SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Team Details :
PRIYANSHU THAKUR (RA2211003011756)
ADITYA GUPTA (RA2211003011821)
RITESH MAHARA (RA2211003011937)

🚀 AI-Based Kubernetes Pod Failure Prediction Model

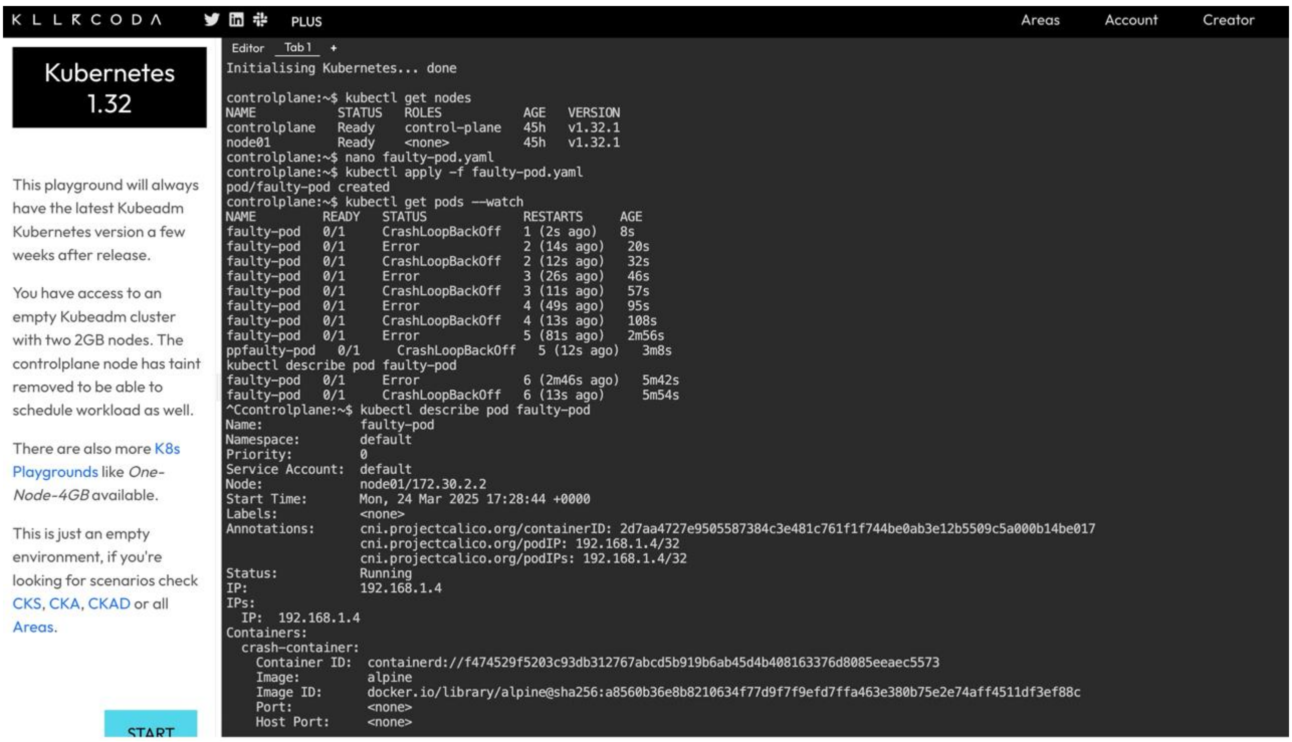
Problem Statement:

