## EKS - Lab 1: Deploying Kubernetes Pods

ELI5 - Kubernetes

Kubernetes is a system for managing containerized applications across a cluster of machines. Here's a simple breakdown:

1. Containers: Imagine each part of an application (like the website, database, etc.) is put into a box called a container. Containers bundle the code and all its dependencies, so they run the same regardless of where they are deployed.
2. Cluster: Think of a cluster as a group of computers (nodes) working together. Kubernetes manages this group.
3. Orchestration: Kubernetes takes care of running these containers across the nodes in the cluster, making sure they are always up and running, even if some nodes fail. It also helps in scaling the application (adding more containers when needed) and distributing the load (ensuring no single node is overloaded).

OBJECTIVES

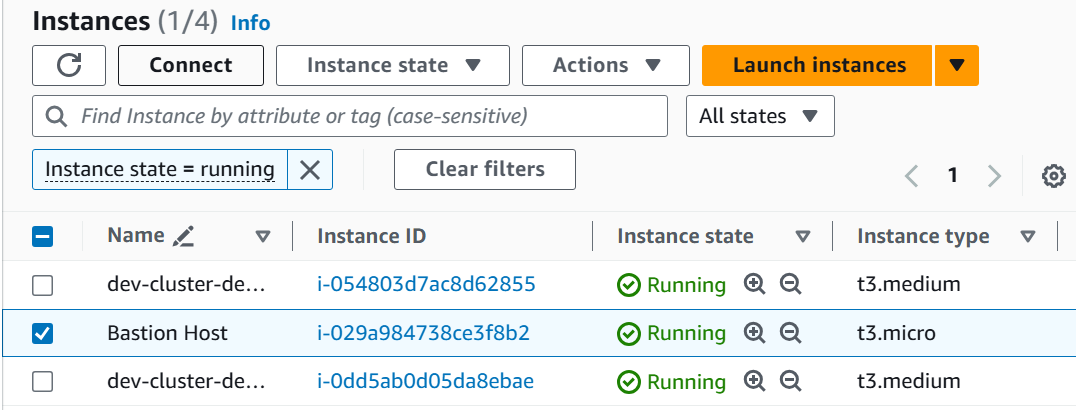
1. Create and deploy a Kubernetes application.
2. Build deployment, service, and namespace resources.
3. View resources in a namespace.
4. Execute commands in a pod.
5. Implement liveness and readiness probes.
6. Delete an application.

**A bastion host serves as a gateway to a private network from an external network.**

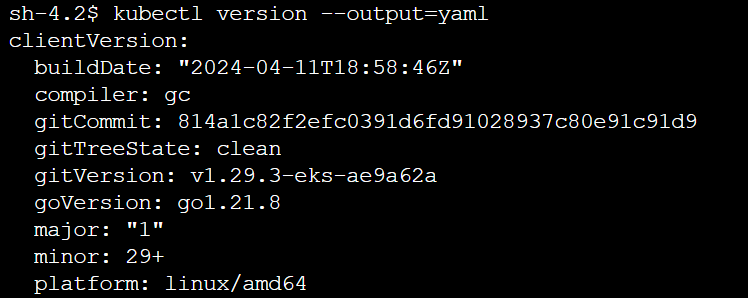
**Task 1: Deploy a Kubernetes application**

1.1 Open AWS and then in EC2 service, open running instances.

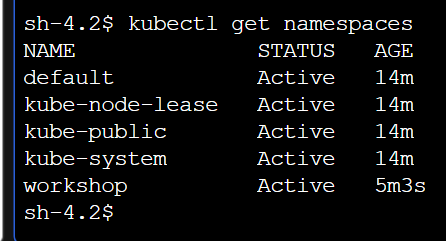
Choose the Bastion Host instance, and then choose Connect.

