'. On the left, there's a sidebar with links like Overview, Repositories (22), Projects, Packages, and Stars. The main area displays three public repositories: 'k8s-resources' (TypeScript, updated 10 hours ago), 'devops-notes-25' (updated last week), and 'xaravind' (updated last week). To the right, there's a sidebar for 'xaravind' with sections for Focusing, Your profile, Your repositories, and other GitHub features. A red box highlights the 'Settings' link at the bottom of this sidebar.

The screenshot shows the GitHub profile settings page ('github.com/settings/profile'). It includes sections for Password and authentication, Code, planning, and automation, Security, Integrations, and Archives. A red box highlights the 'Developer settings' link at the bottom of the page.

The screenshot shows the GitHub developer settings page ('github.com/settings/tokens'). It lists GitHub Apps, OAuth Apps, Personal access tokens (highlighted with a red box), and Tokens (classic). Under Personal access tokens, a new token is being generated, with the 'Generate new token' button highlighted with a red box. A tooltip for the button says: 'Generate new token (classic) Fine-grained, repo-scoped'.

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note: k8s-project

What's this token for?

Expiration: 7 days (Jun 22, 2025)

The token will expire on the selected date.

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories Access commit status Access deployment status Access public repositories Access repository invitations Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> project	Full control of projects Read access of projects
<input type="checkbox"/> admin:gpg_key	Full control of public user GPG keys Write public user GPG keys Read public user GPG keys
<input type="checkbox"/> admin:ssh_signing_key	Full control of public user SSH signing keys Write public user SSH signing keys Read public user SSH signing keys

Generate token Cancel

4. Copy the token and save it somewhere safe.

Personal access tokens (classic)

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

Personal access tokens (classic)

Generate new token

Tokens you have generated that can be used to access the [GitHub API](#).

⚠ Make sure to copy your personal access token now. You won't be able to see it again!

ghp_iJzh3X83Dq...

Delete

5. Go to your private repository and copy the Git URL.

The screenshot shows the GitHub repository page for 'pvt-repo'. The 'Code' dropdown menu is open, highlighting the 'Clone' section. A red box highlights the 'Clone' button and the URL 'https://github.com/xaravind/pvt-repo.git'. Other options like 'Local', 'Codespaces', 'HTTPS', 'SSH', and 'GitHub CLI' are also visible.

6. Open your VS Code or local terminal and clone the repo using the token.

The screenshot shows the VS Code interface. A GitHub sign-in dialog is open, with the 'Token' tab selected. A red box highlights the 'Sign in' button. In the terminal window, a command is being run to clone the repository:

```
aravi@Aravind MINGW64 ~/11_04
$ git clone https://github.com/xaravind/pvt-repo.git
Cloning into 'pvt-repo'...
```

```
```bash
aravi@Aravind MINGW64 ~/11_04
$ git clone https://github.com/xaravind/pvt-repo.git
Cloning into 'pvt-repo'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 3 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
```

```

7. Copy your project code into the cloned Git folder.

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a project structure under "11.04". A red box highlights the "js_code" folder within the "pvt-repo" directory.
- Terminal:** Displays a command-line session:

```
aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)
$ ls
js_code/ README.md test

aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)
$ ls js_code/
dist/ Dockerfile package.json package-lock.json readme.md src/ tsconfig.json webpack.config.js

aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)
$
```
- Bottom Terminal:** Shows the command being run:

```
aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)
$ git add . ; git commit -m "11" ; git push
```

8. Write a Dockerfile in your project, save it, and push the code.

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the same project structure as the previous screenshot, with the "Dockerfile" file highlighted in the "js_code" folder.
- Terminal:** Displays the Dockerfile content:

```
1 # Stage 1: Build the frontend using Node
2 FROM node:lts-alpine3.20 AS build
3
4 WORKDIR /app
5
6 COPY package*.json ./
7 RUN npm install
8
9 COPY . .
10
11 RUN npm run build
12
13 FROM nginx:alpine
14
15 # Copy built app to nginx html directory
16 COPY --from=build /app/dist/ /usr/share/nginx/html
17
18 EXPOSE 80
19
20 CMD ["nginx", "-g", "daemon off;"]
```
- Bottom Terminal:** Shows the command being run:

```
aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)
$ git add . ; git commit -m "11" ; git push
```

9. Open GitHub in your browser and check the files. You should see everything updated.

A screenshot of a GitHub repository page for 'pvt-repo/js_code'. The repository has 12 commits and 315 bytes of code. The Dockerfile contains the following code:

```
1 # Stage 1: Build the frontend using Node
2 FROM node:lts-alpine3.20 AS build
3
4 WORKDIR /app
5
6 COPY package*.json ./
7 RUN npm install
8
9 COPY . .
10
11 RUN npm run build
12
13 FROM nginx:alpine
14
15 # Copy built app to nginx html directory
16 COPY --from=build /app/dist/ /usr/share/nginx/html
17
18 EXPOSE 80
19
20 CMD ["nginx", "--g", "daemon off;"]
```

- ✓ We have now created a GitHub repo, generated a PAT key, cloned the repo locally, added code, created a Dockerfile, and pushed files to GitHub.

✓ Second Stage: Launch EC2 in AWS

1. Login to your AWS account.

The AWS Console Home page shows the 'Recently visited' section with links to S3, AWS Organizations, IAM, Billing and Cost Management, IAM Identity Center, Storage Gateway, EC2 (which is highlighted with a red box), and Elastic Kubernetes Service. The 'Applications' section shows no applications and provides a 'Create application' button. The URL in the browser is <https://us-east-1.console.aws.amazon.com/console/home?region=us-east-1#>.

2. Go to EC2 → Instances → Launch Instance.

The screenshot shows the AWS EC2 home page. On the left, a sidebar menu under 'EC2' includes 'Instances' (which is selected and highlighted with a red box) and 'Images'. The main content area is titled 'Compute' and features a large heading 'Amazon Elastic Compute Cloud (EC2)'. Below it, a sub-headline says 'Create, manage, and monitor virtual servers in the cloud.' A descriptive paragraph explains that Amazon EC2 offers a broadest and deepest compute platform with over 600 instance types. To the right, a callout box titled 'Launch a virtual server' contains a 'Launch instance' button (also highlighted with a red box) and a 'View dashboard' link.