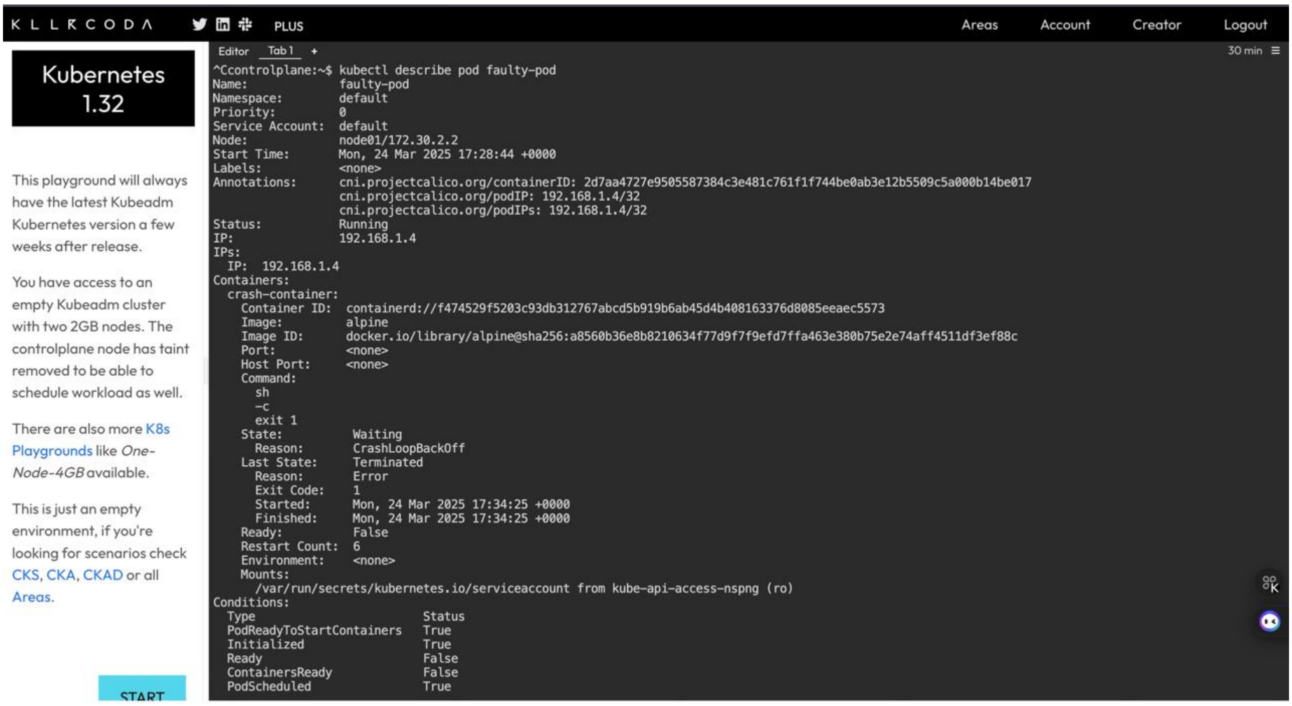
Kubernetes clusters often face issues like **pod failures**, which lead to service downtime and degraded system performance. Detecting these failures early is crucial for maintaining system stability.

Our Approach:

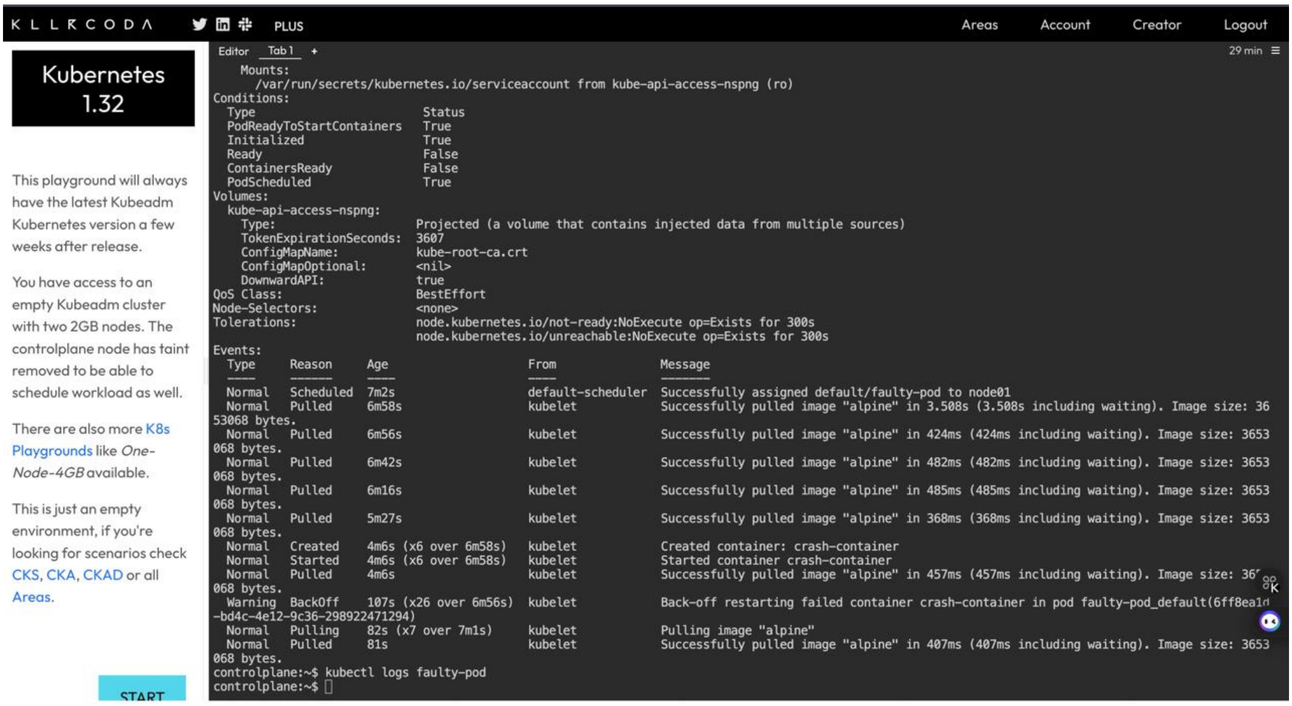
For this hackathon, our team focused on **predicting Kubernetes pod failures** using a Machine Learning model. After analyzing various possible issues, we found that **pod failure prediction**—especially detecting failures like `CrashLoopBackOff`—was the most feasible and impactful to solve within the timeframe.