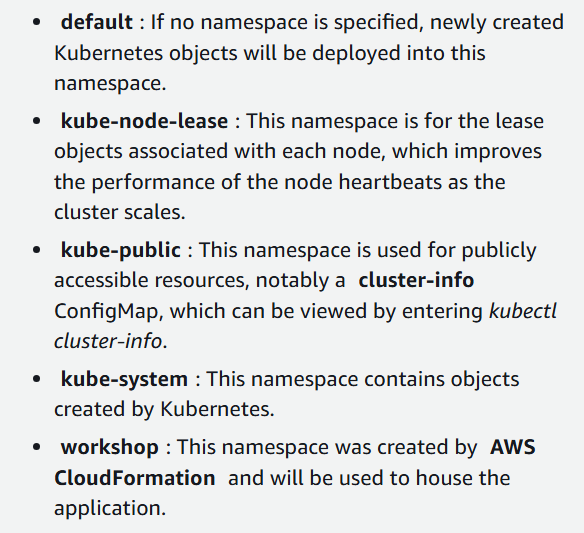


1.2 In the bastion host session, to verify that kubectl is installed, enter:



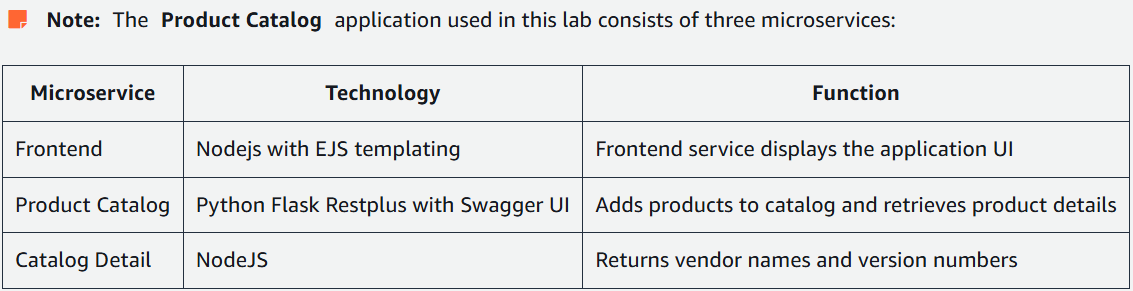
1.3 View the namespaces that have been created



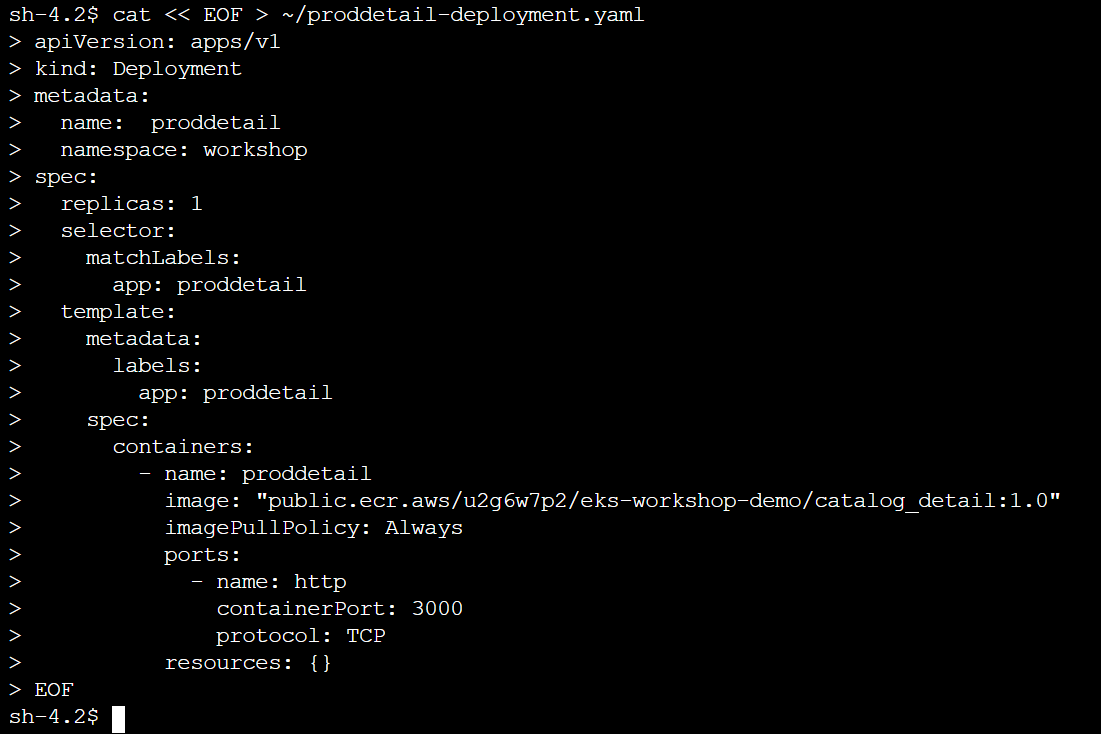


1.4 To view the resources currently deployed in the workshop namespace:





1.5 To deploy the Catalog Detail microservice, start by creating a manifest describing the frontend Deployment. To create the manifest, enter the following command:



The deployment manifest describes the desired state for a Kubernetes Deployment called proddetail, which creates a ReplicaSet with one pod using a container image from a public Amazon ECR repository. Key components include:

APIVersion: Uses v1 to create a Kubernetes Deployment object.