3. Provide a name, **create a key pair**, and leave other settings as default.

The screenshot shows the 'Launch an instance' wizard. The first step, 'Set instance details', is displayed. In the 'Name and tags' section, the 'Name' field contains 'Workstation' (highlighted with a red box). The 'Summary' section shows 'Number of instances' set to 1. In the 'Key pair (login)' section, the 'Key pair name - required' dropdown is set to 'wqqq' (highlighted with a red box), and there is a 'Create new key pair' button.

Create key pair



Key pair name

Key pairs allow you to connect to your instance securely.

k8s

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

RSA

RSA encrypted private and public key pair

ED25519

ED25519 encrypted private and public key pair

Private key file format

.pem

For use with OpenSSH

.ppk

For use with PuTTY



When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** [Learn more](#)

[Cancel](#)

[Create key pair](#)

Keep all other settings as **default**. This VM will be used to build images, create EKS clusters, and communicate with the cluster.

4. Select **Amazon Linux AMI** (RHEL-based) and launch.

5. Copy the public IP, and connect via **Putty/MobaXterm** using your PEM key.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with 'EC2' selected. Under 'Instances', it lists 'Instances', 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', and 'Capacity Reservations'. Under 'Images', it lists 'AMIs' and 'AMI Catalog'. At the bottom of the sidebar is 'Amazon EBS'. The main area shows 'Instances (1/1)' with a search bar and filters for 'Name' and 'Instance ID'. A single instance is listed: 'Workstation' (i-03386499ab45eea13), which is 'Pending' and has an 't2.micro' instance type. Below this, a detailed view for 'i-03386499ab45eea13 (Workstation)' is shown with tabs for 'Details', 'Status and alarms', 'Monitoring', 'Security', 'Networking', 'Storage', and 'Tags'. The 'Details' tab is active, showing 'Instance summary' with fields for 'Instance ID' (i-03386499ab45eea13), 'Public IPv4 address' (35.175.244.216), and 'Private IPv4 addresses' (172.31.86.127). The 'Public IPv4 address' field is highlighted with a red box.

The screenshot shows the MobaXterm session settings dialog box. It has tabs for 'Basic SSH settings', 'Advanced SSH settings', 'Terminal settings', 'Network settings', and 'Bookmark settings'. In the 'Basic SSH settings' tab, fields include 'Remote host' (35.175.244.216), 'Specify username' (ec2-user), and 'Port' (22). These three fields are highlighted with a red box. In the 'Advanced SSH settings' tab, there are checkboxes for 'X11-Forwarding', 'Compression', 'Execute command', 'SSH-browser type' (set to 'SFTP protocol'), and 'Use private key' (with a file path C:\Users\arav\Downloads\k8s.p). The 'Use private key' checkbox and its associated file path are highlighted with a red box. There are also buttons for 'OK' and 'Cancel' at the bottom.

6. Switch to root user and update the server:

```
sudo su -  
yum update -y
```

- ✓ In this stage, we created an EC2 instance, generated a .pem key, launched the instance with default settings, connected to it, and updated the server.

Third Stage: Docker Setup & Push Image

1. Install Git and Docker, then start Docker:

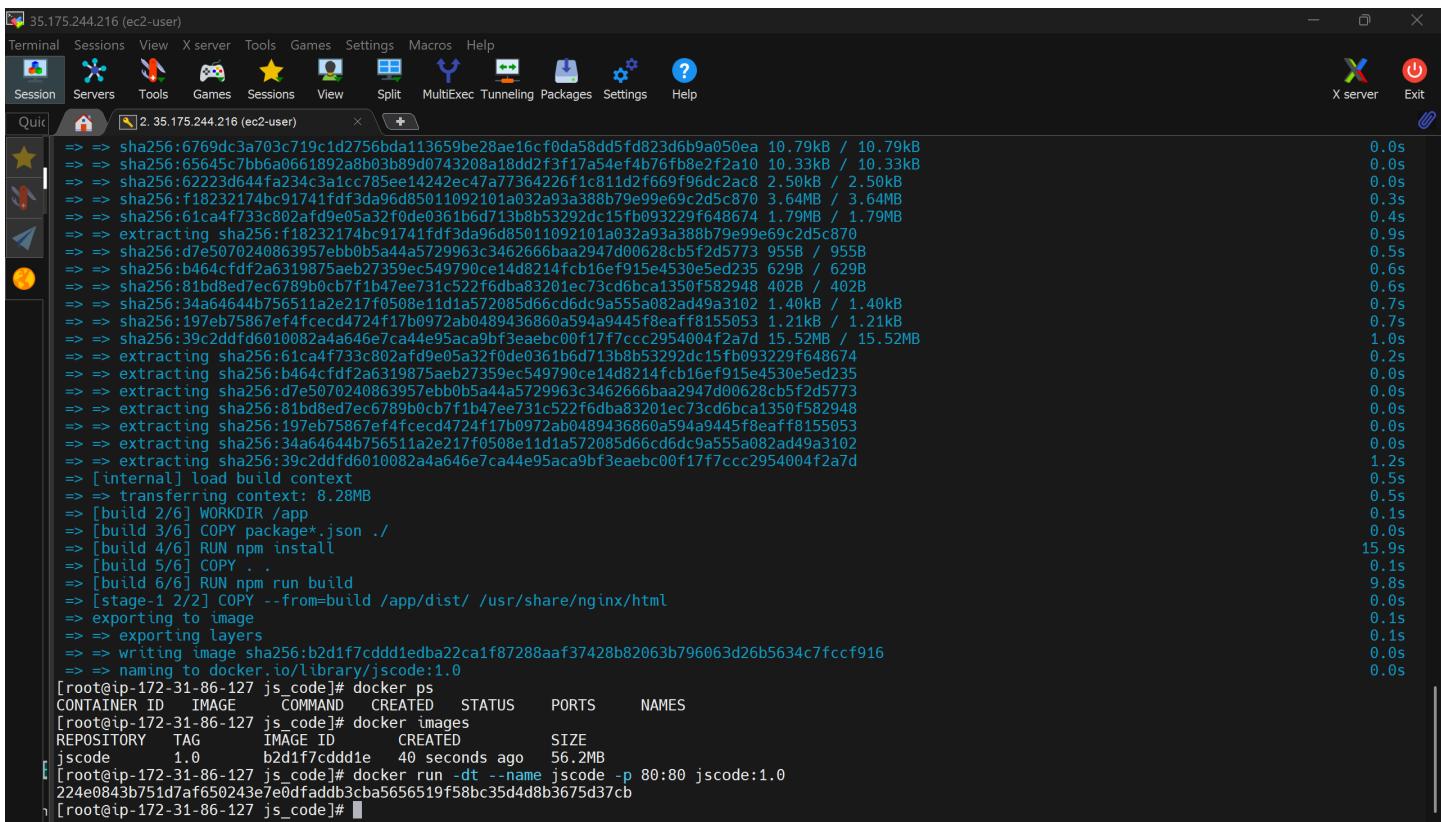
```
yum install -y git docker
systemctl start docker
```

