Capstone Project – Book-My-Show DevOps Lifecycle

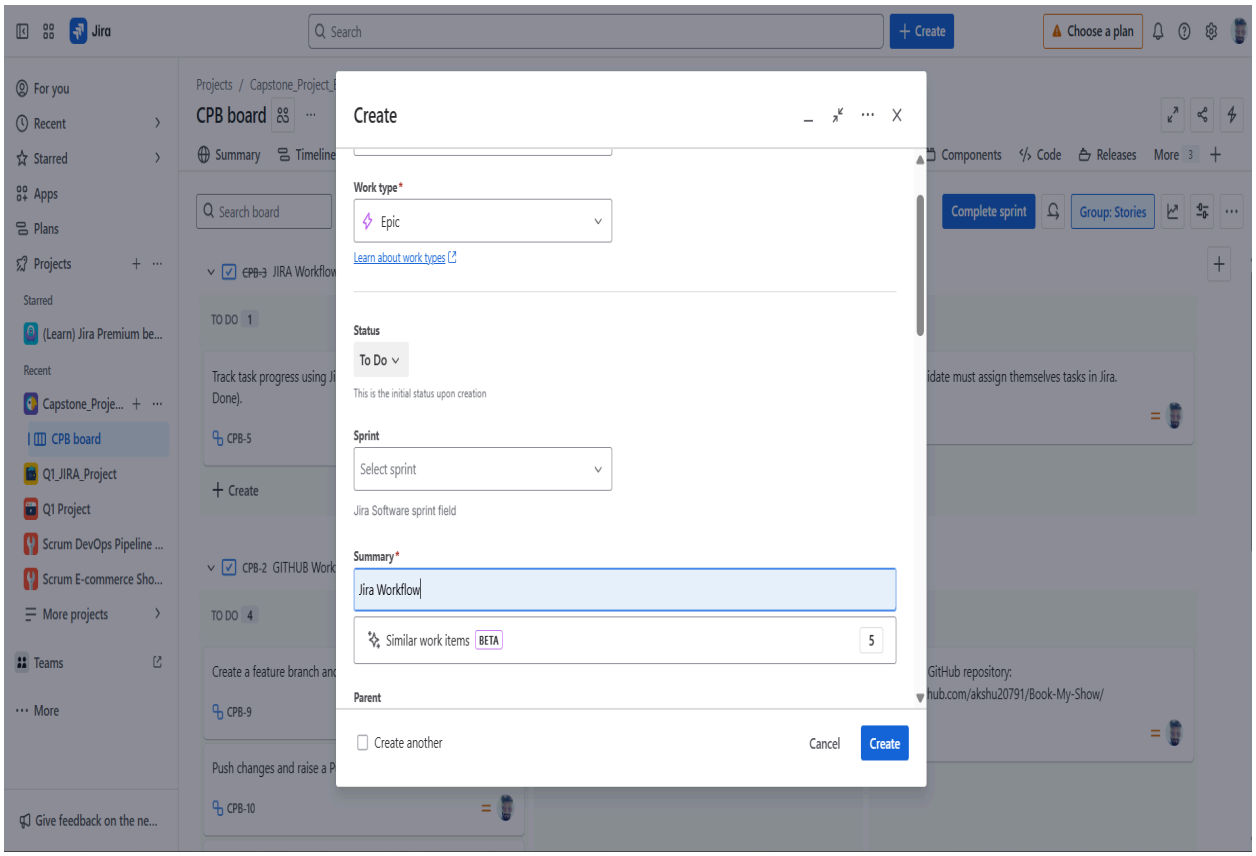
Name: ARJU Kumar
Github Repo: https://github.com/arju7jha/Book-My-Show.git

Stage 1: Jira Workflow

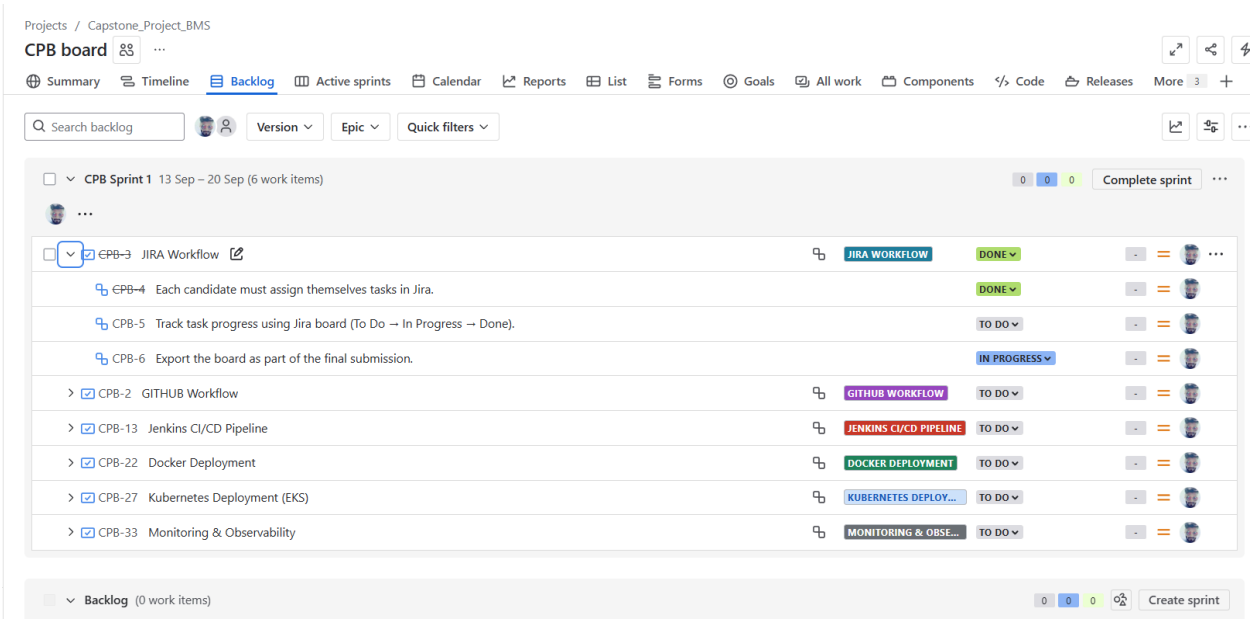
Step 1: Log in to Jira workspace.

Step 2: Go to the **project board** for creating the tasks and subtasks for the Capstone Project and to be assigned to myself.

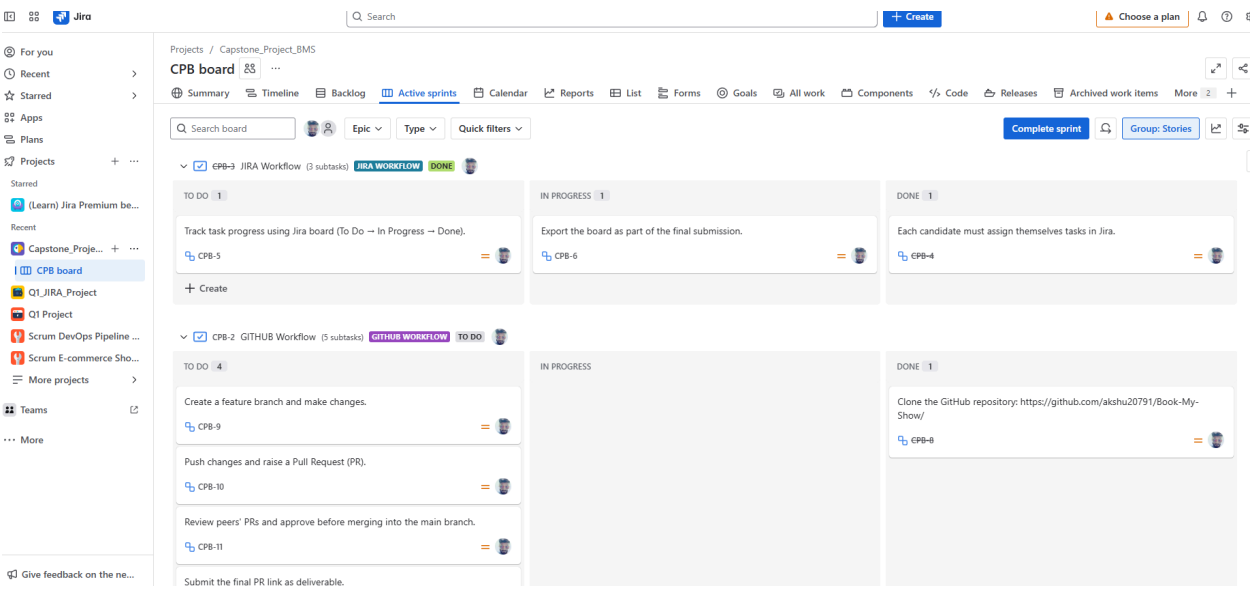
Step 3: Create **"Epic"**for each Steps of "Book-My-Show DevOps Lifecycle - Capstone Project " and for each epic assign the subtask of the Capstone Project Steps and open each task , Click **Assign** → Choose my name to be assigned tasks to me.



Step 4: Now got to the **“Backlog”** tab and start the sprint and move all the tasks to the stared Sprint.



