Practice Lab: Introduction to Kubernetes Objects

Estimated time needed: 45 minutes

This practice lab is designed to provide hands-on experience with Kubernetes, focusing on creating services, using various kubectl commands, and deploying StatefulSets and DaemonSets.

Objectives

After completing this lab, you will be able to:

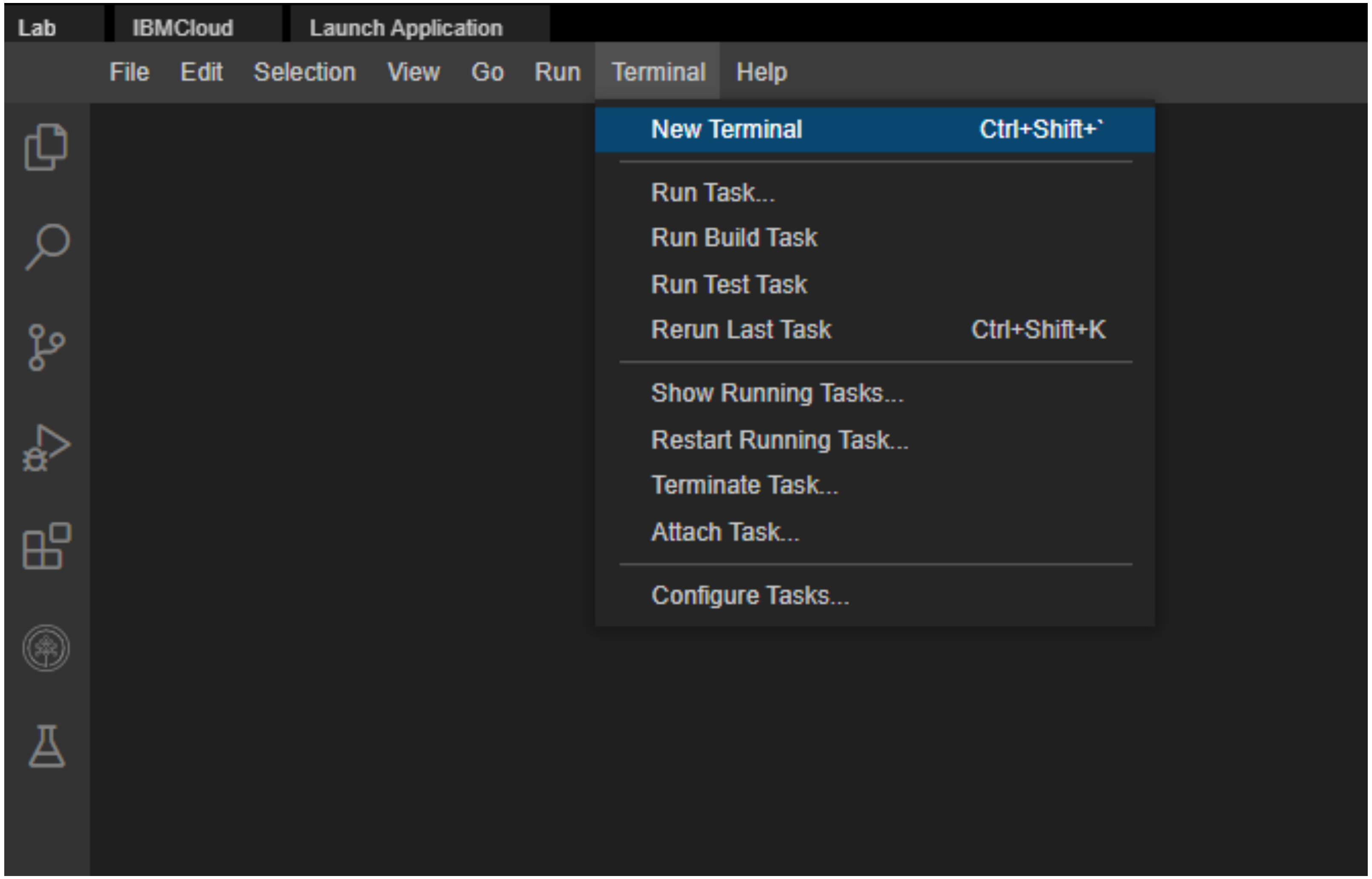
- Create a Kubernetes Service
- Use various `kubectl` commands
- Deploy a StatefulSet for managing stateful applications
- Implement a DaemonSet for running a single pod on all nodes