2. Clone the **private repo** using your GitHub username and the **PAT key** as password.

3. Switch to the project directory and **build the Docker image**.

4. Check the image and run the container:

```
docker ps
docker run -dt --name jscode -p 80:80 jscode:1.0
```



The screenshot shows a terminal window titled '2. 35.175.244.216 (ec2-user)' displaying the output of a Docker build process. The output shows the creation of a Docker image named 'jscode' from a Dockerfile, with various layers being built and their sizes listed. The terminal also shows the command to run the container and its status.

```
35.175.244.216 (ec2-user)
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick 2. 35.175.244.216 (ec2-user) + ↻
[star] [home] [refresh] [quit]
--> sha256:6769dc3a703c719c1d2756bda113659be28ae16cf0da58dd5fd823d6b9a050ea 10.79kB / 10.79kB
--> sha256:65645c7bb6a0661892a0b03b89d0743208a18dd2f3f17a54ef4b76fb8e2f2a10 10.33kB / 10.33kB
--> sha256:62223d644fa234c3a1cc785ee14242ec47a77364226fc181d2f669796dc2ac8 2.50kB / 2.50kB
--> sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB
--> sha256:61ca4f733c802afde095a32f0de0361b6d713b8b53292dc15fb093229f648674 1.79MB / 1.79MB
--> extracting sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870
--> sha256:d7e5070240863957eb0b5a44a5729963c3462666baa2947d00628cb5f2d5773 955B / 955B
--> sha256:b464cfdf2a6319875aeb27359ec549790ce14d8214fc16ef915e4530e5ed235 629B / 629B
--> sha256:81bd8ed7ec6789b0cb7f1b47ee731c522f6dba83201ec73cd6bca1350f582948 402B / 402B
--> sha256:34a64644b756511a2e217f0508e11d1a572085d66cd6dc9a555a082ad49a3102 1.40kB / 1.40kB
--> sha256:197eb75867ef4fcecd4724f17b0972ab0489436860a594a9445f8eaff8155053 1.21kB / 1.21kB
--> sha256:39c2ddfd6010082a4a646e7ca44e95aca0bf3eaeabc00f17fccc2954004f2a7d 15.52MB / 15.52MB
--> extracting sha256:61ca4f733c802afde095a32f0de0361b6d713b8b53292dc15fb093229f648674
--> extracting sha256:b464cfdf2a6319875aeb27359ec549790ce14d8214fc16ef915e4530e5ed235
--> extracting sha256:d7e5070240863957eb0b5a44a5729963c3462666baa2947d00628cb5f2d5773
--> extracting sha256:81bd8ed7ec6789b0cb7f1b47ee731c522f6dba83201ec73cd6bca1350f582948
--> extracting sha256:197eb75867ef4fcecd4724f17b0972ab0489436860a594a9445f8eaff8155053
--> extracting sha256:34a64644b756511a2e217f0508e11d1a572085d66cd6dc9a555a082ad49a3102
--> extracting sha256:39c2ddfd6010082a4a646e7ca44e95aca0bf3eaeabc00f17fccc2954004f2a7d
--> [internal] load build context
--> transferring context: 8.28MB
--> [build 2/6] WORKDIR /app
--> [build 3/6] COPY package*.json .
--> [build 4/6] RUN npm install
--> [build 5/6] COPY .
--> [build 6/6] RUN npm run build
--> [stage-1 2/2] COPY --from=build /app/dist/ /usr/share/nginx/html
--> exporting to image
--> writing image sha256:b2d1f7cccc1edba22ca1f87288aaf37428b82063b796063d26b5634c7fccf916
--> naming to docker.io/library/jscode:1.0
[root@ip-172-31-86-127 js_code]# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
[root@ip-172-31-86-127 js_code]# docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
jscode 1.0 b2d1f7cccc1 40 seconds ago 56.2MB
[root@ip-172-31-86-127 js_code]# docker run -dt --name jscode -p 80:80 jscode:1.0
224e0843b751d7af650243e7e0dfaddb3cba5656519f58bc35d4d8b3675d37cb
[root@ip-172-31-86-127 js_code]#
```

-  Our Docker image is just **56.2MB** as we used the Alpine image (lightweight). Here's an improved version of your steps (5–8) with emphasis added on **real-world relevance** and an **interview-style explanation** for the scenario you mentioned:

5. Go inside the running container and manually update files like `index.html` to simulate version changes (e.g., v1 → v2).

- This is useful in **emergency situations** where you need to quickly patch or debug an application without restarting the container or rebuilding the image.

```
[root@ip-172-31-86-127 js_code]# docker exec -it 224e0843b7 sh  
/ # cd /usr/share/nginx/html  
/usr/share/nginx/html # ls  
50x.html    bundle.js   images      index.html  
/usr/share/nginx/html # >index.html  
/usr/share/nginx/html # vi index.html
```

6. To access the app using `<public-ip>:80`, make sure to **modify the EC2 security group** and add **port 80** to inbound rules.
7. Once your changes are tested inside the container, you can **create a new Docker image with all modifications**. This step is **very important** in real-world scenarios where you tweak live containers and want to preserve the state.

```
[root@ip-172-31-86-127 js_code]# docker commit 224e0843b751 jscode:2.0  
sha256:afa8f59407427d1339a6140c73cdedb4ca1153437ab007eb21b4e02980740cb5  
[root@ip-172-31-86-127 js_code]# docker images  
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE  
jscode          2.0          afa8f5940742  5 seconds ago  56.2MB  
jscode          1.0          b2d1f7cddd1e  13 minutes ago  56.2MB  
[root@ip-172-31-86-127 js_code]# docker ps  
CONTAINER ID      IMAGE          COMMAND           CREATED        STATUS        PORTS  
224e0843b751    jscode:1.0    "/docker-entrypoint...."  11 minutes ago  Up 11 minutes  0.0.
```

Interview Scenario:

Q: I did some modifications in a running container. Can I create a Docker image with those changes? If yes, how? **Yes, it's possible.** Here's how you do it:

```
docker commit <container_id> <new_image_name>:<tag>
```

Example:

```
docker commit jscode xaravind/jscode:2.0
```

This creates a new image from the **current state** of the running container, including all manual edits or installed packages.