STEP 1: Creation of a Pods failure for the visual representation






```
^Ccontrolplane:~$ kubectl describe pod faulty-pod
Name:             faulty-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             node01/172.30.2.2
Start Time:       Mon, 24 Mar 2025 17:28:44 +0000
Labels:           <none>
Annotations:      cni.projectcalico.org/containerID: 2d7aa4727e9505587384c3e481c761f1f744be0ab3e12b5509c5a000b14be017
                  cni.projectcalico.org/podIP: 192.168.1.4/32
                  cni.projectcalico.org/podIPs: 192.168.1.4/32
Status:           Running
IP:               192.168.1.4
IPs:
  IP:  192.168.1.4
Containers:
  crash-container:
    Container ID:  containerd://f474529f5203c93db312767abcd5b919b6ab45d4b408163376d8085eeaec5573
    Image:         alpine
    Image ID:      docker.io/library/alpine@sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c
    Port:          <none>
    Host Port:     <none>
    Command:
      sh
      -c
      exit 1
    State:          Waiting
      Reason:       CrashLoopBackOff
    Last State:     Terminated
      Reason:       Error
      Exit Code:    1
      Started:      Mon, 24 Mar 2025 17:34:25 +0000
      Finished:     Mon, 24 Mar 2025 17:34:25 +0000
    Ready:          False
    Restart Count:  6
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-nspng (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       False
  ContainersReady             False
  PodScheduled                True
```




```
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-nspng (ro)
Conditions:
  Type                        Status
  PodReadyToStartContainers   True
  Initialized                 True
  Ready                       False
  ContainersReady             False
  PodScheduled                True
Volumes:
  kube-api-access-nspng:
    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  7m2s                  default-scheduler  Successfully assigned default/faulty-pod to node01
  Normal   Pulled     6m58s                 kubelet            Successfully pulled image "alpine" in 3.508s (3.508s including waiting). Image size: 36
53068 bytes.
  Normal   Pulled     6m56s                 kubelet            Successfully pulled image "alpine" in 424ms (424ms including waiting). Image size: 3653
068 bytes.
  Normal   Pulled     6m42s                 kubelet            Successfully pulled image "alpine" in 482ms (482ms including waiting). Image size: 3653
068 bytes.
  Normal   Pulled     6m16s                 kubelet            Successfully pulled image "alpine" in 485ms (485ms including waiting). Image size: 3653
068 bytes.
  Normal   Pulled     5m27s                 kubelet            Successfully pulled image "alpine" in 368ms (368ms including waiting). Image size: 3653
068 bytes.
  Normal   Created    4m6s (x6 over 6m58s)  kubelet            Created container: crash-container
  Normal   Started    4m6s (x6 over 6m58s)  kubelet            Started container crash-container
  Normal   Pulled     4m6s                  kubelet            Successfully pulled image "alpine" in 457ms (457ms including waiting). Image size: 36
068 bytes.
  Warning  BackOff    107s (x26 over 6m56s)  kubelet           Back-off restarting failed container crash-container in pod faulty-pod_default(6ff8ea1d
-bd4c-4e12-9c36-298922471294)
  Normal   Pulling    82s (x7 over 7m1s)    kubelet            Pulling image "alpine"
  Normal   Pulled     81s                   kubelet            Successfully pulled image "alpine" in 407ms (407ms including waiting). Image size: 3653
068 bytes.
controlplane:~$ kubectl logs faulty-pod
controlplane:~$
```

STEP 2: Dataset Creation
We generated a dataset by collecting Kubernetes pod metrics such as:

- **Timestamp**
- **Namespace**
- **Pod Status** (`Running`, `CrashLoopBackOff`, `Failed`, etc.)
- **Restart Count**
- **CPU Usage**
- **Memory Usage**