Step 5: Now got to the **“Active Sprints”** and check all the tasks and Subtasks and according to progress of the tasks and subtasks update the status from **ToDO → In-progress → Done**



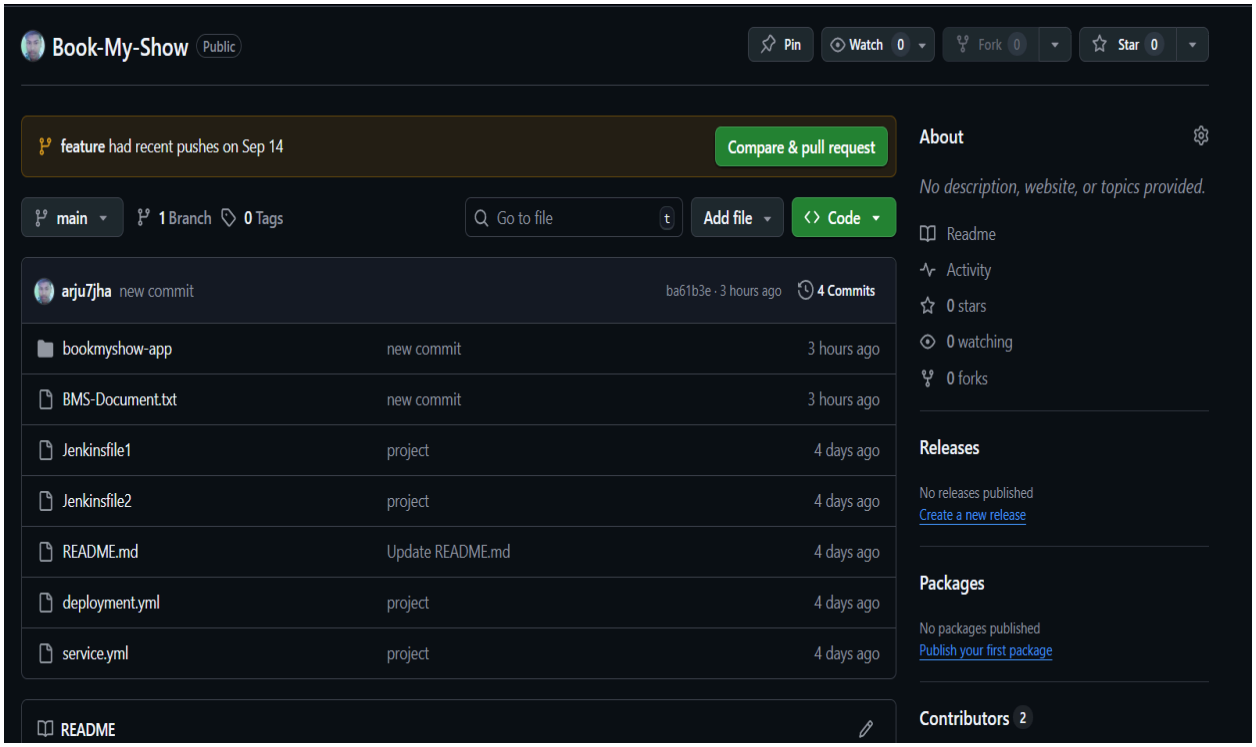
Stage 2: GitHub Workflow

Step 1: Clone the GitHub repository: https://github.com/arju7jha/Book-My-Show.git using below command :

→ **"git clone** https://github.com/arju7jha/Book-My-Show.git **"**
→ **"cd Book-My-Show"**

Step 2: Create a feature branch and make changes, push changes and raise a Pull Request (PR). using below command :

→ **"git branch"**
→ **"git checkout -b feature"**
→ **"git branch"**
→ **"git add ."**
→ **"git commit -m "new feature branch" "**
→ **"git push -u origin feature"**



Step 3: Review peer's PRs and approve before merging into the main branch.

A). Review Peers' PRs

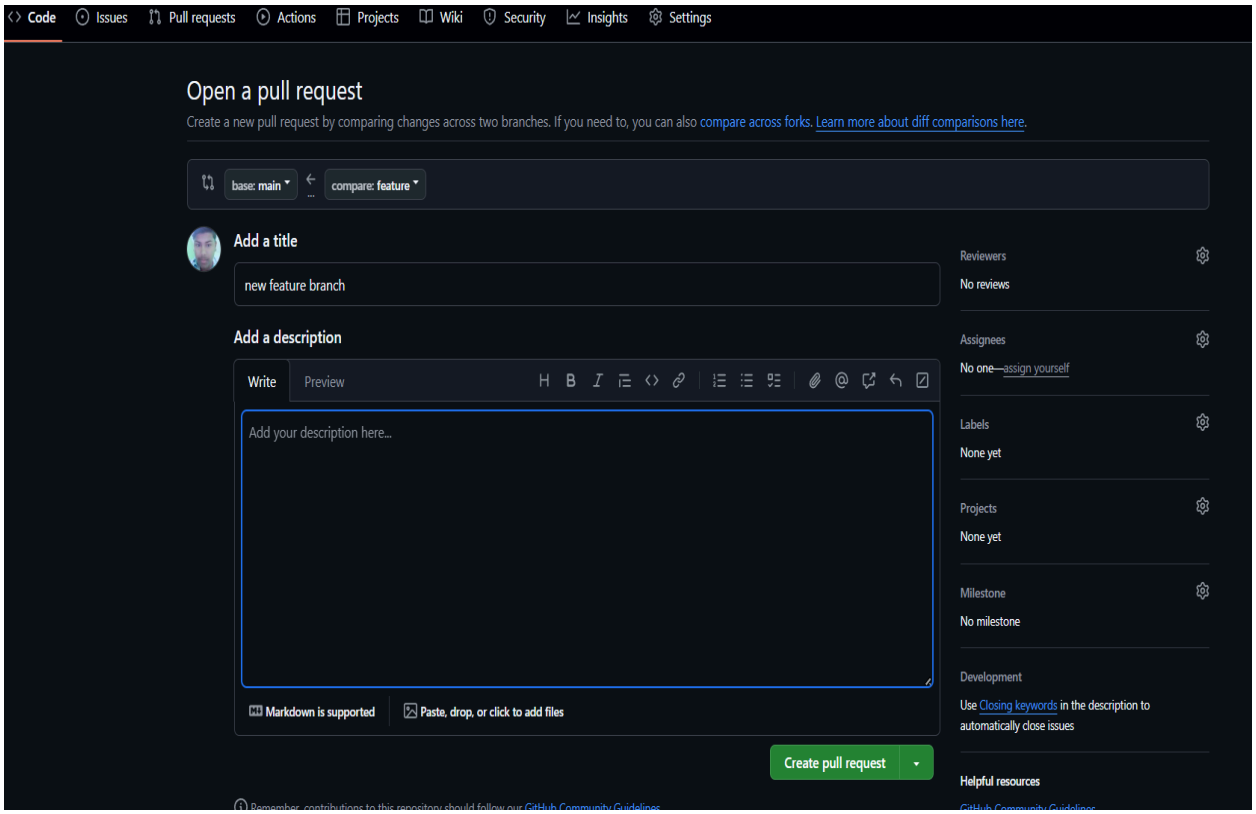
Go to the Pull Requests tab.

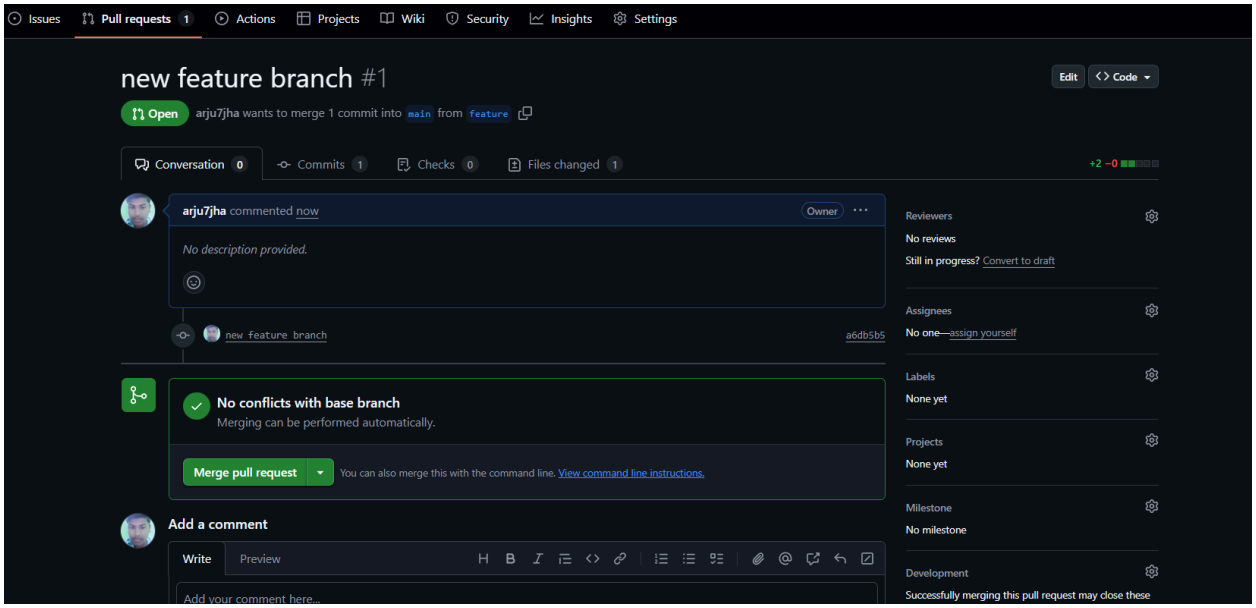
Open your peers' PRs, review their code, and leave comments or approve.