8. Now login to **Docker Hub**, and create a **private repository** to store your updated Docker images.

The screenshot shows the Docker Hub 'My Hub' interface. On the left is a sidebar with options like 'Repositories', 'Collaborations', 'Settings', etc. The main area is titled 'Repositories' and lists one repository: 'xaravind/alm'. At the top right, there is a blue button labeled 'Create a repository' which is highlighted with a red box.

The screenshot shows the 'Create repository' form on Docker Hub. The 'Repository Name' field contains 'jscode' and is highlighted with a red box. Below it, the 'Visibility' section shows two options: 'Public' (unchecked) and 'Private' (checked). The 'Private' option is highlighted with a red box. To the right, there is a 'Pushing images' section with instructions and a code snippet for using the CLI.

9. Tag images for pushing:

```
docker tag jscode:1.0 xaravind/jscode:v1  
docker tag jscode:2.0 xaravind/jscode:v2
```

10. Login to Docker using your credentials. This will create /root/.docker/config.json which stores credentials. We will use this file to pull images in Kubernetes.

```
[root@ip-172-31-86-127 js_code]# docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope
```

Username: xaravind

Password:

WARNING! Your password will be stored unencrypted in /root/.docker/config.json.

Configure a credential helper to remove this warning. See

<https://docs.docker.com/engine/reference/commandline/login/#credentials-store>

Login Succeeded



11. Push the tagged images to Docker Hub:

```
docker push xaravind/jscode:v1
docker push xaravind/jscode:v2
```

12. Open Docker Hub in browser and verify images are uploaded.

The screenshot shows the Docker Hub interface for the repository `xaravind/jscode`. The left sidebar shows the user's profile and navigation links like Explore, My Hub, Notifications, Instances, and Modern AngularJS Material App. The main content area shows the repository details for `xaravind/jscode`, which was last pushed 2 minutes ago and has a size of 22 MB. The `General` tab is selected. A red box highlights the `Tags` section, which displays two tags: `v2` and `v1`. Both tags are listed under the `Image` type and show a `Pulled` timestamp of "less than 1 day" and a `Pushed` timestamp of "2 minutes". To the right of the tags, there is a "Docker commands" section with a placeholder for pushing a new tag: `docker push xaravind/jscode:tagname`. Below the tags, there is an advertisement for buildcloud.

| Tag | OS | Type | Pulled | Pushed |
|-----|----|-------|-----------------|-----------|
| v2 | | Image | less than 1 day | 2 minutes |
| v1 | | Image | less than 1 day | 2 minutes |

- In this stage, we installed packages, cloned the repo, built Docker images, ran and modified containers, and pushed final images to Docker Hub.

✓ Fourth Stage: EKS Setup with IAM Role

1. Go to AWS Console → Search for IAM.

The screenshot shows the AWS EC2 home page. On the left sidebar, under the EC2 section, there is a 'Services' category. Within this category, the 'IAM' card is highlighted with a red box. The card has the text 'Manage access to AWS resources'. To the right of the services list, there is a sidebar with a 'Launch instances' button and a 'Status check' section indicating '2/2 checks passed'.

2. Click Roles → Create role.

The screenshot shows the IAM Roles page. On the left sidebar, under the 'Access management' section, the 'Roles' option is selected and highlighted with a red box. In the main content area, there is a table titled 'Roles (8)'. The table has columns for 'Role name', 'Trusted entities', and 'Last activity'. The first row shows a role named 'AWSServiceRoleForOrganizations' with 'AWS Service: organizations (Service-Linked)' as the trusted entity and no last activity shown. The 'Create role' button is located at the top right of the roles table and is also highlighted with a red box.

| Role name | Trusted entities | Last activity |
|---|--|---------------|
| AWSServiceRoleForOrganizations | AWS Service: organizations (Service-Linked) | - |
| AWSServiceRoleForSupport | AWS Service: support (Service-Linked) | - |
| AWSServiceRoleForTrustedAdvisor | AWS Service: trustedadvisor (Service-Linked) | - |
| StorageGatewayBucketAccessRole17489440770800.2067476521288938 | AWS Service: storagegateway | 11 days ago |
| StorageGatewayBucketAccessRole17489443923560.3796534906349802 | AWS Service: storagegateway | 11 days ago |
| StorageGatewayBucketAccessRole17489445566890.5340462669381898 | AWS Service: storagegateway | 11 days ago |

3. Select Trusted entity type: AWS Service and use case as EC2, then click Next.

The screenshot shows the 'Select trusted entity' step of the IAM role creation wizard. The 'AWS service' option is selected and highlighted with a red box. The 'Service or use case' dropdown is set to 'EC2' and is also highlighted with a red box.

4. Choose **AdministratorAccess** policy (has full EC2 + EKS permissions), then click **Next**.

The screenshot shows the 'Add permissions' step of the IAM role creation wizard. The 'AdministratorAccess' policy is selected and highlighted with a red box in the list of available policies.

5. Name the role and click **Create Role**.

Step 2
Add permissions

Step 3
Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
k8s-role

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.

Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _+=,. @-/\[{\}]#\$%^&`~`

Step 1: Select trusted entities

Trust policy

6. Go to EC2 Instances, select your instance → Actions → Security → Modify IAM role.

EC2

Instances (1/1) Info

Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

All states ▾

Name | Instance ID | Instance state | Instance type

Workstation i-03386499ab45eea13 Running

Actions ▾

Launch instances

Instance diagnostics

Instance settings

Networking

Security

Image and templates

Monitor and troubleshoot

7. Attach the newly created IAM role to the instance.

Modify IAM role info

Attach an IAM role to your instance.

