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).

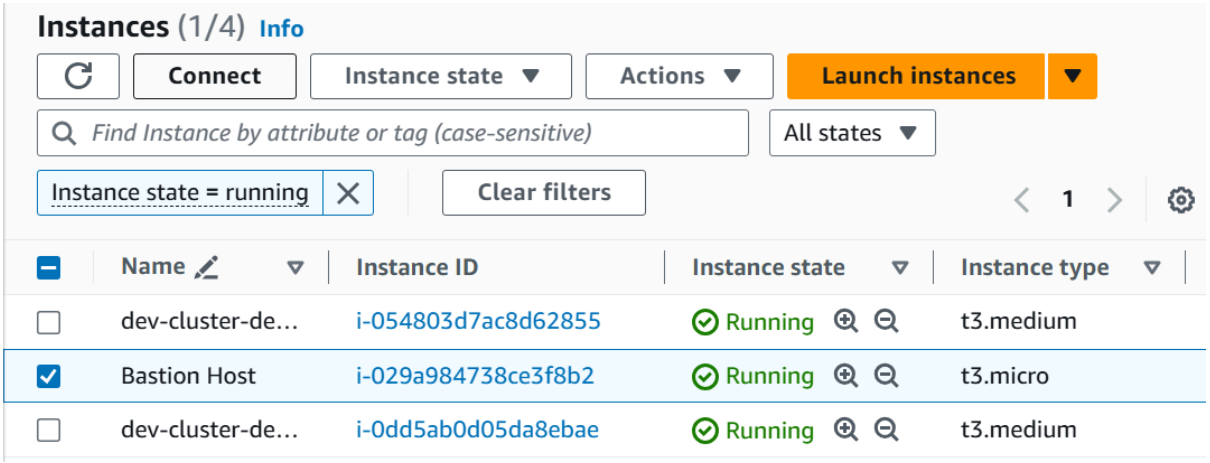
OBJECTIVES

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:

```
sh-4.2$ kubectl version --output=yaml
clientVersion:
  buildDate: "2024-04-11T18:58:46Z"
  compiler: gc
  gitCommit: 814a1c82f2efc0391d6fd91028937c80e91c91d9
  gitTreeState: clean
  gitVersion: v1.29.3-eks-ae9a62a
  goVersion: go1.21.8
  major: "1"
  minor: 29+
  platform: linux/amd64
```

1.3 View the namespaces that have been created

```
sh-4.2$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   14m
kube-node-lease   Active   14m
kube-public       Active   14m
kube-system       Active   14m
workshop          Active   5m3s
sh-4.2$
```

- **default** : If no namespace is specified, newly created Kubernetes objects will be deployed into this namespace.
- **kube-node-lease** : This namespace is for the lease objects associated with each node, which improves the performance of the node heartbeats as the cluster scales.
- **kube-public** : This namespace is used for publicly accessible resources, notably a **cluster-info** ConfigMap, which can be viewed by entering *kubectl cluster-info*.
- **kube-system** : This namespace contains objects created by Kubernetes.
- **workshop** : This namespace was created by **AWS CloudFormation** and will be used to house the application.

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

```
sh-4.2$ kubectl get deploy,svc,pod -n workshop
NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/frontend      1/1     1            1           6m58s
deployment.apps/prodcatalog   1/1     1            1           6m58s

NAME                  TYPE           CLUSTER-IP       EXTERNAL-IP
                                       PORT(S)         AGE
service/frontend      LoadBalancer   172.20.242.184   ac08d3a1065e74eaa8576189e8688e02-2
46646263.us-west-2.elb.amazonaws.com   80:30721/TCP    6m58s
service/prodcatalog   ClusterIP      172.20.100.135   <none>
                                       5000/TCP        6m58s

NAME                                READY   STATUS    RESTARTS   AGE
pod/frontend-778b579c5b-qj7wt       1/1     Running   0          6m58s
pod/prodcatalog-7fc4b697f6-mlg6j    1/1     Running   0          6m58s
sh-4.2$
```