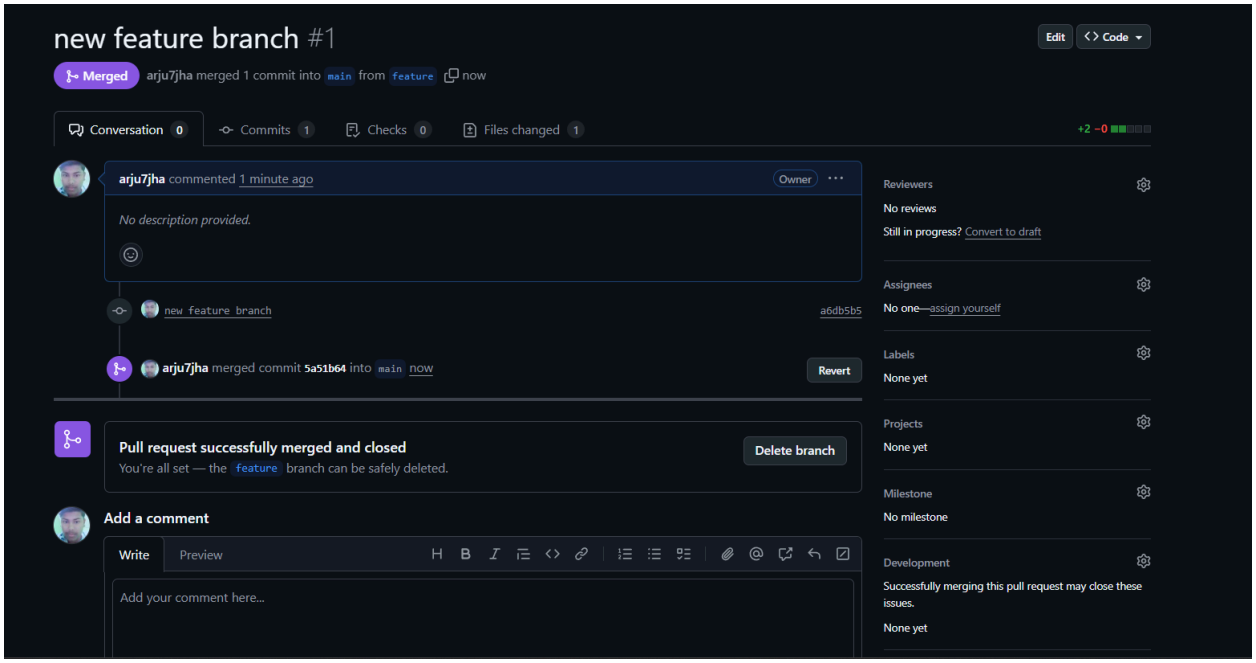
Make sure everything passes tests and meets requirements.

B). Merge PR into Main Branch

Once PRs are approved, merge into main branch using the Merge button on GitHub.







Step 4: Submit the Final PR Link

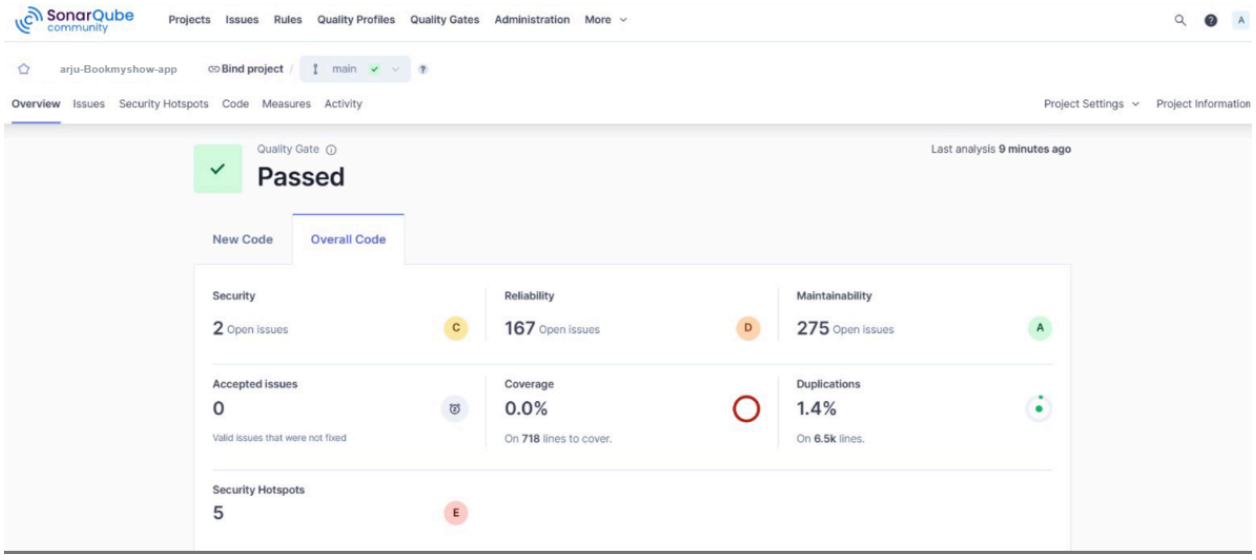
→ **URL of your Pull Request :** https://github.com/arju7jha/Book-My-Show/tree/feature

Stage 3: Jenkins CI/CD Pipeline

Step 1: Cleared Jenkins' workspace before starting the build so there were **no leftover files** from previous runs.

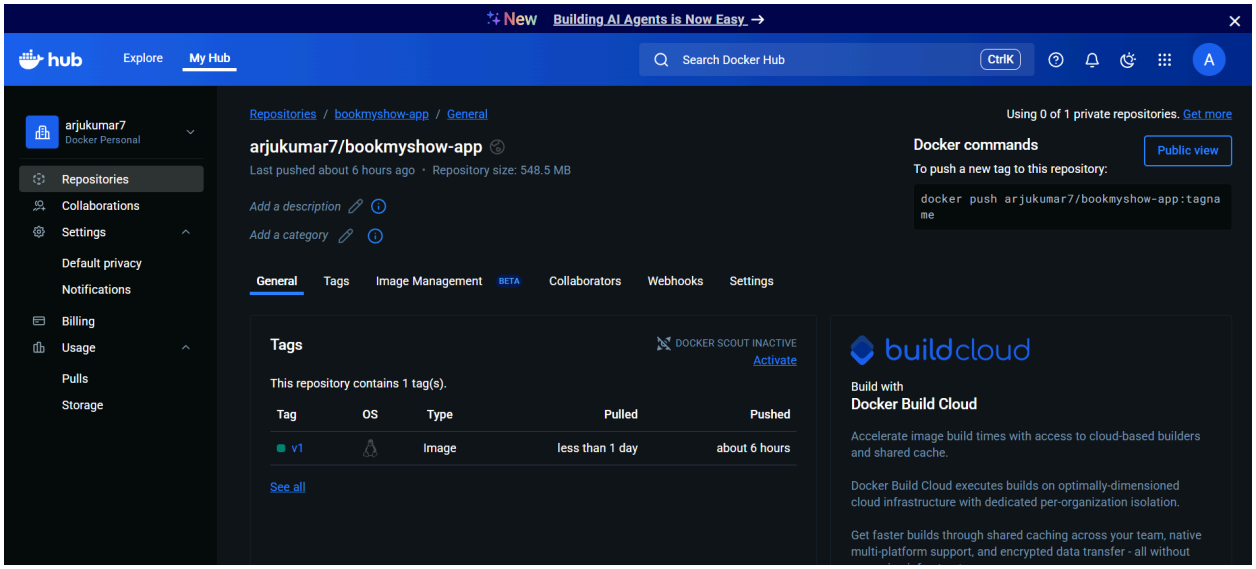
Step 2: Use Jenkins to pull the latest code from our GitHub repository. This ensures the pipeline always works with the latest code changes.

Step 3: Run the SonarQube to analyze the code for bugs, security issues, and code smells. The pipeline only continued if the quality gate passed (good, clean code).

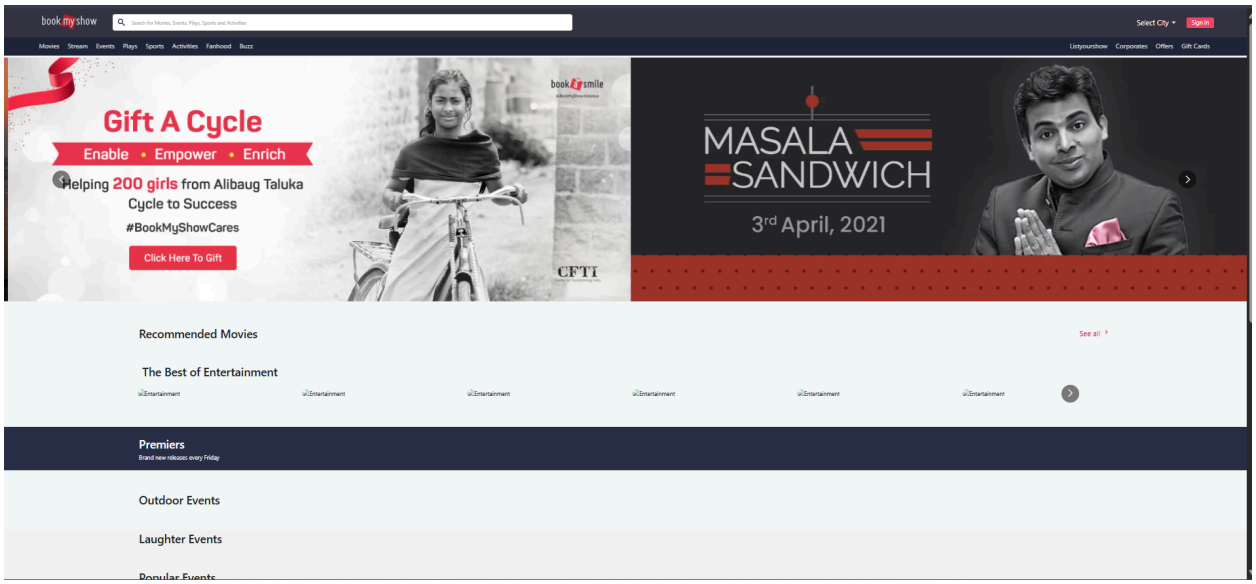


Step 4: Installed all the Node.js dependencies by running: **" npm install "**

Step 5: Created a Docker image of the app and pushed it to DockerHub with a version tag. This makes the app container image available anywhere for deployment.



Step 6: Run the Docker container using the image we just built and pushed to dockerhub. The app is now running live on port 3000.



Step 7: Configure Jenkins to send an email after the build. The email tells us whether the build succeeded or failed to keep the team updated.