Instance ID: i-03386499ab45eea13 (Workstation)

IAM role:

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

k8s-role

Create new IAM role

Cancel

Update IAM role

Now your EC2 instance can control AWS services via AWS CLI.

8. AWS CLI comes pre-installed in Amazon Linux. For other OS, you'll need to install it manually.

9. Check AWS access:

```
aws s3 mb s3://test-k8s.js  
aws s3 ls
```

10. Create an `eks.yml` file locally and push it to GitHub.

The screenshot shows a VS Code interface with the following details:

- EXPLORER:** Shows files like `k8s-resources`, `11_04`, and `pvt-repo`.
- CODE EDITOR:** Displays the `eks.yml` file content:

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: expense
  region: us-east-1
managedNodeGroups:
- name: expense
  instanceType: m5.large
  desiredCapacity: 3
  spot: true
```
- TERMINAL:** Shows the command being run:

```
git add . ; git commit -m "11" ; git push
```
- OUTPUT:** Shows the GitHub push process:

```
Compressing objects: 100% (32/32), done.  
Writing objects: 100% (34/34), 2.01 MiB | 921.00 KiB/s, done.  
Total 34 (delta 6), reused 0 (delta 0), pack-reused 0 (from 0)  
remote: Resolving deltas: 100% (6/6), done.  
To https://github.com/xaravind/pvt-repo.git  
9bb0ae0..d5153a7 main -> main
```
- SIDE BAR:** Shows a terminal session with the command `git add . ; git commit -m "11" ; git push`.

> We are using ****spot instances**** for cost-saving ($\sim 90\%$ discount), but they may be terminated

```
```bash  
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
name: project
region: us-east-1
```

```
managedNodeGroups:
- name: project
instanceType: m5.large
desiredCapacity: 3
spot: true
```
```

> This will provision VPCs, subnets, IAM roles, node groups – and may take several minutes.

11. Pull the updated code into the EC2 instance.

```
[root@ip-172-31-86-127 ~]# cd pvt-repo/  
[root@ip-172-31-86-127 pvt-repo]# ls  
README.md  js_code  
[root@ip-172-31-86-127 pvt-repo]# git pull -a  
[root@ip-172-31-86-127 pvt-repo]# ls  
README.md  eks.yml  js_code
```

12. Install eksctl : Follow instructions from (<https://eksctl.io/installation>)

```
[root@ip-172-31-86-127 pvt-repo]# ls  
README.md  eks.yml  js_code  
[root@ip-172-31-86-127 pvt-repo]# # for ARM systems, set ARCH to: `arm64`, `armv6` or `arm  
ARCH=amd64  
PLATFORM=$(uname -s)_$ARCH  
  
curl -sLO "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_${PLATFORM}.t  
# (Optional) Verify checksum  
curl -sL "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_checksums.tx  
tar -xzf eksctl_${PLATFORM}.tar.gz -C /tmp && rm eksctl_${PLATFORM}.tar.gz  
  
sudo mv /tmp/eksctl /usr/local/bin  
eksctl_Linux_amd64.tar.gz: OK  
rm: remove regular file 'eksctl_Linux_amd64.tar.gz'? y  
  
[root@ip-172-31-86-127 pvt-repo]# eksctl version  
0.210.0
```

13. Use the eks.yml file to create an EKS cluster.

This takes time and will create required AWS services like nodes, VPCs, SGs, ASG, etc.

```
[root@ip-172-31-86-127 pvt-repo]# eksctl create cluster --config-file=eks.yml  
2025-06-15 07:05:09 [i]  eksctl version 0.210.0  
2025-06-15 07:05:09 [i]  using region us-east-1  
2025-06-15 07:05:09 [i]  setting availability zones to [us-east-1b us-east-1a]  
2025-06-15 07:05:09 [i]  subnets for us-east-1b - public:192.168.0.0/19 private:192.168.64  
2025-06-15 07:05:09 [i]  subnets for us-east-1a - public:192.168.32.0/19 private:192.168.9  
2025-06-15 07:05:09 [i]  nodegroup "project" will use "" [AmazonLinux2023/1.32]  
2025-06-15 07:05:09 [i]  using Kubernetes version 1.32  
2025-06-15 07:18:00 [✓]  EKS cluster "project" in "us-east-1" region is ready
```

14. Install kubectl to interact with the cluster.

```
[root@ip-172-31-86-127 pvt Repo]# curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.13.0-eks-802817d/kubectl
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current
                                         Dload  Upload   Total   Spent   Left  Speed
100 57.3M  100 57.3M    0      0  21.2M      0  0:00:02  0:00:02 --:--:-- 21.2M
[root@ip-172-31-86-127 pvt Repo]# chmod +x ./kubectl
[root@ip-172-31-86-127 pvt Repo]# mv ./kubectl /usr/local/bin
[root@ip-172-31-86-127 pvt Repo]# kubectl version --client
Client Version: v1.33.0-eks-802817d
Kustomize Version: v5.6.0
```



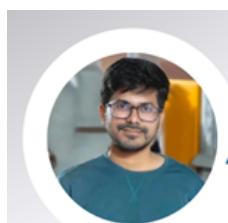
15. Test kubectl:

```
[root@ip-172-31-86-127 pvt Repo]# kubectl api-resources | wc -l
66
[root@ip-172-31-86-127 pvt Repo]# kubectl get nodes
NAME                  STATUS  ROLES   AGE    VERSION
ip-192-168-27-169.ec2.internal  Ready   <none>  9m7s  v1.32.3-eks-473151a
ip-192-168-55-115.ec2.internal  Ready   <none>  9m8s  v1.32.3-eks-473151a
ip-192-168-63-135.ec2.internal  Ready   <none>  9m7s  v1.32.3-eks-473151a
```

16. Create a base64 secret from Docker credentials to use in Kubernetes secrets:

```
[root@ip-172-31-86-127 pvt Repo]# cat ~/.docker/config.json | base64 -w 0
ewoJImF1dGhzIjxxxxxxxxxxxxxxxxxxxxxxxxxxxxCn0=
```

17. Create a manifest file locally, push it to GitHub, and pull it on the EC2 instance.



Aravind Basava

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer (Left):** Shows a project structure with files like `manifest.yml`, `eks.yml`, `Dockerfile`, and several YAML configuration files for Kubernetes services and secrets.
- Code Editor (Top Right):** Displays the content of `manifest.yml`, which defines a Service named `frontend` in the `project` namespace. It uses a LoadBalancer type and selector for pods labeled `app: frontend` and `tier: web`. The service exposes port `80` (TCP) with a target port of `8080`.
- Terminal (Bottom):** Shows a terminal session with the following history:
 - Initial prompt: `aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)`
 - User command: `$ ls`
 - Terminal response: `eks.yml js_code/ manifest.yml README.md`
 - Final prompt: `aravi@Aravind MINGW64 ~/11_04/pvt-repo (main)`
 - User command: `$ git add . ; git commit -m "ll" ; git push`
- Bottom Status Bar:** Shows icons for PROBLEMS (87), OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, and SPELL CHECKER (87).
- Right Sidebar:** Shows a powershell and bash terminal session for the `pvt-repo` directory.

```
```bash
[root@ip-172-31-86-127 pvt Repo]# cat manifest.yml
apiVersion: v1
kind: Secret
metadata:
name: regcred
namespace: project
type: kubernetes.io/dockerconfigjson
data:
.dockerconfigjson: ewoJImF1dGhzIjogewoJCSJodHRwczovL2luZGV4LmRvY2tlci5pbv92MS8i0iB7CgkJCSJhdXR

Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
name: frontend
namespace: project
labels:
 app: frontend
 environment: project
spec:
replicas: 3
selector:
 matchLabels:
 app: frontend
 tier: web
template:
 metadata:
 labels:
 app: frontend
 tier: web
spec:
```

```

imagePullSecrets:
 - name: regcred
containers:
 - name: frontend
 image: xaravind/jscode:v2
 ports:
 - containerPort: 80
 resources:
 requests:
 memory: "64Mi"
 cpu: "250m"
 limits:
 memory: "256Mi"
 cpu: "500m"
 readinessProbe:
 httpGet:
 path: /
 port: 80
 initialDelaySeconds: 5
 periodSeconds: 10
 livenessProbe:
 httpGet:
 path: /
 port: 80
 initialDelaySeconds: 15
 periodSeconds: 20

Service
apiVersion: v1
kind: Service
metadata:
 name: frontend
 namespace: project
spec:
 type: LoadBalancer
 selector:
 app: frontend
 tier: web
 ports:
 - protocol: TCP
 port: 80
 targetPort: 80

```

Above manifest file defines everything required to deploy a containerized application securely and publicly in a Kubernetes cluster.

## 1. Docker Registry Secret

- **Purpose:** Allows Kubernetes to pull an image from a **private Docker Hub repository** using a base64-encoded Docker config ( `~/.docker/config.json` ).

- **Key:** `regcred` (referenced in the deployment under `imagePullSecrets` ).

## 2. Deployment

- **Replicas:** 3 – Ensures high availability by running 3 pods.
- **Image:** `xaravind/jscode:v1` –
- **Probes:**
  - **Readiness Probe** – Checks if the container is ready to serve traffic.
  - **Liveness Probe** – Monitors container health and restarts if unhealthy.
- **Resources:**
  - **Requests:** Memory: 64Mi , CPU: 250m
  - **Limits:** Memory: 256Mi , CPU: 500m
- **Image Pull Secret:** Uses the previously created `regcred` secret to authenticate with Docker Hub.

## 3. LoadBalancer Service

- **Type:** `LoadBalancer` – Automatically provisions an **external load balancer** (like AWS ELB when running on EKS).
- **Port Mapping:**
  - **port:** 80 (exposed publicly)
  - **targetPort:** 80 (mapped to the container port)

## Summary

- ✓ \*\*Pulls a private image\*\* securely using a Docker Hub token
- ✓ \*\*Deploys the frontend application\*\* as a replicated, monitored set of pods
- ✓ \*\*Uses probes\*\* to maintain app health and availability
- ✓ \*\*Exposes the app\*\* to the public internet via a load balancer

## 18. Create a namespace and apply the manifest file:

```
kubectl create namespace frontend
kubectl apply -f manifest.yaml -n frontend
```

## 19. Check pods and services. Kubernetes will create a **LoadBalancer service** with a random **NodePort** (30000–32767).

```
[root@ip-172-31-86-127 pvt-repo]# kubectl get pods -n project
NAME READY STATUS RESTARTS AGE
frontend-644df5c8b4-97gbj 0/1 Running 1 (32s ago) 92s
```

```

frontend-6467b94fb8-hghzx 0/1 Running 0 11s
frontend-6467b94fb8-t95fs 1/1 Running 0 25s
frontend-6d78b8b5b6-xw8hl 0/1 CrashLoopBackOff 5 (21s ago) 6m22s
[root@ip-172-31-86-127 pvt Repo]# kubectl describe pod frontend-6467b94fb8-hghzx -n project
Name: frontend-6467b94fb8-hghzx

```

Tolerations:

node.kubernetes.io/not-ready:NoExecute op=Exists for 300s	
node.kubernetes.io/unreachable:NoExecute op=Exists for 300s	

Events:

Type	Reason	Age	From	Message
Normal	Scheduled	28s	default-scheduler	Successfully assigned project/frontend-6467b94
Normal	Pulled	27s	kubelet	Container image "xaravind/jscode:v1" already p
Normal	Created	27s	kubelet	Created container: frontend
Normal	Started	27s	kubelet	Started container frontend

```

[root@ip-172-31-86-127 pvt Repo]# kubectl get svc frontend -n project
NAME TYPE CLUSTER-IP EXTERNAL-IP

```

20. To access the app, add inbound rule for NodePort range (30000–32767) in the security group.

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with navigation links like Dashboard, EC2 Global View, Events, Instances, Images, and Elastic Block Store. The main area displays a table of instances under the heading 'Instances (1/4) Info'. The table includes columns for Name, Instance ID, Instance state, Instance type, Status check, and Alarm status. One row is selected, showing the instance 'expense-expense-Node' with ID i-0074d84196ae7550e, which is running. Below the table, a detailed view for the selected instance is shown, with tabs for Details, Status and alarms, Monitoring, Security (which is currently selected), Networking, Storage, and Tags. Under the Security tab, there's a section for 'Security details' showing the IAM Role (eksctl-expense-nodegroup-expense-NodeInstanceRole-W4JHMZ48M6Pc) and Owner ID (115690362715). To the right, the Launch time is listed as Sun Jun 15 2025 12:46:36 GMT+0530 (India Standard Time). A red box highlights the 'Security groups' section, which lists the security group sg-03b173931a41da8a6 (eks-cluster-sg-expense-1309357992).