Note: The **Product Catalog** application used in this lab consists of three microservices:

Microservice	Technology	Function
Frontend	Nodejs with EJS templating	Frontend service displays the application UI
Product Catalog	Python Flask Restplus with Swagger UI	Adds products to catalog and retrieves product details
Catalog Detail	NodeJS	Returns vendor names and version numbers

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:

```
sh-4.2$ cat << EOF > ~/proddetail-deployment.yaml
> apiVersion: apps/v1
> kind: Deployment
> metadata:
>   name:  proddetail
>   namespace: workshop
> spec:
>   replicas: 1
>   selector:
>     matchLabels:
>       app: proddetail
>   template:
>     metadata:
>       labels:
>         app: proddetail
>     spec:
>       containers:
>         - name: proddetail
>           image: "public.ecr.aws/u2g6w7p2/eks-workshop-demo/catalog_detail:1.0"
>           imagePullPolicy: Always
>           ports:
>             - name: http
>               containerPort: 3000
>               protocol: TCP
>           resources: {}
> EOF
sh-4.2$
```

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:

```
sh-4.2$ cat << EOF > ~/proddetail-service.yaml
> apiVersion: v1
> kind: Service
> metadata:
>   name: proddetail
>   namespace: workshop
>   labels:
>     app: proddetail
>   annotations:
>     owner: student
> spec:
>   type: ClusterIP
>   ports:
>     - port: 3000
>       name: http
>   selector:
>     app: proddetail
> EOF
sh-4.2$
```

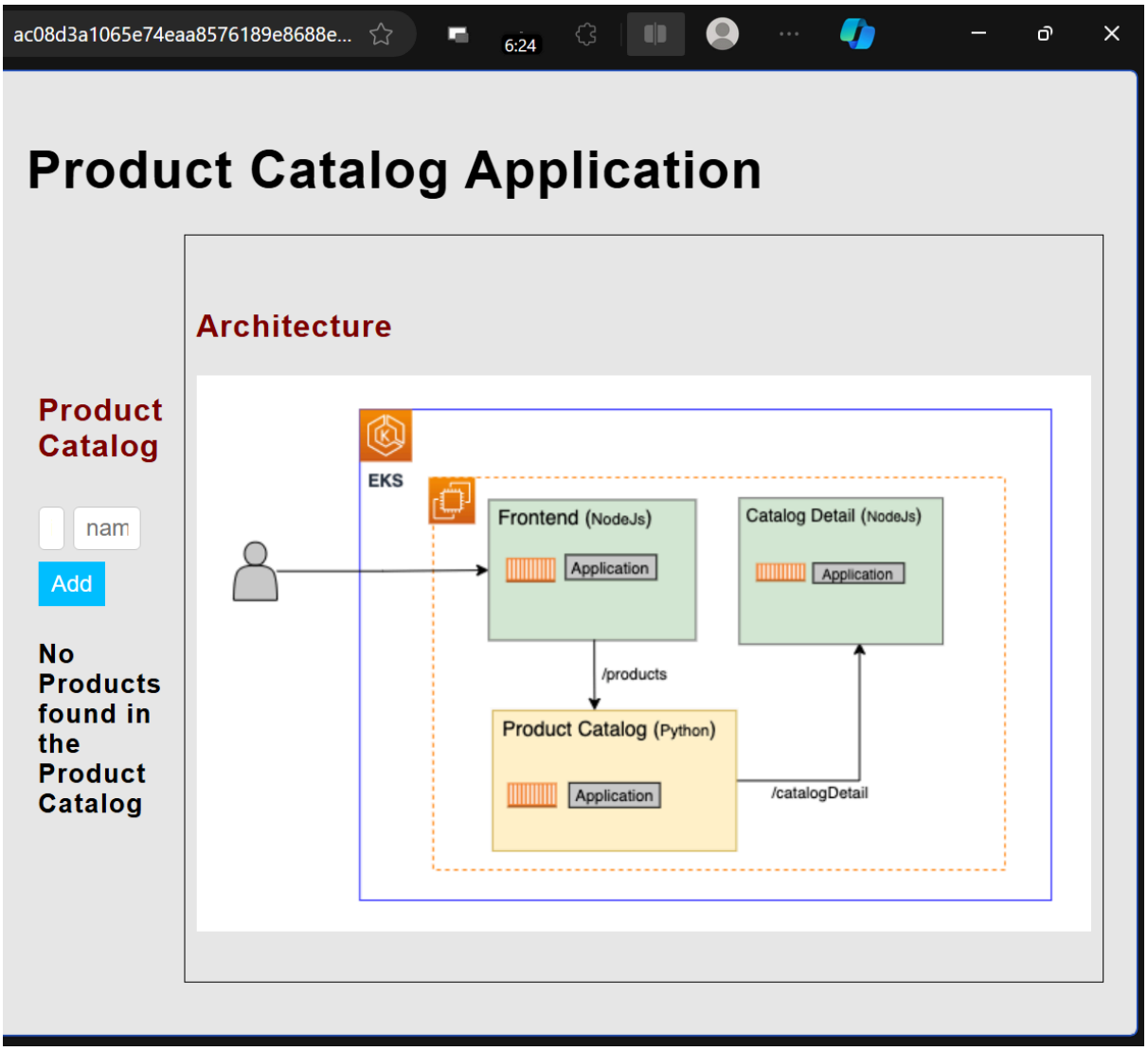
1.7 After creating manifests for the proddetail deployment and service, apply them to your cluster, using the following command:

```
sh-4.2$ kubectl apply -f ~/proddetail-deployment.yaml
deployment.apps/proddetail created
sh-4.2$ kubectl apply -f ~/proddetail-service.yaml
service/proddetail created
```