> **Note: Kindly complete the lab in a single session without any break because the lab may go on offline mode and may cause errors. If you face any issues or errors during the lab process, please log out of the lab environment. Then clear your system cache and cookies and try to complete the lab.**

Setup Environment

Open a terminal window using the menu: `Terminal > New Terminal`.

> Note: If the terminal is already opened, please skip this step.

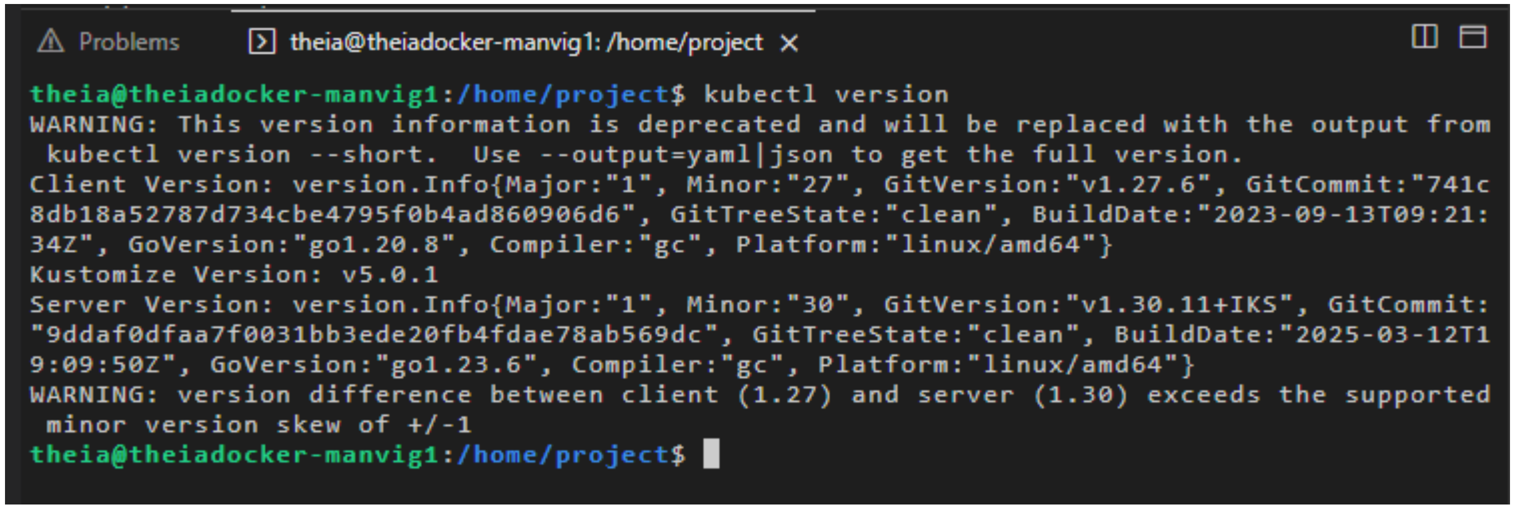


Step 1: Verify kubectl Version

Before proceeding, ensure that you have kubectl installed and properly configured. To check the version of kubectl, run the following command:

```
kubectl version
```

You should see the following output, although the versions may be different:



```
theia@theiadocker-manvig1:/home/project$ kubectl version
WARNING: This version information is deprecated and will be replaced with the output from
 kubectl version --short.  Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"27", GitVersion:"v1.27.6", GitCommit:"741c
8db18a52787d734cbe4795f0b4ad860906d6", GitTreeState:"clean", BuildDate:"2023-09-13T09:21:
34Z", GoVersion:"go1.20.8", Compiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v5.0.1
Server Version: version.Info{Major:"1", Minor:"30", GitVersion:"v1.30.11+IKS", GitCommit:
"9ddaf0dfaa7f0031bb3ede20fb4fdae78ab569dc", GitTreeState:"clean", BuildDate:"2025-03-12T1
9:09:50Z", GoVersion:"go1.23.6", Compiler:"gc", Platform:"linux/amd64"}
WARNING: version difference between client (1.27) and server (1.30) exceeds the supported
 minor version skew of +/-1
theia@theiadocker-manvig1:/home/project$
```