21. Go to EC2 → Nodes → Security Group → Edit Inbound Rules → Add port range.

**Edit inbound rules** Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0e866a14db5aca88e	All traffic	All	All	Cust...	Allows EFA traffic, which is not matched by CIDR rules.
sgr-01b9c0e04795eb91	All traffic	All	All	Cust...	
sgr-08911bdd2de7c2080	All traffic	All	All	Cust...	Allow unmanaged nodes to communicate with control plane (all ports)
-	Custom TCP	TCP	30000 - 31000	Any...	Allow unmanaged nodes to communicate with control plane (all ports)

**Add rule**

## 22. Access the application via LoadBalancer or NodePort.

**Load balancers (1/1)**

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Name	DNS name	State	VPC ID	Availability Zones	Type
aefdbdc4ee74c4c8db74e87daff5a637	aefdbdc4ee74c4c8db74e87daff5a637	-	vpc-04f1e648e35cb2396	2 Availability Zones	classic

**Load balancer: aefdbdc4ee74c4c8db74e87daff5a637**

simultaneously, choose Manage instances.

Instance ID	Name	Health status	Health status descri...	Security groups
i-0074d84196ae7550e	expense-expense-Node	Out-of-service	Instance has failed at l...	eks-cluster-sg-ex...
i-061ea6c9194fdebce	expense-expense-Node	Out-of-service	Instance has failed at l...	eks-cluster-sg-ex...
i-09ac5a24a0238e6d7	expense-expense-Node	Out-of-service	Instance has failed at l...	eks-cluster-sg-ex...

This screenshot shows the AWS EC2 Load Balancer console. The left sidebar includes sections for Dashboard, EC2 Global View, Events, Instances (with sub-options like Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), and Network & Security. The main content area shows a message about migrating to a next-generation load balancer. Below it, a section titled 'Distribution of targets by Availability Zone (AZ)' provides details on registered instances and their health states. A table lists three target instances, all of which are marked as 'In-service' (highlighted with a red box). The table has columns for Instance ID, Name, Health status, and Health status description.

Instance ID	Name	Health status	Health status description
<a href="#">i-0074d84196ae7550e</a>	expense-expense-Node	In-service	Not applicable
<a href="#">i-061ea6c9194fdebce</a>	expense-expense-Node	In-service	Not applicable
<a href="#">i-09ac5a24a0238e6d7</a>	expense-expense-Node	In-service	Not applicable

```
```bash
curl http://<EXTERNAL-IP>
curl http://ab39bfabc67f9405f8e6c5b308273a51-706994256.us-east-1.elb.amazonaws.com/
````
```

This screenshot shows a web browser window with the URL 'ab39bfabc67f9405f8e6c5b308273a51-706994256.us-east-1.elb.amazonaws.com/#!/login'. The page displays a 'Login' form with fields for Email and Password, a 'Remember me' checkbox, and a 'LOGIN' button. Below the form, there is a link to 'Sign up'.