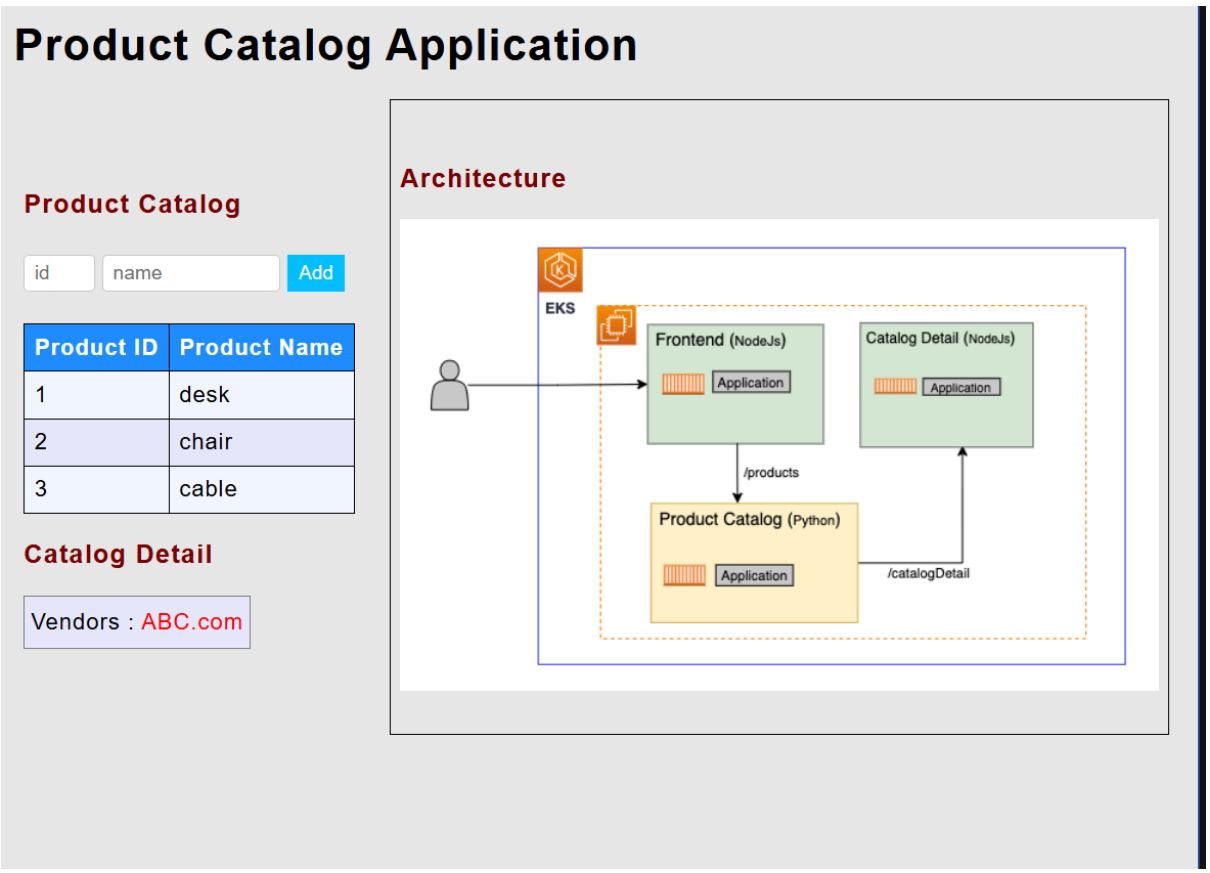
1.8 to retrieve the URL pointing to the application frontend:

```
sh-4.2$ echo "http://"$(kubectl get svc frontend -n workshop | awk 'END { print $4 }'
)
http://ac08d3a1065e74eaa8576189e8688e02-246646263.us-west-2.elb.amazonaws.com
```