We simulated pod failures by:

- Increasing the restart count beyond safe thresholds
- Assigning failure statuses like `CrashLoopBackOff`
- Injecting resource-heavy conditions (high CPU and memory usage)

Feature Engineering:

We enhanced the dataset with additional features: ✅ **Hour of the Day**
✅ **Day of the Week**
✅ **Resource Stress Score** (based on CPU > 1.5 cores or Memory > 400MB)
✅ **Namespace Historical Risk** (calculated from past failure data per namespace)

STEP 3: Model Training

We used a **Random Forest Classifier** to train the model.
Data was split into training and testing sets, and scaled using **StandardScaler** for better model performance.

Training Highlights:

- Balanced class weights to handle imbalanced data
- High accuracy in predicting failure-prone pods
- Evaluated using **Confusion Matrix** and **Classification Report**

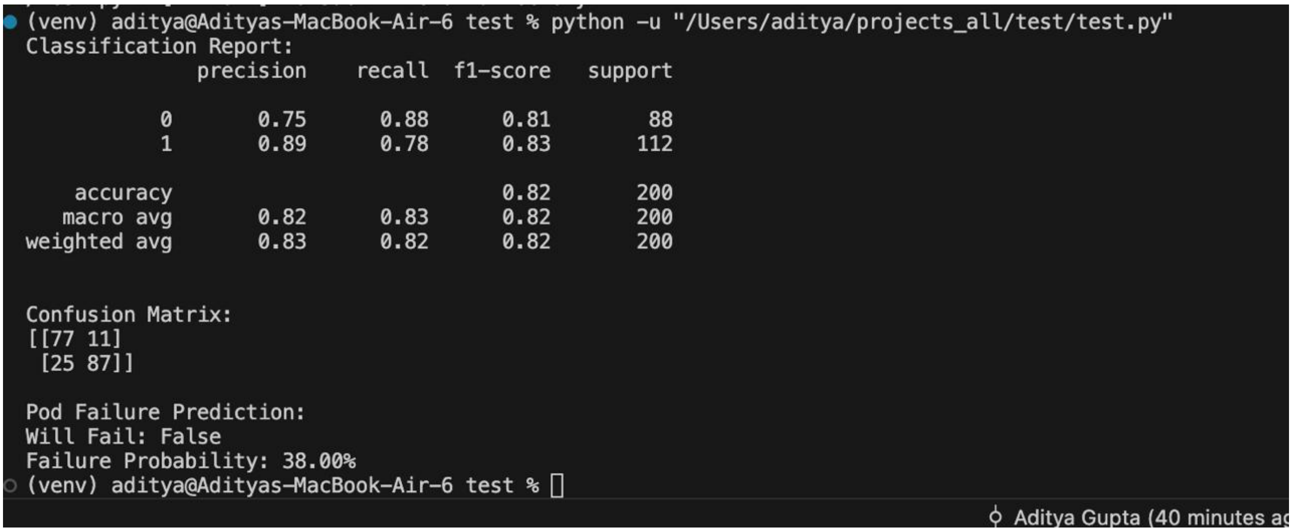
Prediction Process:

For any new pod, the model:

1. Takes real-time pod metrics (CPU, Memory, Restarts, etc.)
2. Computes the resource stress and historical namespace risk
3. Predicts whether the pod is likely to fail
4. Provides the **probability of failure**

Example Output:

✅ **Will Fail:** Yes
✅ **Failure Probability:** 85.34%



```
(venv) aditya@Adityas-MacBook-Air-6 test % python -u "/Users/aditya/projects_all/test/test.py"
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.88      0.81        88
           1       0.89      0.78      0.83       112

    accuracy                           0.82       200
   macro avg       0.82      0.83      0.82       200
weighted avg       0.83      0.82      0.82       200


Confusion Matrix:
[[77 11]
 [25 87]]

Pod Failure Prediction:
Will Fail: False
Failure Probability: 38.00%
(venv) aditya@Adityas-MacBook-Air-6 test %
                                                         Aditya Gupta (40 minutes a
```

Why This Solution is Effective:

✔ Combines **real-time metrics** with **historical data**
✔ Detects potential failures **before they happen**
✔ Helps DevOps teams **prevent downtime**
✔ Scalable and can be integrated into **Kubernetes monitoring tools**

Future Scope:

- Integrate with live Kubernetes clusters for real-time predictions
- Add alerting systems based on failure probability thresholds
- Extend the model to predict other Kubernetes issues like **node failures**, **OOM kills**, etc.