23. Update manifest file with v2 image - xaravind/jscode:v2

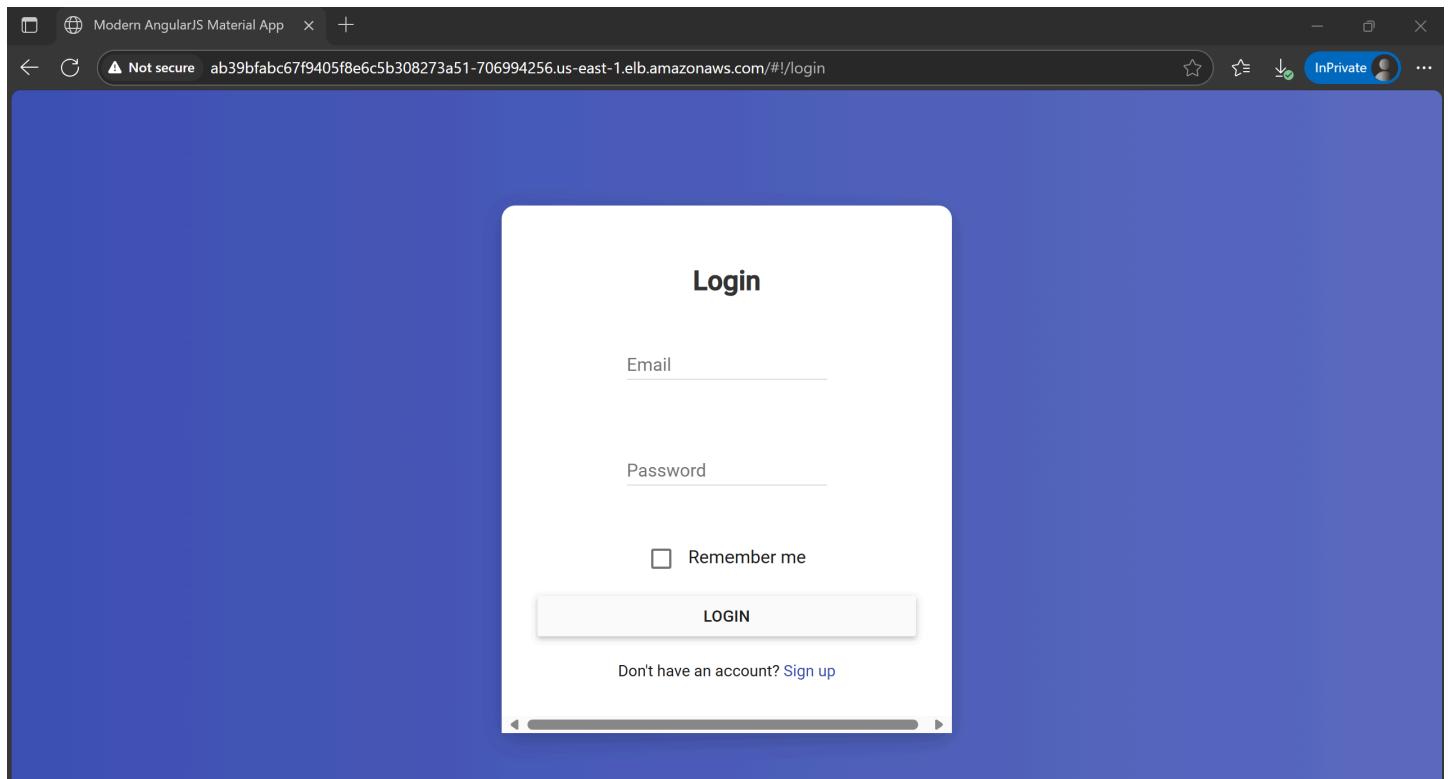
24. repply the changes

```
[root@ip-172-31-86-127 pvt Repo]# vi manifest.yml
[root@ip-172-31-86-127 pvt Repo]# kubectl get pods -o wide -n project
NAME READY STATUS RESTARTS AGE IP NODE
frontend-6467b94fb8-hghzx 1/1 Running 0 7m12s 192.168.62.210 ip-192-1
frontend-6467b94fb8-hs94j 1/1 Running 0 6m59s 192.168.43.93 ip-192-1
frontend-6467b94fb8-t95fs 1/1 Running 0 7m26s 192.168.29.173 ip-192-1
[root@ip-172-31-86-127 pvt Repo]# kubectl apply -f manifest.yml
secret/regcred unchanged
deployment.apps/frontend configured
service/frontend unchanged
[root@ip-172-31-86-127 pvt Repo]# kubectl get pods -o wide -n project
NAME READY STATUS RESTARTS AGE IP NODE
frontend-6467b94fb8-hghzx 1/1 Running 0 7m55s 192.168.62.210 ip-192-1
frontend-6467b94fb8-hs94j 1/1 Running 0 7m42s 192.168.43.93 ip-192-1
frontend-6467b94fb8-t95fs 1/1 Running 0 8m9s 192.168.29.173 ip-192-1
frontend-f8f8b89df-zn2gn 0/1 Running 0 4s 192.168.2.38 ip-192-1
[root@ip-172-31-86-127 pvt Repo]# kubectl get pods -o wide -n project
NAME READY STATUS RESTARTS AGE IP NODE
frontend-6467b94fb8-hghzx 1/1 Running 0 8m3s 192.168.62.210 ip-192-1
frontend-6467b94fb8-hs94j 1/1 Running 0 7m50s 192.168.43.93 ip-192-1
frontend-6467b94fb8-t95fs 1/1 Running 0 8m17s 192.168.29.173 ip-192-1
frontend-f8f8b89df-zn2gn 0/1 Running 0 12s 192.168.2.38 ip-192-1
[root@ip-172-31-86-127 pvt Repo]# kubectl get pods -o wide -n project
NAME READY STATUS RESTARTS AGE IP NODE
frontend-6467b94fb8-hghzx 1/1 Running 0 8m10s 192.168.62.210 ip-192-1
frontend-6467b94fb8-hs94j 1/1 Running 0 7m57s 192.168.43.93 ip-192-1
frontend-f8f8b89df-ggncq 0/1 Running 0 5s 192.168.47.194 ip-192-1
frontend-f8f8b89df-zn2gn 1/1 Running 0 19s 192.168.2.38 ip-192-1
[root@ip-172-31-86-127 pvt Repo]# kubectl get pods -o wide -n project
NAME READY STATUS RESTARTS AGE IP NODE
frontend-f8f8b89df-ggncq 1/1 Running 0 61s 192.168.47.194 ip-192-168-
frontend-f8f8b89df-xmjqn 1/1 Running 0 49s 192.168.39.242 ip-192-168-
frontend-f8f8b89df-zn2gn 1/1 Running 0 75s 192.168.2.38 ip-192-168-
```



- Observe how Kubernetes rolls out the new version of your app without downtime – this is called a **rolling update**.

25. Access the application, we easily updated the code with zero downtime. and with single command.



## ✓ Final Cleanup

- Once done, **clean up your project** to avoid charges:
  - Delete EKS cluster
  - Remove EC2 instance
  - Delete S3 buckets and Docker images if not needed

```
eksctl delete cluster --config-file=eks.yml
[root@ip-172-31-86-127 pvt-repo]# eksctl delete cluster --config-file=eks.yml
2025-06-15 08:25:07 [i] deleting EKS cluster "project"
2025-06-15 08:25:07 [i] will drain 0 unmanaged nodegroup(s) in cluster "project"
2025-06-15 08:25:07 [i] starting parallel draining, max in-flight of 1
2025-06-15 08:25:07 [i] deleted 0 Fargate profile(s)
2025-06-15 08:25:08 [✓] kubeconfig has been updated
2025-06-15 08:25:08 [i] cleaning up AWS load balancers created by Kubernetes objects
2025-06-15 08:25:33 [i]
2 sequential tasks: { delete nodegroup "project", delete cluster control plane "projec
}
2025-06-15 08:25:33 [i] will delete stack "eksctl-project-nodegroup-project"
2025-06-15 08:25:33 [i] waiting for stack "eksctl-project-nodegroup-project" to get d
2025-06-15 08:25:33 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:26:03 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:26:48 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:27:40 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr
2025-06-15 08:28:30 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr
```

```
2025-06-15 08:29:31 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr"
2025-06-15 08:30:55 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr"
2025-06-15 08:32:25 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr"
2025-06-15 08:34:07 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr"
2025-06-15 08:35:46 [i] waiting for CloudFormation stack "eksctl-project-nodegroup-pr"
2025-06-15 08:35:46 [i] will delete stack "eksctl-project-cluster"
....
```

- ✓ In this stage, we **created an IAM role** with Administrator access, **attached the role** to the EC2 instance, **installed eksctl and kubectl packages**, **created an EKS cluster**, **set up a namespace**, **applied the manifest file**, **accessed the application**, **updated the deployment** with image v2, and finally **performed cleanup** of all resources.



Aravind Basava