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:

```
sh-4.2$ kubectl describe service proddetail -n workshop
Name:                     proddetail
Namespace:                workshop
Labels:                   app=proddetail
Annotations:              owner: student
Selector:                 app=proddetail
Type:                     ClusterIP
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       172.20.117.133
IPs:                      172.20.117.133
Port:                     http  3000/TCP
TargetPort:               3000/TCP
Endpoints:                10.10.117.14:3000
Session Affinity:         None
Events:                   <none>
```

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

```
sh-4.2$ kubectl exec -it $DETAIL_POD -n workshop -- /bin/bash
root@proddetail-6478f64679-m68j6:/usr/src/app#
```

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

```
root@proddetail-6478f64679-m68j6:/usr/src/app# cat /etc/resolv.conf
search workshop.svc.cluster.local svc.cluster.local cluster.local us-west-2.compute.interna
l
nameserver 172.20.0.10
options ndots:5
root@proddetail-6478f64679-m68j6:/usr/src/app#
```

2.4 exit from the pod using:

```
root@proddetail-6478f64679-m68j6:/usr/src/app# exit
exit
sh-4.2$
```

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:

```
sh-4.2$ cat -n ~/detail_deployment_with_probes.yaml && echo
     1  apiVersion: apps/v1
     2  kind: Deployment
     3  metadata:
```

```
24            livenessProbe:
25              httpGet:
26                path: /ping
27                port: 3000
28              initialDelaySeconds: 5
29              periodSeconds: 5
30              timeoutSeconds: 1
31              successThreshold: 1
32              failureThreshold: 3
33            readinessProbe:
34              exec:
35                command:
36                  - /bin/bash
37                  - -c
38                  - cat readiness.txt | grep ready
39              initialDelaySeconds: 15
40              periodSeconds: 3
```

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:

```
sh-4.2$ kubectl apply -f ~/detail_deployment_with_probes.yaml
deployment.apps/proddetail configured
sh-4.2$
```