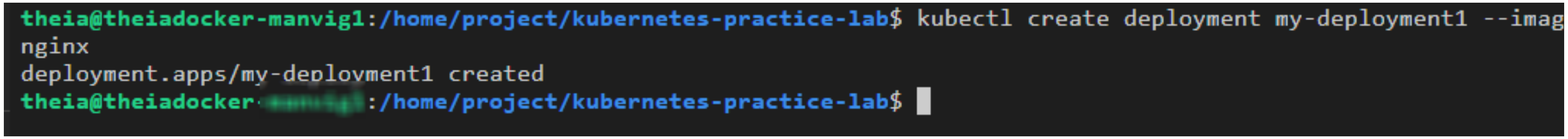
Task 1: Create a Kubernetes Service using nginx image

A popular open-source web server, **nginx** is known for its high performance, stability, and low resource usage. It can also function as a reverse proxy, load balancer, and HTTP cache.

Creating a Kubernetes Service using an nginx image involves setting up a networking layer that allows other components within the Kubernetes cluster or external users to access the nginx application running in pods. To run nginx as a service in Kubernetes, you can follow these steps:

1. **Create a Deployment named `my-deployment1` using the nginx image**

```
kubectl create deployment my-deployment1 --image=nginx
```



```
theia@theiadocker-manvig1:/home/project/kubernetes-practice-lab$ kubectl create deployment my-deployment1 --imag
nginx
deployment.apps/my-deployment1 created
theia@theiadocker-[redacted]:/home/project/kubernetes-practice-lab$
```

`kubectl`: The command-line tool for interacting with the Kubernetes API.

`create deployment`: Tells Kubernetes that you want to create a new Deployment. A Deployment is a Kubernetes object that manages a set of replicated Pods, ensuring that the specified number of replicas are running and updated.

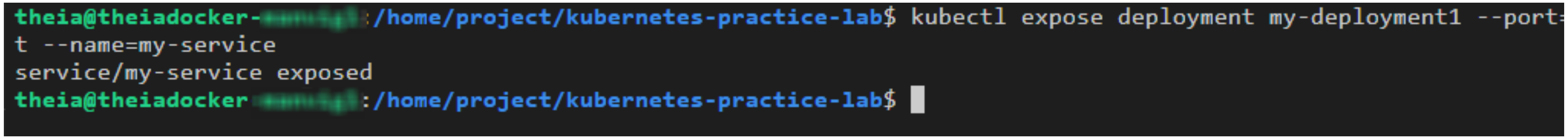
`my-deployment1`: It is the name of the Deployment being created. In this case, the Deployment is named my-deployment1.

`--image=nginx`: It specifies the container image used for the Pods managed by this Deployment. The nginx image is a popular web server and reverse proxy server.

It creates a Deployment named my-deployment1 that uses the nginx image. Deployments manage the rollout and scaling of applications.

2. **Expose the deployment as a service**

```
kubectl expose deployment my-deployment1 --port=80 --type=NodePort --name=my-service1
```

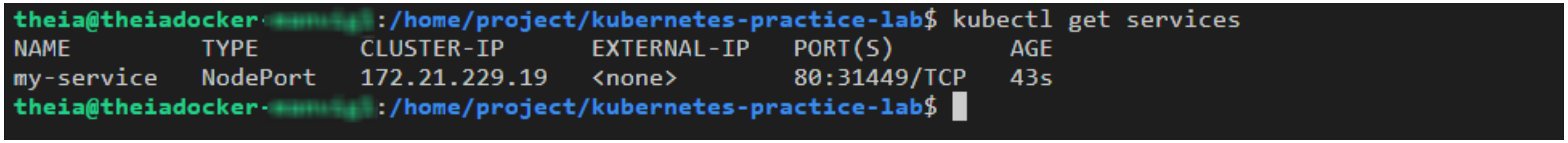


```
theia@theiadocker-[redacted]:/home/project/kubernetes-practice-lab$ kubectl expose deployment my-deployment1 --port=
t --name=my-service
service/my-service exposed
theia@theiadocker[redacted]:/home/project/kubernetes-practice-lab$ 
```

It exposes the my-deployment1 Deployment as a Service named my-service1, making it accessible on port 80 through a NodePort. NodePort services allow external traffic to access the service.