Stage 4: Docker Deployment

Step 1: Create a **`Dockerfile`** at the root of the Book-My-Show project to build the BMS app image.

ne_Project > Book-My-Show > bookmyshow-app > Dockerfile

```
FROM node:18

WORKDIR /app

COPY package.json package-lock.json ./

# Install chokidar@3 explicitly (fix for missing dependency)
RUN npm install chokidar@3 --legacy-peer-deps

# Your existing postcss fix
RUN npm install postcss@8.4.21 postcss-safe-parser@6.0.0 --legacy-peer-deps

# Install rest of the dependencies
RUN npm install --legacy-peer-deps

COPY . .

EXPOSE 3000

ENV NODE_OPTIONS=--openssl-legacy-provider
ENV PORT=3000

CMD ["npm", "start"]
```

Step 2: To Build a **`Docker Image`** and push to DockerHub via Jenkins created a Jenkins file and build the script in jenkins.

→ In Jenkins go to manage Jenkins and add DockerHub Credentials as: **dockerhub-cred**

→ To run Jenkins Pipeline and push to DockerHub via Jenkins :

A). Create a new Jenkins Pipeline job.

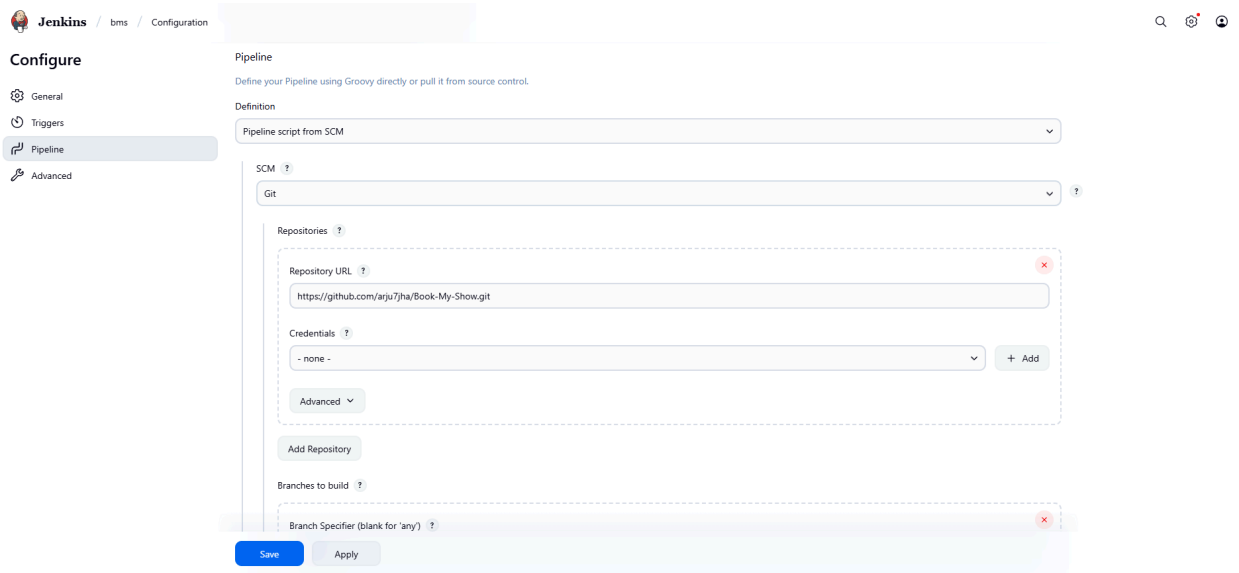
B). Under **Definition**, choose: **Pipeline script from SCM**

C). In **SCM:** Select **Git** and the **Repository URL:**

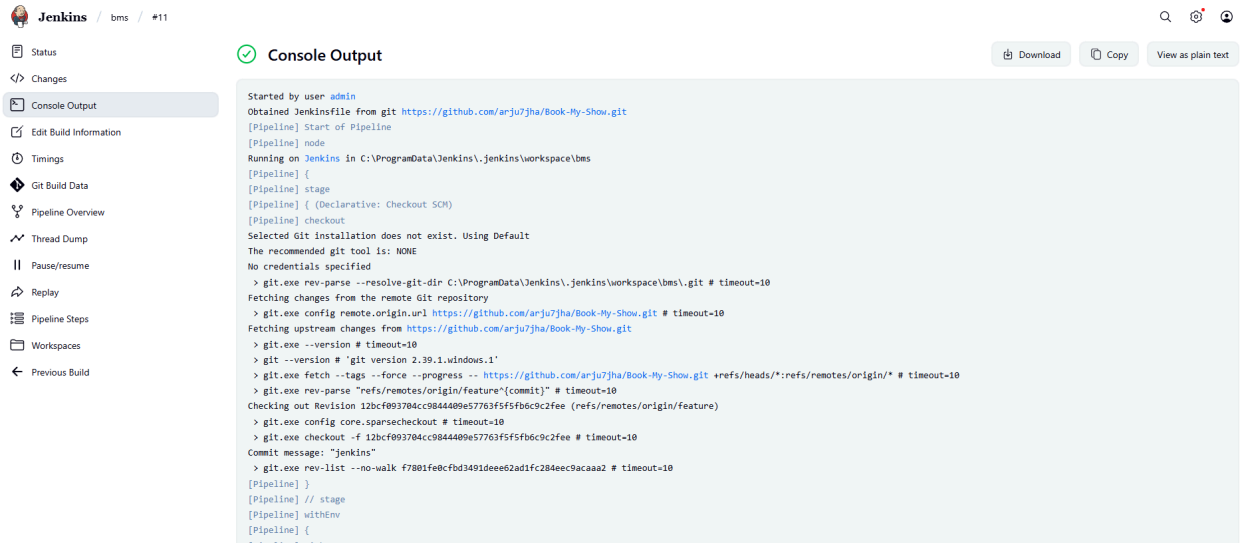
D). Enter the repo branch.

E). Provide the path of the jenkinsfile and click one apply and save .

F). Now build the pipeline .



Build Succeeded:



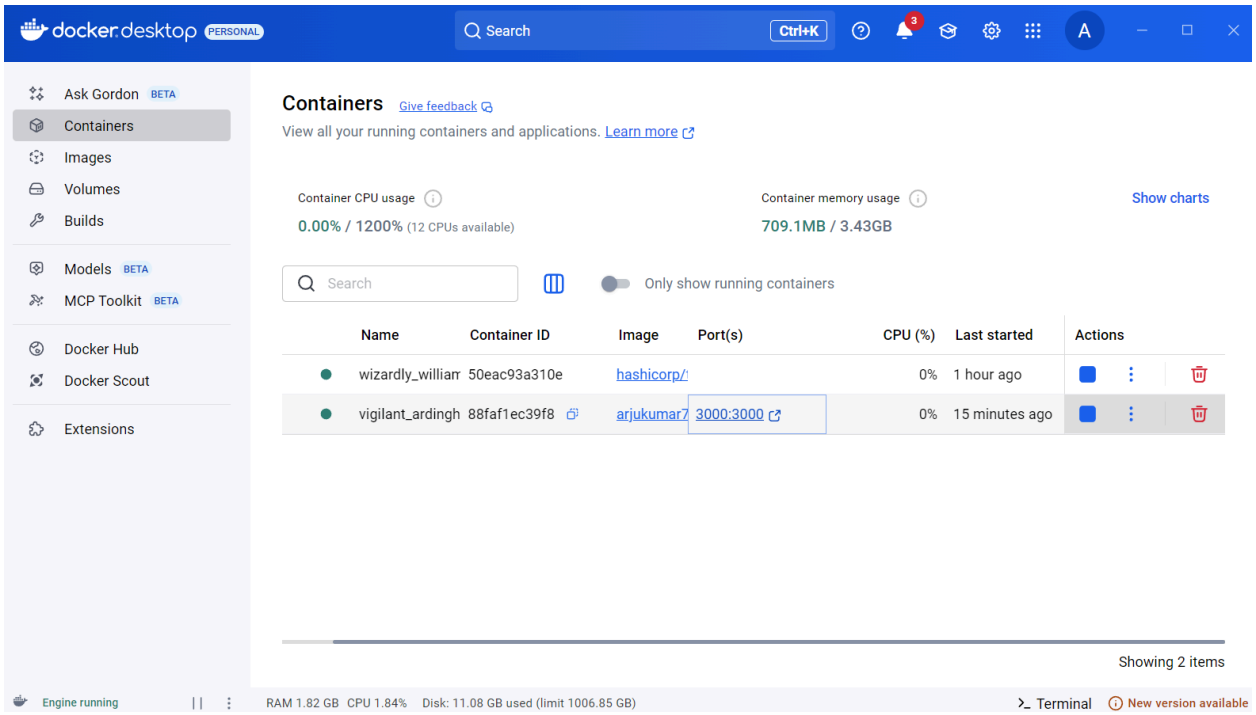
Step 3: To run container locally and validate accessibility on port 3000.