Metadata: Assigns a name to the deployment and optionally selects a namespace (default if not specified).

Kind: Specifies the type of Kubernetes object (Deployment).

Spec: Defines the state of the object.

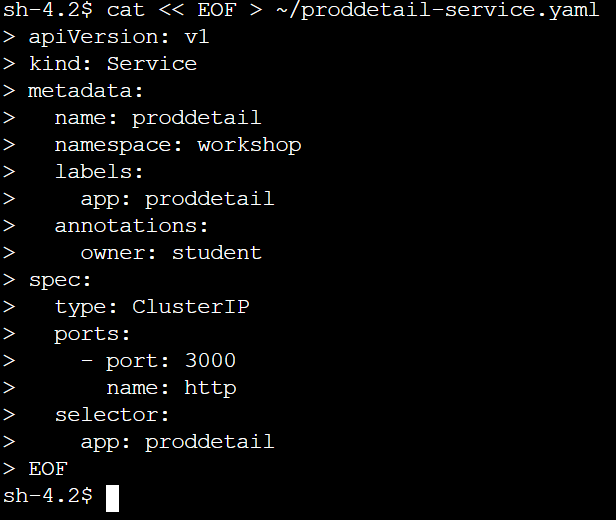
ReplicaSet: Ensures a specific number of pod instances are running (one replica in this case).

Selector: Matches pods with the label app: proddetail.

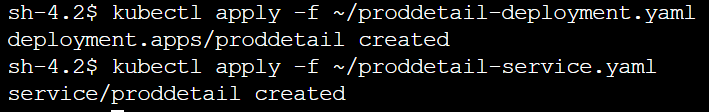
Template: Defines how new pods are created, specifying a container using a public Amazon ECR image, named proddetail, and opens port 3000 for HTTP traffic.

If the replicas field is set to 3, two additional pods will be created and maintained.

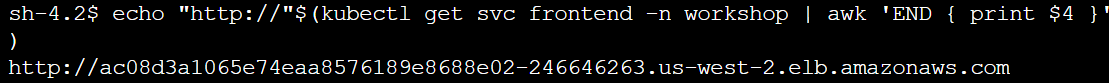
1.6 Create a second manifest declaring the state for your Product Detail service. To create the manifest, enter the following command:



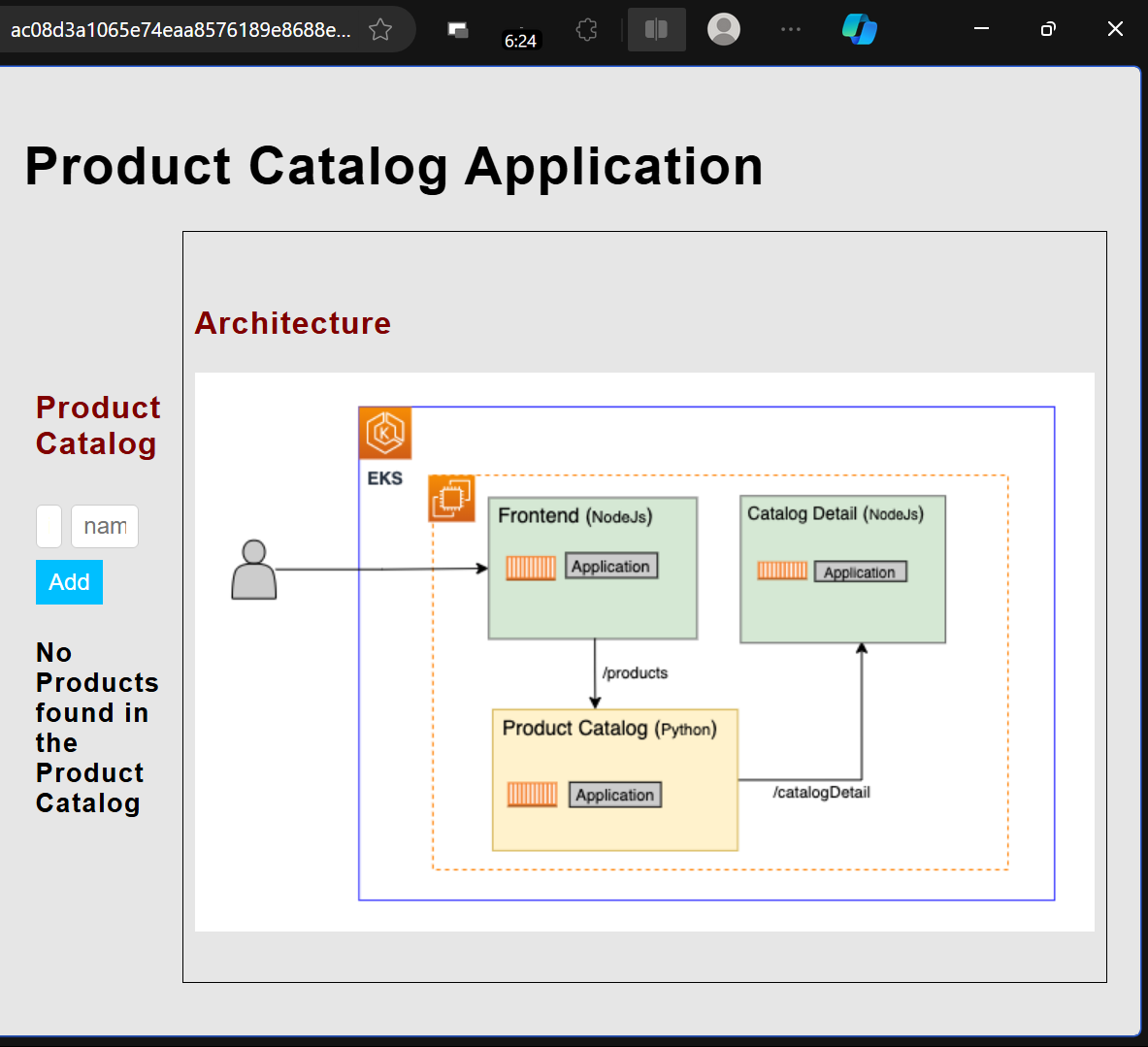
1.7 After creating manifests for the proddetail deployment and service, apply them to your cluster, using the following command:



1.8 to retrieve the URL pointing to the application frontend:

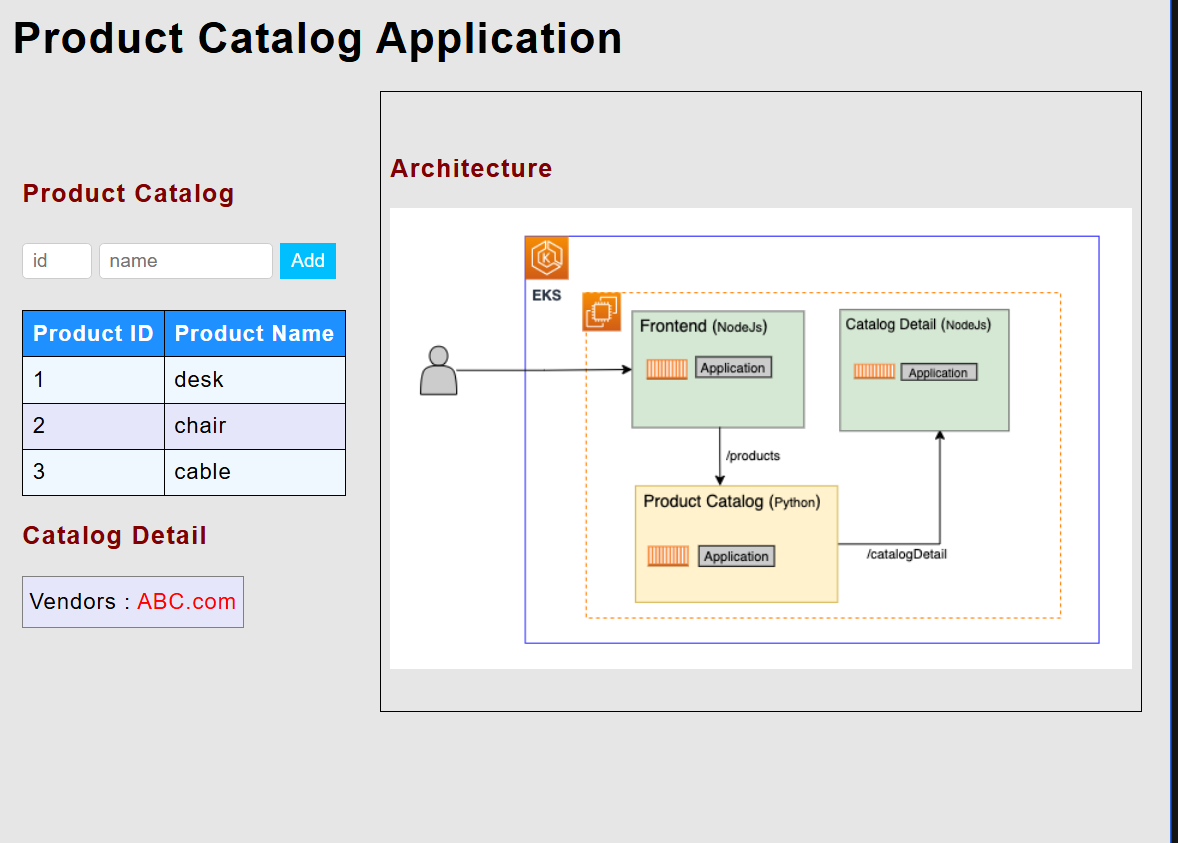


1.9 Open the url in a new tab



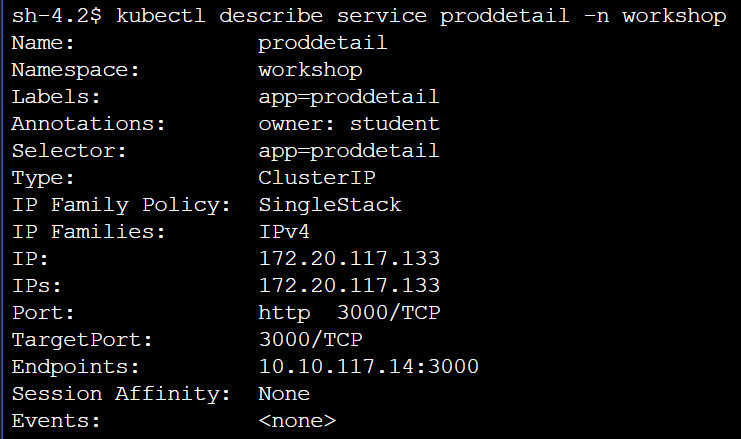
1.10 Enter the following as a product



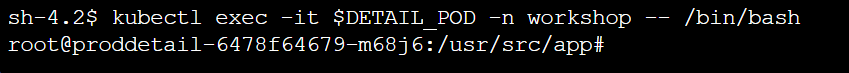


**Task 2: Explore the application resources**

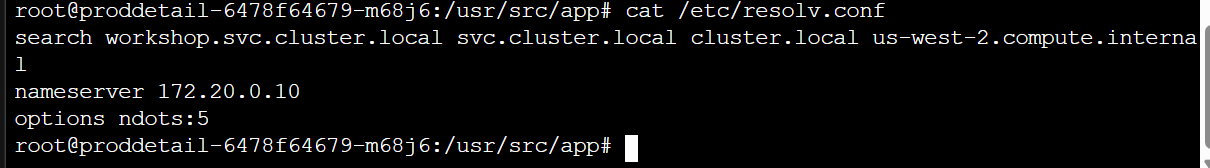
2.1 Return to CLI of session manager, and to view the details of the deployed service, enter the following command:



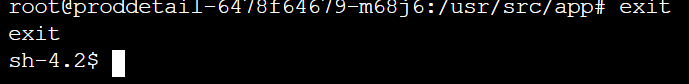
2.2 To connect to a bash shell in the pod, enter the following command:



2.3 To view the pod’s DNS configuration file, enter the following command:



2.4 exit from the pod using:



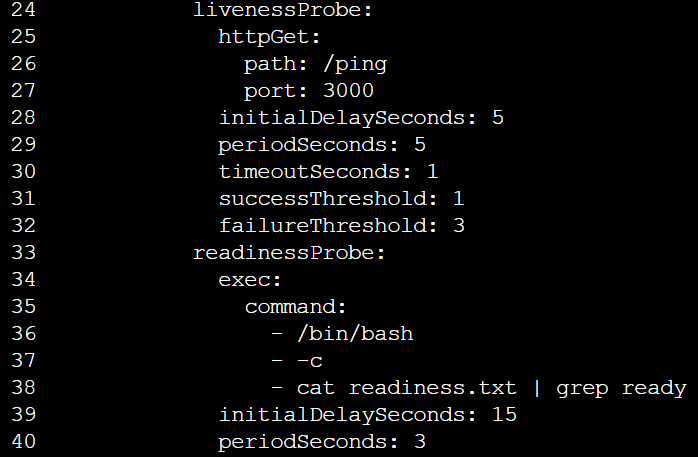
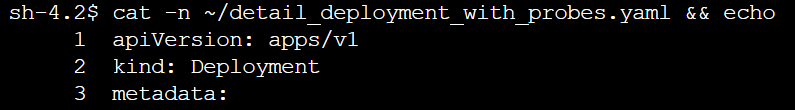
**You have successfully explored the Catalog Detail pod, connected to it, and run commands from inside of it.**

**Task 3: Implement liveness and readiness probes**

Kubernetes uses health checks to detect and remedy situations.

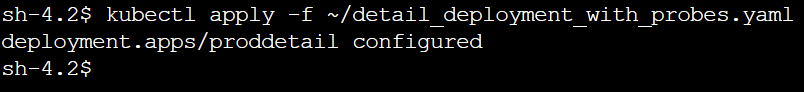
For example, you could use **liveness** probes to catch a deadlock or race condition, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs. Similarly, **readiness** probes help developers to ensure that their services do not send requests to pods before they are ready to start accepting traffic.

3.1 A deployment manifest including liveness and readiness probes has been saved to the BastionHost (by the lab itself). To view the manifest, enter the following command:



In this example, the kubelet will check the liveness probe every 5 seconds and the readiness probe every 3 seconds.

3.2 To update the Catalog Detail deployment to include liveness and readiness probes, enter the following command:

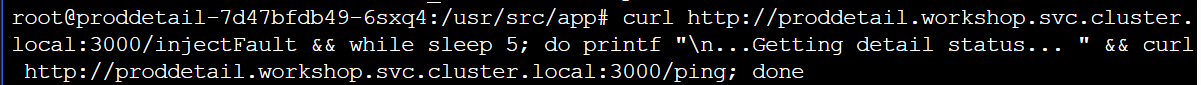


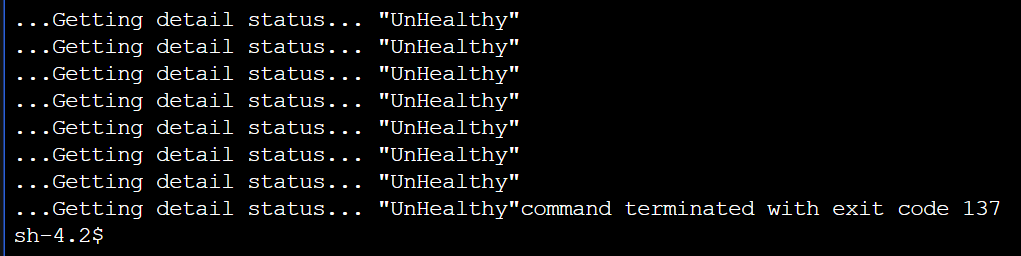
3.3 To become the root user inside the container:



3.4 To inject a fault and then repeatedly curl the endpoint to check its status, enter the following command:

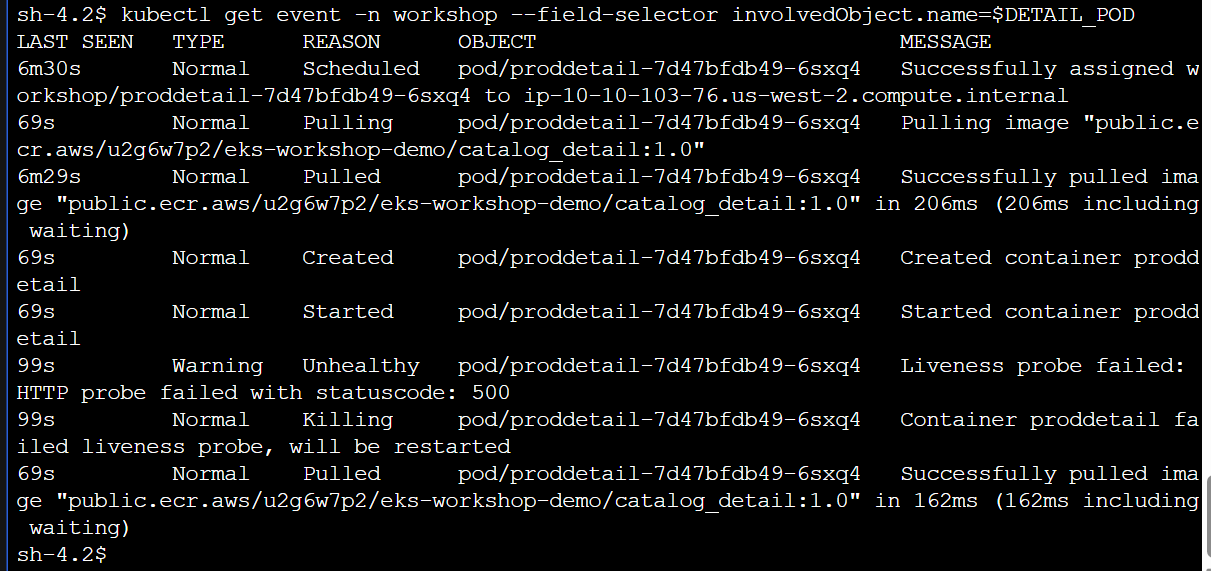
(this is done to fail the liveness probe)



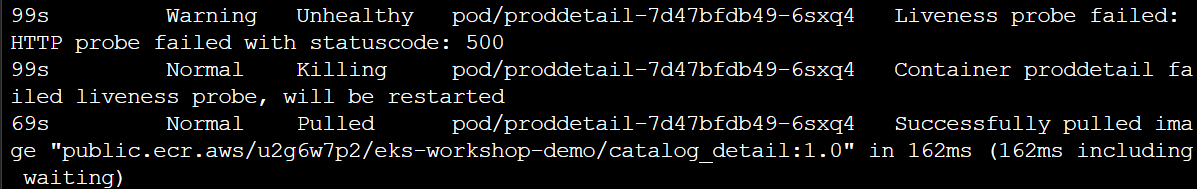


the command returns exit code 137, indicating that the container has been terminated.

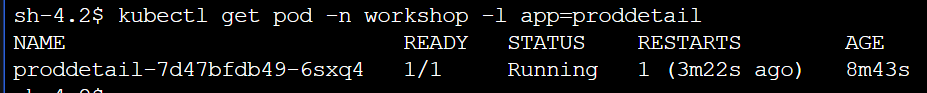
3.5 To retrieve the logs for the pod, enter the following command:



As you can see, about 69 seconds ago, the liveness probe for the pod failed with an HTTP 500 status code, so the pod was killed and restarted by Kubernetes. This caused the image to be pulled again upon restart.

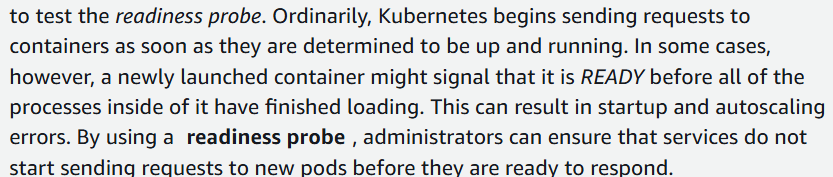


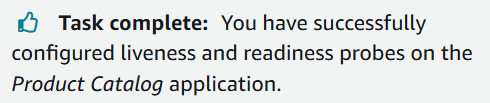
3.6 To confirm that the pod has successfully restarted, enter the following command:



We have tested the liveness probe.

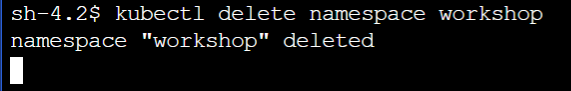
Similarly readiness probe can also be tested:





**Task 4: Delete the application**

4.1 To remove the sample service, deployment, pods, and namespace, enter the following command:

****