3.3 To become the root user inside the container:

```
sh-4.2$ kubectl exec -it $DETAIL_POD -n workshop -- bash
root@proddetail-7d47bfdb49-6sxq4:/usr/src/app#
```

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)

```
root@proddetail-7d47bfdb49-6sxq4:/usr/src/app# curl http://proddetail.workshop.svc.cluster.
local:3000/injectFault && while sleep 5; do printf "\n...Getting detail status... " && curl
 http://proddetail.workshop.svc.cluster.local:3000/ping; done
```

```
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"
...Getting detail status... "UnHealthy"command terminated with exit code 137
sh-4.2$
```

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:

```
sh-4.2$ kubectl get event -n workshop --field-selector involvedObject.name=$DETAIL_POD
LAST SEEN   TYPE      REASON      OBJECT                          MESSAGE
6m30s       Normal    Scheduled   pod/proddetail-7d47bfdb49-6sxq4   Successfully assigned w
orkshop/proddetail-7d47bfdb49-6sxq4 to ip-10-10-103-76.us-west-2.compute.internal
69s         Normal    Pulling     pod/proddetail-7d47bfdb49-6sxq4   Pulling image "public.e
cr.aws/u2g6w7p2/eks-workshop-demo/catalog_detail:1.0"
6m29s       Normal    Pulled      pod/proddetail-7d47bfdb49-6sxq4   Successfully pulled ima
ge "public.ecr.aws/u2g6w7p2/eks-workshop-demo/catalog_detail:1.0" in 206ms (206ms including
 waiting)
69s         Normal    Created     pod/proddetail-7d47bfdb49-6sxq4   Created container prodd
etail
69s         Normal    Started     pod/proddetail-7d47bfdb49-6sxq4   Started container prodd
etail
99s         Warning   Unhealthy   pod/proddetail-7d47bfdb49-6sxq4   Liveness probe failed:
HTTP probe failed with statuscode: 500
99s         Normal    Killing     pod/proddetail-7d47bfdb49-6sxq4   Container proddetail fa
iled liveness probe, will be restarted
69s         Normal    Pulled      pod/proddetail-7d47bfdb49-6sxq4   Successfully pulled ima
ge "public.ecr.aws/u2g6w7p2/eks-workshop-demo/catalog_detail:1.0" in 162ms (162ms including
 waiting)
sh-4.2$
```

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.

```
99s         Warning   Unhealthy   pod/proddetail-7d47bfdb49-6sxq4   Liveness probe failed:
HTTP probe failed with statuscode: 500
99s         Normal    Killing     pod/proddetail-7d47bfdb49-6sxq4   Container proddetail fa
iled liveness probe, will be restarted
69s         Normal    Pulled      pod/proddetail-7d47bfdb49-6sxq4   Successfully pulled ima
ge "public.ecr.aws/u2g6w7p2/eks-workshop-demo/catalog_detail:1.0" in 162ms (162ms including
 waiting)
```

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

```
sh-4.2$ kubectl get pod -n workshop -l app=proddetail
NAME                          READY   STATUS    RESTARTS        AGE
proddetail-7d47bfdb49-6sxq4   1/1     Running   1 (3m22s ago)   8m43s
```

We have tested the liveness probe.
Similarly readiness probe can also be tested:

to test the *readiness probe*. Ordinarily, Kubernetes begins sending requests to containers as soon as they are determined to be up and running. In some cases, however, a newly launched container might signal that it is *READY* before all of the processes inside of it have finished loading. This can result in startup and autoscaling errors. By using a **readiness probe** , administrators can ensure that services do not start sending requests to new pods before they are ready to respond.

Task complete: You have successfully configured liveness and readiness probes on the *Product Catalog* application.

Task 4: Delete the application

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

```
sh-4.2$ kubectl delete namespace workshop
namespace "workshop" deleted
```

Session terminated a54d7174-4dc8-4dea-9460-704a39a0ddb9-pukasd5gbxchfye7znkobnlwoa

Your session has been terminated.

Cancel Close