→ Run the container locally after pushing the image, run the container locally to verify it works:

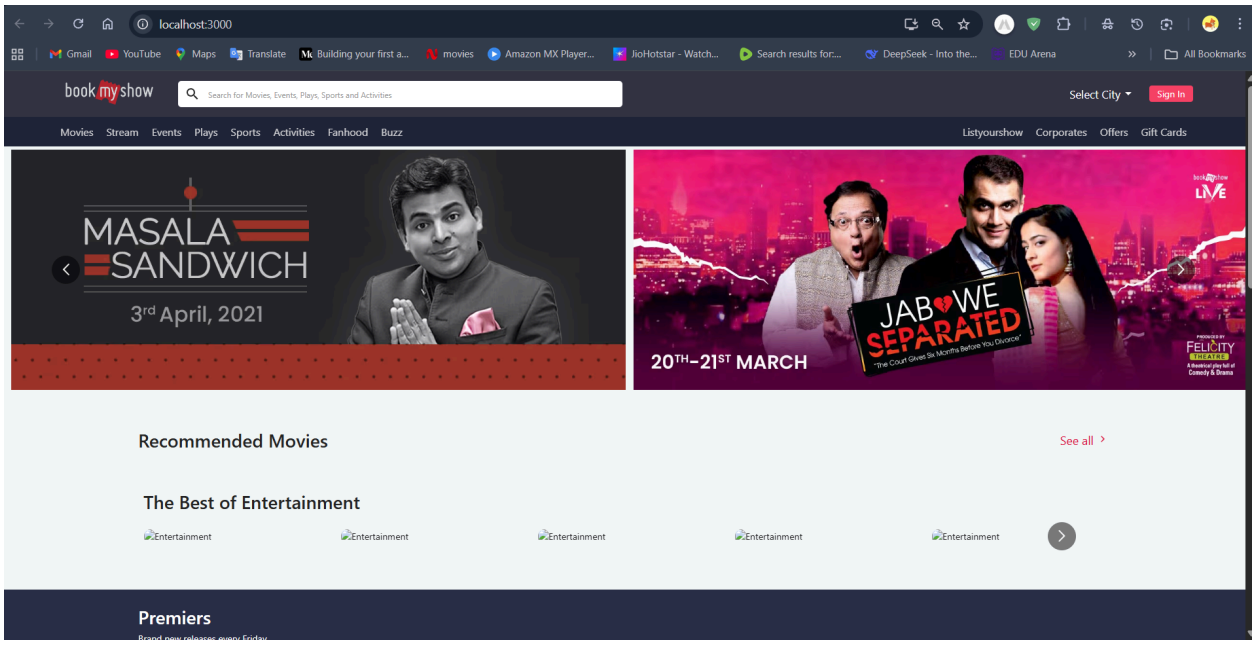
A) use command: **“docker run -d -p 3000:3000 arjukumar7/bookmyshow-app:v1”**

→ Then, open your browser and visit: “ http://localhost:3000 “

→ Below attached the BMS application running proof on port 3000.

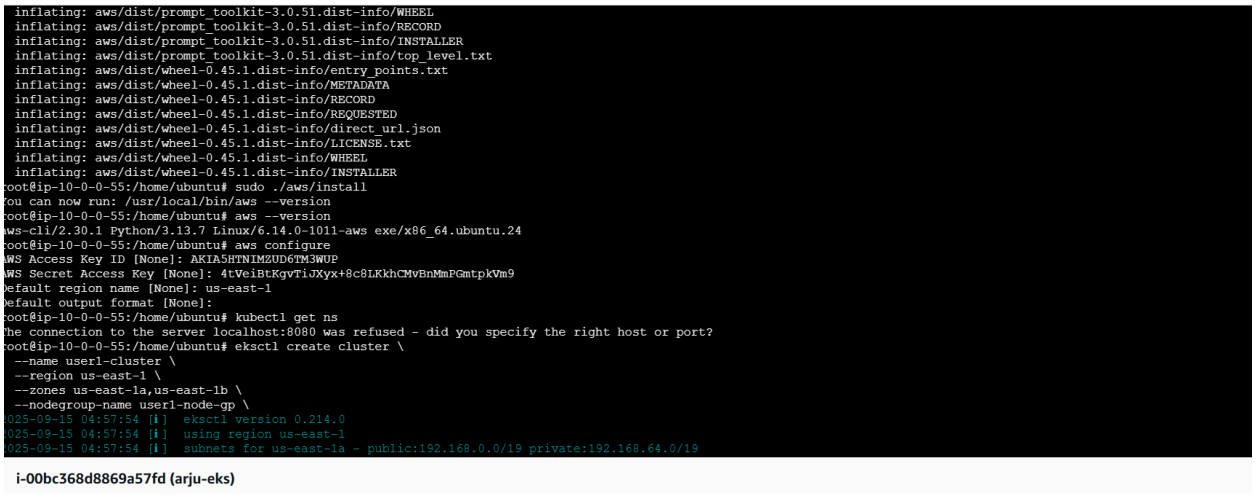


→ Here you can see the BMS application running successfully locally on port 3000.

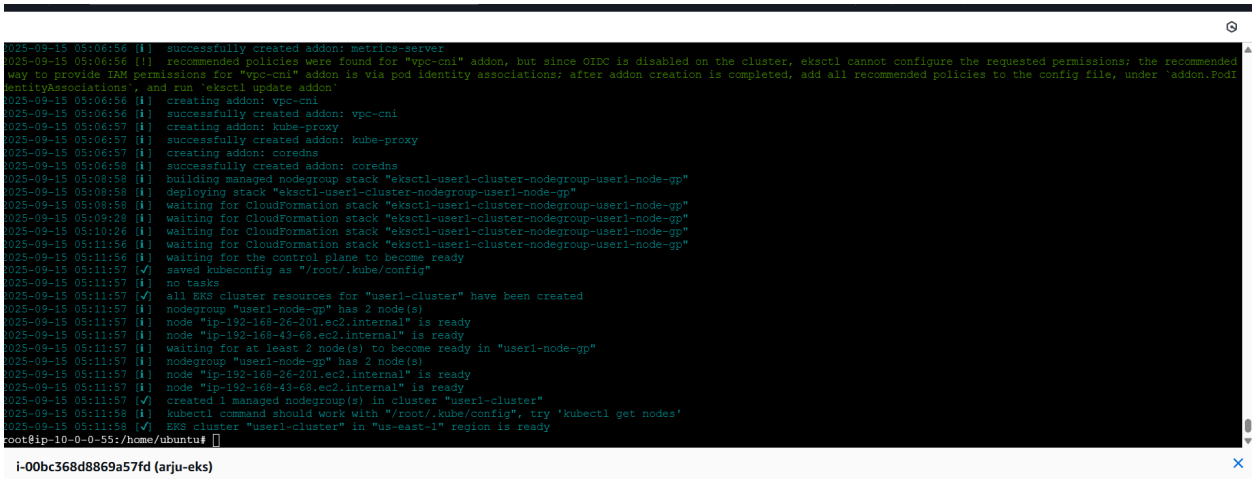


Stage 5: Kubernetes Deployment (EKS)

Step 1: Install all the necessary tools and setup the configuration to run the Cluster



→ Successfully set up the EKS cluster .



Step 2: Created two files of Kubernetes manifests (`deployment.yaml`, `service.yaml`):

→ **deployment.yaml** – In this file defines how many replicas (pods) i want, what container image to use, and other configurations.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bookmyshow-deployment
  namespace: arju
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bookmyshow
  template:
    metadata:
      labels:
        app: bookmyshow
    spec:
      containers:
      - name: bookmyshow
        image: arjukumar7/bookmyshow-app:v1
        ports:
        - containerPort: 3000
~
~
~
~
~
~
~
~
~
~
"deployment.yaml" 20L, 373B
```

i-00bc368d8869a57fd (arju-eks)

→ **service.yaml** – In this file exposes the application to use by using the NodePort or LoadBalancer.

```
apiVersion: v1
kind: Service
metadata:
  name: bookmyshow-service
  namespace: arju
spec:
  type: LoadBalancer
  selector:
    app: bookmyshow
  ports:
    - protocol: TCP
      port: 80
      targetPort: 3000
~
~
~
~
~
~
~
```

Step 3: To deploy on EKS Cluster used the commands “**kubectl apply**” to deploy the app, and to create the pods and service inside the EKS cluster by using below commands:.

→ **“ kubectl apply -f deployment.yaml ”**

→ **“ kubectl apply -f service.yaml “**

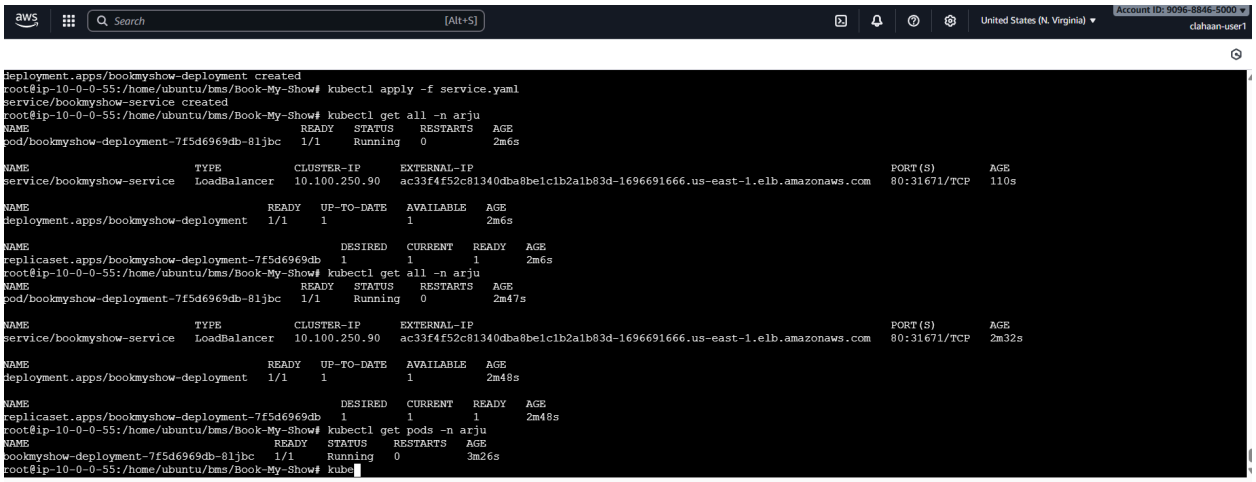
```
remote: Enumerating objects: 188, done.
remote: Counting objects: 100% (188/188), done.
remote: Compressing objects: 100% (149/149), done.
remote: Total 188 (delta 46), reused 168 (delta 33), pack-reused 0 (from 0)
Receiving objects: 100% (188/188), 1.60 MiB | 30.83 MiB/s, done.
Resolving deltas: 100% (46/46), done.
root@ip-10-0-0-55:/home/ubuntu/bms# cd ..
root@ip-10-0-0-55:/home/ubuntu# kubectl create ns arju
namespace/arju created
root@ip-10-0-0-55:/home/ubuntu# ls
aws  awscliv2.zip  bms
root@ip-10-0-0-55:/home/ubuntu# kubectl get  ns
NAME              STATUS   AGE
arju              Active   20s
default           Active   25m
kube-node-lease   Active   25m
kube-public       Active   25m
kube-system       Active   25m
root@ip-10-0-0-55:/home/ubuntu# cd bms/
root@ip-10-0-0-55:/home/ubuntu/bms# ls
Book-My-Show
root@ip-10-0-0-55:/home/ubuntu/bms# cd Book-My-Show/
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# ls
BMS-Document.txt  Jenkinsfile  Jenkinsfile1  Jenkinsfile2  README.md  bookmyshow-app  deployment.yaml  deployment.yml  service.yaml  service.yml
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# vi deployment.yaml
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# vi service.yaml
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl apply -f deployment.yaml
deployment.apps/bookmyshow-deployment created
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl apply -f service.yaml
service/bookmyshow-service created
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show#
```

i-00bc368d8869a57fd (arju-eks)

Step 4: Exposed service using a NodePort or LoadBalancer in **service.yaml** to make the app accessible.

→ **NodePort:** Opens a specific port on each node.

→ **LoadBalancer:** Creates an AWS ELB (Elastic Load Balancer) to route traffic automatically.



Step 5: To validate Deployment to make sure that everything is running correctly by using:

→ **“kubectl get pods”** : Shows a **list of all pods** running in cluster

→ **“kubectl get svc”** : Shows a **list of all services** running in a cluster and gives us the external IP or NodePort to access the app.

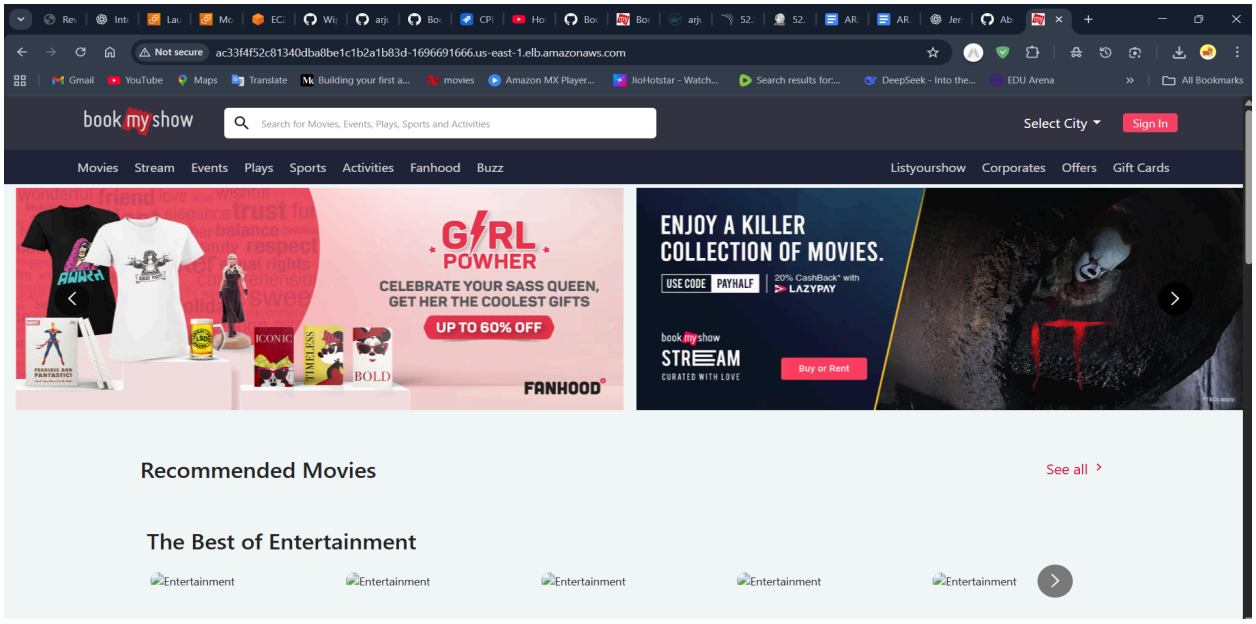
```
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-pxn9z (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       True
  ContainersReady             True
  PodScheduled                True
Volumes:
  kube-api-access-pxn9z:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
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  14m   default-scheduler  Successfully assigned arju/bookmyshow-deployment-7f5d6969db-81jbc to ip-192-168-26-201.ec2.internal
  Normal  Pulling    14m   kubelet            Pulling image "arjukumar7/bookmyshow-app:v1"
  Normal  Pulled     14m   kubelet            Successfully pulled image "arjukumar7/bookmyshow-app:v1" in 34.353s (34.353s including waiting). Image size: 575160739 bytes.
  Normal  Created    14m   kubelet            Created container: bookmyshow
  Normal  Started    14m   kubelet            Started container bookmyshow
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show#
```

```
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl get pods -n arju
NAME                                    READY   STATUS    RESTARTS   AGE
bookmyshow-deployment-7f5d6969db-81jbc   1/1     Running   0          11m
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show#
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show#
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl get svc -n arju
NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
bookmyshow-service   LoadBalancer   10.100.250.90   ac33f4f52c81340dba8be1c1b2a1b83d-1696691666.us-east-1.elb.amazonaws.com   80:31671/TCP   12m
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl get all -n arju
NAME                                    READY   STATUS    RESTARTS   AGE
pod/bookmyshow-deployment-7f5d6969db-81jbc   1/1     Running   0          14m

NAME                         TYPE           CLUSTER-IP      EXTERNAL-IP                                                                PORT(S)        AGE
service/bookmyshow-service   LoadBalancer   10.100.250.90   ac33f4f52c81340dba8be1c1b2a1b83d-1696691666.us-east-1.elb.amazonaws.com   80:31671/TCP   13m

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/bookmyshow-deployment   1/1     1            1           14m

NAME                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/bookmyshow-deployment-7f5d6969db   1         1         1       14m
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show#
root@ip-10-0-0-55:/home/ubuntu/bms/Book-My-Show# kubectl describe pod bookmyshow-deployment-7f5d6969db-81jbc -n arju
Name:             bookmyshow-deployment-7f5d6969db-81jbc
Namespace:        arju
Priority:         0
Node:             ip-192-168-26-201.ec2.internal/192.168.26.201
Start Time:       Mon, 15 Sep 2025 05:35:47 +0000
Labels:           app=bookmyshow
                  pod-template-hash=7f5d6969db
Annotations:      <none>
Status:           Running
```



Stage 6: Monitoring & Observability

Step 1: Install Prometheus & Node Exporter

→ Deployed Prometheus on our Kubernetes cluster using Helm.

→ Installed Node Exporter on each node to collect system-level metrics (CPU, Memory, Disk, Network).

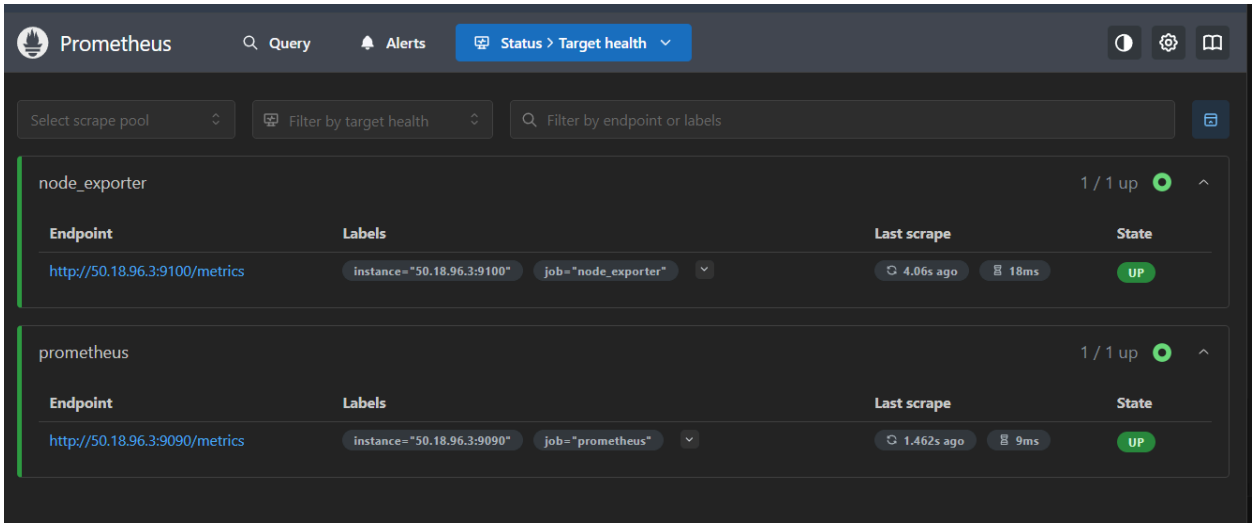
→Verified Prometheus was running by checking pods and services with:

→ **"kubectl get pods -n monitoring"**

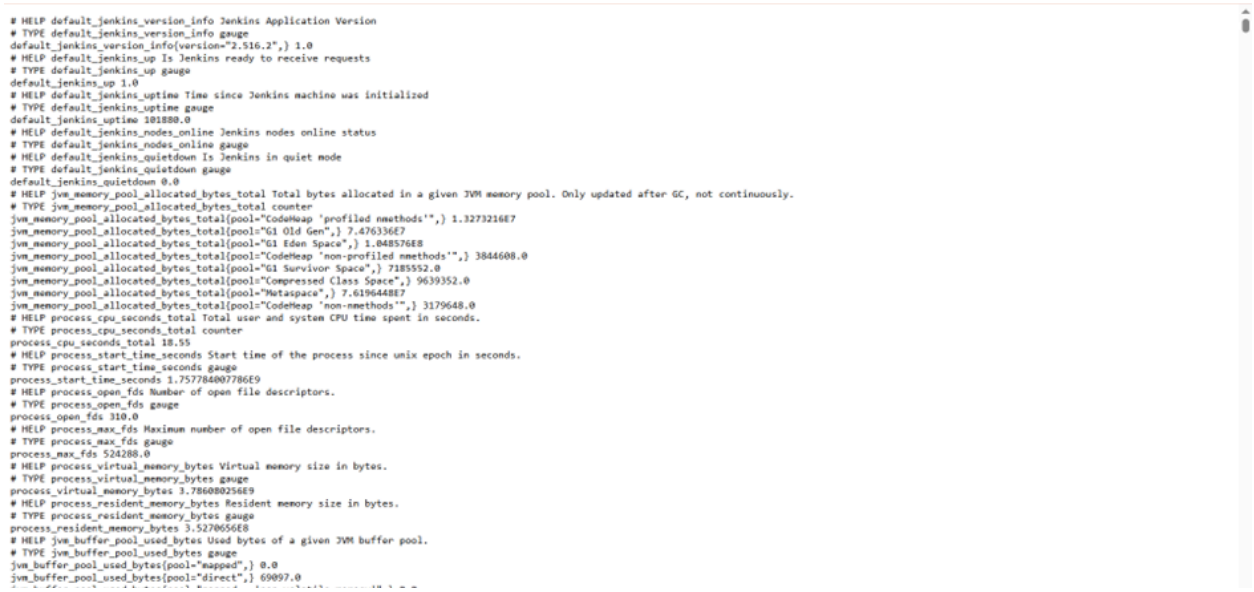
→ **"kubectl get svc -n monitoring"**

```
service/grafana-external exposed
root@ip-10-0-0-55:/home/ubuntu# kubectl get svc -n monitoring
NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)          AGE
grafana                               ClusterIP      10.100.98.219    <none>                                                                    80/TCP           9m20s
grafana-external                      LoadBalancer   10.100.190.221   a1fc1f3f66167400886e7eec77151ab2-444174145.us-east-1.elb.amazonaws.com    80:31875/TCP     16s
prometheus-alertmanager               ClusterIP      10.100.62.168    <none>                                                                    9093/TCP         23m
prometheus-alertmanager-headless      ClusterIP      None             <none>                                                                    9093/TCP         23m
prometheus-kube-state-metrics         ClusterIP      10.100.196.207   <none>                                                                    8080/TCP         23m
prometheus-prometheus-node-exporter   ClusterIP      10.100.197.246   <none>                                                                    9100/TCP         23m
prometheus-prometheus-pushgateway     ClusterIP      10.100.235.184   <none>                                                                    9091/TCP         23m
prometheus-server                     ClusterIP      10.100.206.230   <none>                                                                    80/TCP           23m
root@ip-10-0-0-55:/home/ubuntu# kubectl expose deployment prometheus-server \
  --type=LoadBalancer \
  --name=prometheus-external \
  --port=80 \
  --target-port=9090 \
  -n monitoring
service/prometheus-external exposed
root@ip-10-0-0-55:/home/ubuntu# kubectl get svc -n monitoring
NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)          AGE
grafana                               ClusterIP      10.100.98.219    <none>                                                                    80/TCP           10m
grafana-external                      LoadBalancer   10.100.190.221   a1fc1f3f66167400886e7eec77151ab2-444174145.us-east-1.elb.amazonaws.com    80:31875/TCP     95s
prometheus-alertmanager               ClusterIP      10.100.62.168    <none>                                                                    9093/TCP         24m
prometheus-alertmanager-headless      ClusterIP      None             <none>                                                                    9093/TCP         24m
prometheus-external                   LoadBalancer   10.100.120.3     af76b848369f941139b820e6447e5ab9-750060204.us-east-1.elb.amazonaws.com    80:30643/TCP     5s
prometheus-kube-state-metrics         ClusterIP      10.100.196.207   <none>                                                                    8080/TCP         24m
prometheus-prometheus-node-exporter   ClusterIP      10.100.197.246   <none>                                                                    9100/TCP         24m
prometheus-prometheus-pushgateway     ClusterIP      10.100.235.184   <none>                                                                    9091/TCP         24m
prometheus-server                     ClusterIP      10.100.206.230   <none>                                                                    80/TCP           24m
root@ip-10-0-0-55:/home/ubuntu# ^C
root@ip-10-0-0-55:/home/ubuntu# kubectl expose deployment grafana \
```

Step 2: Enabled Jenkins Prometheus plugin to expose metrics at /prometheus.

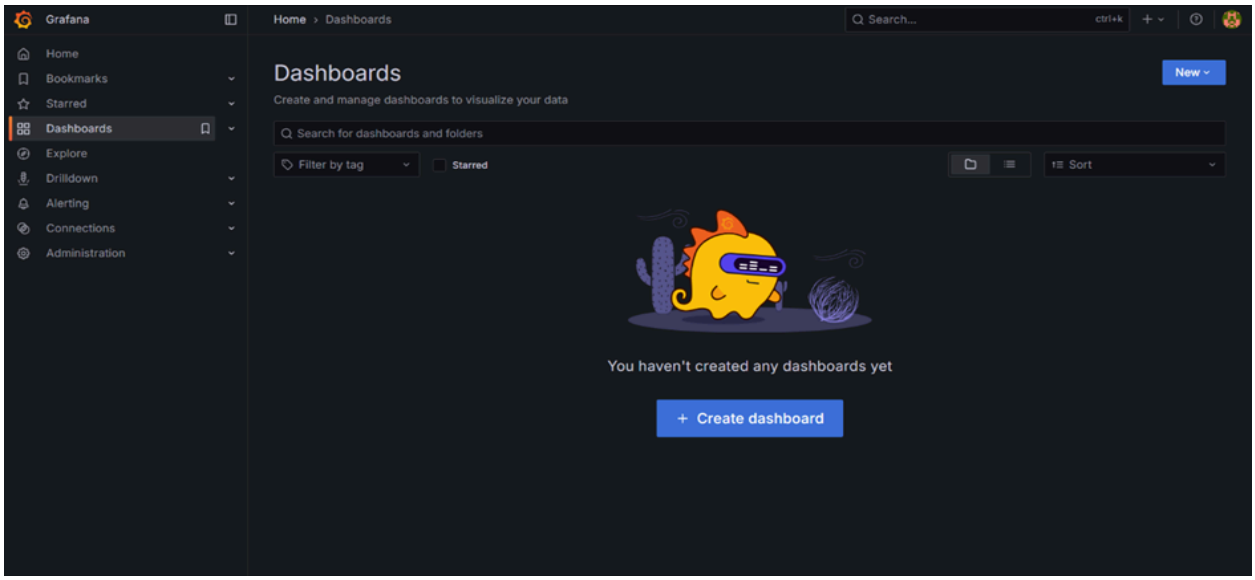


Step 3: Verified that Prometheus was successfully gathering metrics data Source from the application, Node Exporter, and Jenkins, confirming that the monitoring stack was fully functional and ready to be integrated with Grafana.

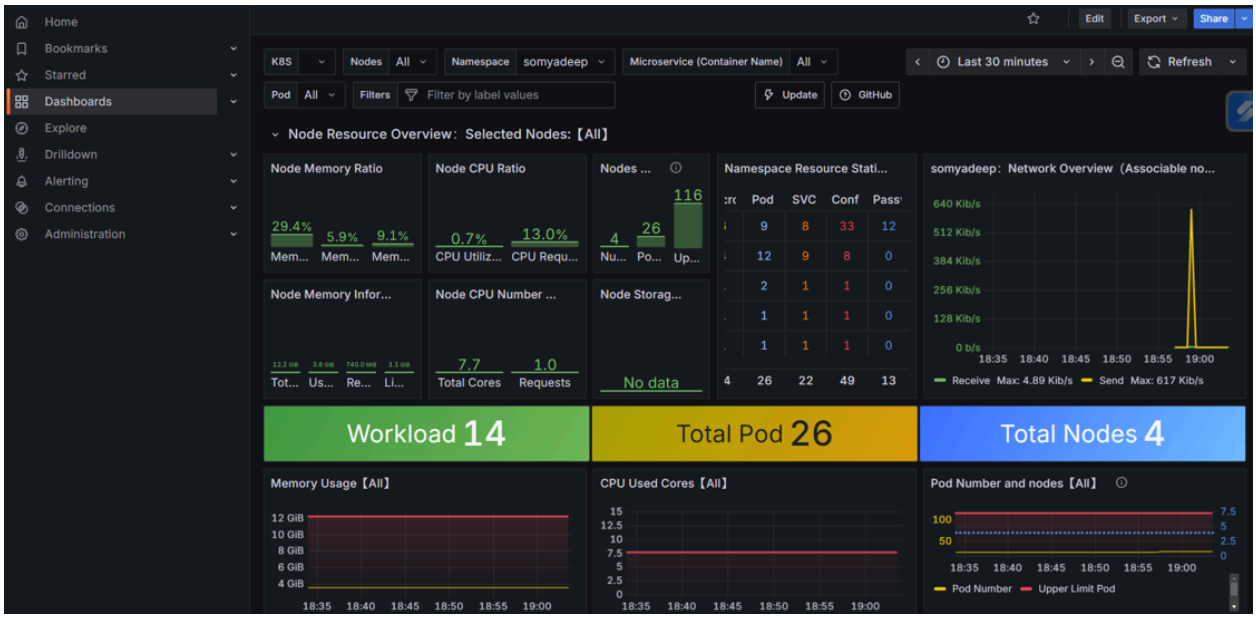


```
# HELP default_jenkins_version_info Jenkins Application Version
# TYPE default_jenkins_version_info gauge
default_jenkins_version_info{version="2.516.2",} 1.0
# HELP default_jenkins_up Is Jenkins ready to receive requests
# TYPE default_jenkins_up gauge
default_jenkins_up 1.0
# HELP default_jenkins_uptime Time since Jenkins machine was initialized
# TYPE default_jenkins_uptime gauge
default_jenkins_uptime 101880.0
# HELP default_jenkins_nodes_online Jenkins nodes online status
# TYPE default_jenkins_nodes_online gauge
# HELP default_jenkins_quietdown Is Jenkins in quiet mode
# TYPE default_jenkins_quietdown gauge
default_jenkins_quietdown 0.0
# HELP jvm_memory_pool_allocated_bytes_total Total bytes allocated in a given JVM memory pool. Only updated after GC, not continuously.
# TYPE jvm_memory_pool_allocated_bytes_total counter
jvm_memory_pool_allocated_bytes_total{pool="CodeHeap 'profiled nmethods'",} 1.327321687
jvm_memory_pool_allocated_bytes_total{pool="G1 Old Gen",} 7.476336E7
jvm_memory_pool_allocated_bytes_total{pool="G1 Eden Space",} 1.048576E8
jvm_memory_pool_allocated_bytes_total{pool="CodeHeap 'non-profiled nmethods'",} 3844608.0
jvm_memory_pool_allocated_bytes_total{pool="G1 Survivor Space",} 7185552.0
jvm_memory_pool_allocated_bytes_total{pool="Compressed Class Space",} 9639352.0
jvm_memory_pool_allocated_bytes_total{pool="Metaspace",} 7.619644887
jvm_memory_pool_allocated_bytes_total{pool="CodeHeap 'non-nmethods'",} 3179648.0
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 18.55
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.757784007786E9
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 310.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 524288.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 3.786080256E9
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 3.5270656E8
# HELP jvm_buffer_pool_used_bytes Used bytes of a given JVM buffer pool.
# TYPE jvm_buffer_pool_used_bytes gauge
jvm_buffer_pool_used_bytes{pool="mapped",} 0.0
jvm_buffer_pool_used_bytes{pool="direct",} 69097.0
```

Step 3: Add dashboards for Node health and Jenkins performance.

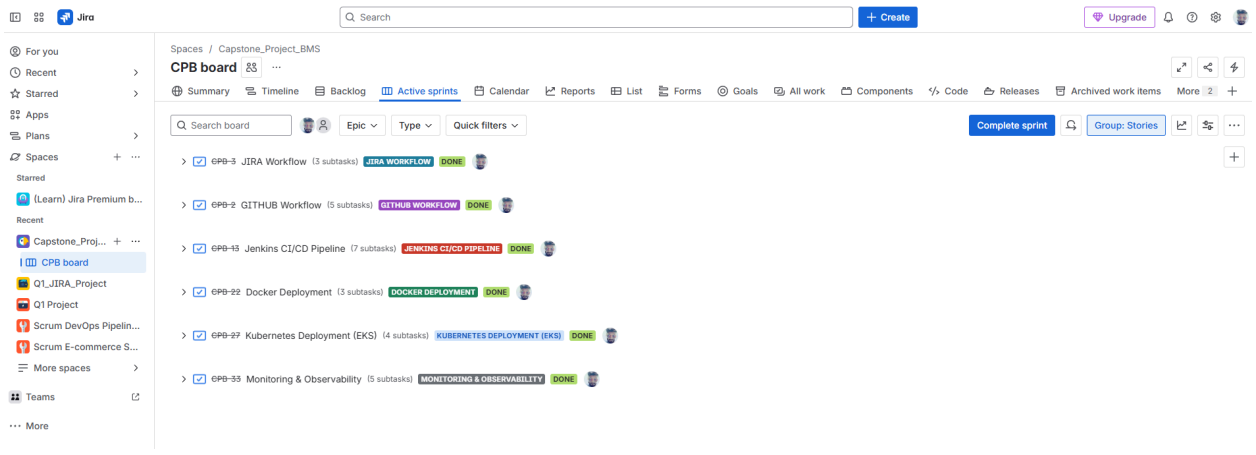


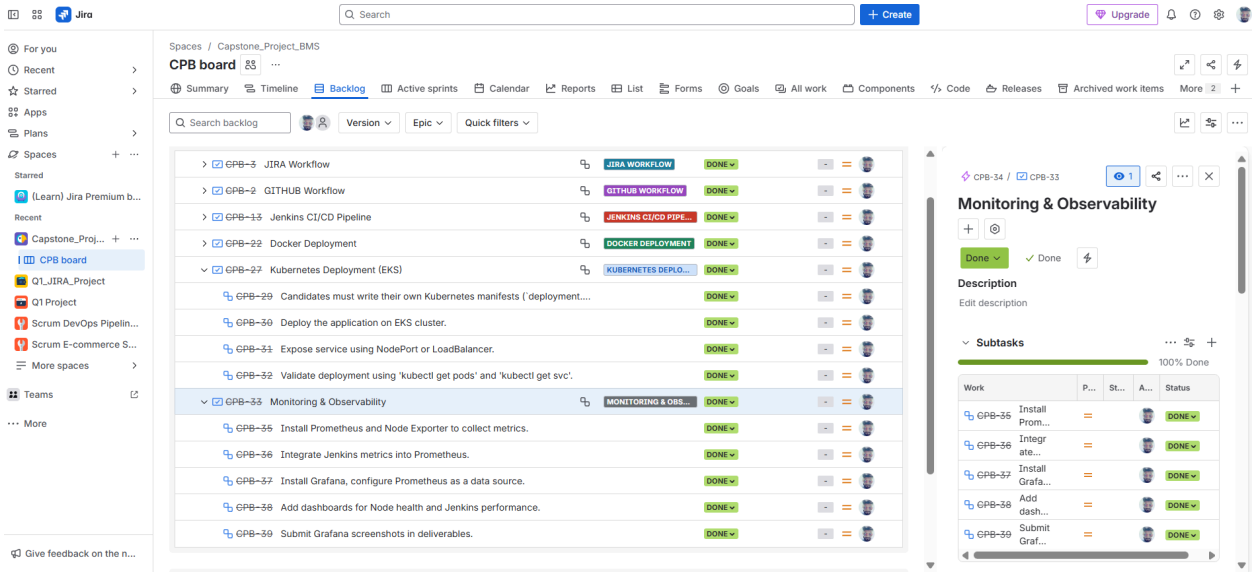
Step 4: Used Grafana Dashboard ID 15661, which is built for monitoring Linux system metrics through Prometheus Node Exporter. It offers a comprehensive, real-time view of system performance, helping you easily monitor resource usage and overall system health.

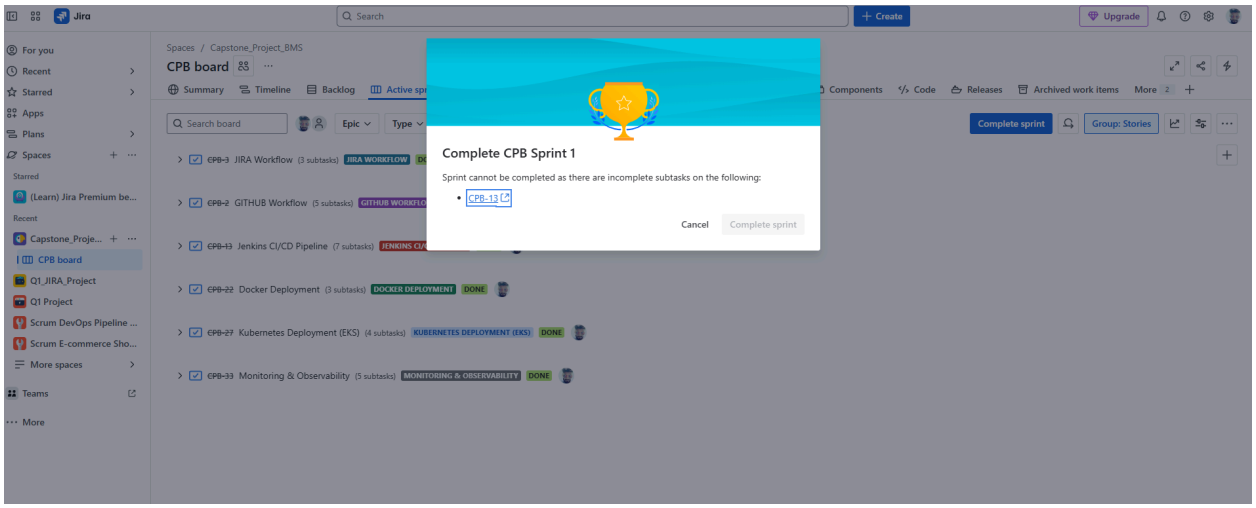


Jira Submission: Task completion proof

→ Completed all the assigned tasks and updated them in the jira dashboard also .




Jira
Search
Create
Upgrade
Spaces / Capstone_Project_BMS
CPB board
Summary
Timeline
Backlog
Active sprints
Calendar
Reports
List
Forms
Goals
All work
Components
Code
Releases
Archived work items
More
Search backlog
Version
Epic
Quick filters
CPB-3 JIRA Workflow
CPB-2 GITHUB Workflow
CPB-13 Jenkins CI/CD Pipeline
CPB-22 Docker Deployment
CPB-27 Kubernetes Deployment (EKS)
CPB-29 Candidates must write their own Kubernetes manifests (`deployment....
CPB-30 Deploy the application on EKS cluster.
CPB-31 Expose service using NodePort or LoadBalancer.
CPB-32 Validate deployment using 'kubectl get pods' and 'kubectl get svc'.
CPB-33 Monitoring & Observability
CPB-35 Install Prometheus and Node Exporter to collect metrics.
CPB-36 Integrate Jenkins metrics into Prometheus.
CPB-37 Install Grafana, configure Prometheus as a data source.
CPB-38 Add dashboards for Node health and Jenkins performance.
CPB-39 Submit Grafana screenshots in deliverables.
Monitoring & Observability
Done
Description
Edit description
Subtasks
100% Done


Jira
Search
Create
Upgrade
Spaces / Capstone_Project_BMS
CPB board
Complete sprint
Group: Stories
Complete CPB Sprint 1
Sprint cannot be completed as there are incomplete subtasks on the following:
CPB-13
Cancel
Complete sprint