3. **Lists all services in the default namespace. Services provide a stable IP address and DNS name for accessing a set of pods.**

   ```
   kubectl get services
   ```



```
theia@theiadocker[redacted]:/home/project/kubernetes-practice-lab$ kubectl get services
NAME         TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
my-service   NodePort   172.21.229.19   <none>        80:31449/TCP   43s
theia@theiadocker-[redacted]:/home/project/kubernetes-practice-lab$ 
```

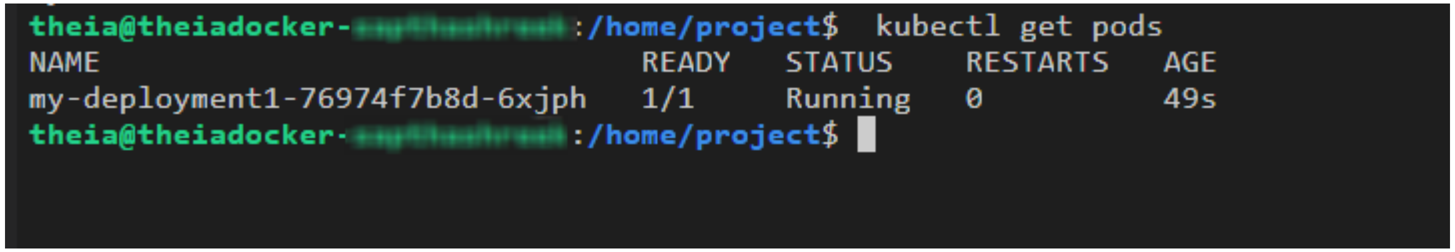
This command lists all the services in the default namespace, including nginx-service, and provides details such as the ClusterIP, NodePort, and target port.

By following these steps, you create a Kubernetes Service named `nginx`, which routes traffic to the nginx pods running in your cluster, making them accessible internally and externally via the assigned NodePort.

Task 2: Manage Kubernetes Pods and Services

1. Get the list of pods

```
kubectl get pods
```



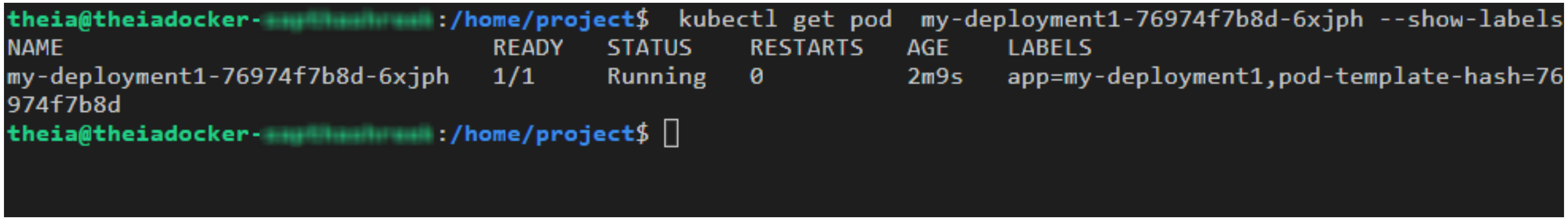
```
theia@theiadocker-[redacted]:/home/project$  kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-deployment1-76974f7b8d-6xjph    1/1     Running   0          49s
theia@theiadocker-[redacted]:/home/project$
```

This command displays all pods, including those created by the my-deployment1 Deployment.

2. Show labels

Replace `<pod-name>` with the actual pod Name:

```
kubectl get pod <pod-name> --show-labels
```



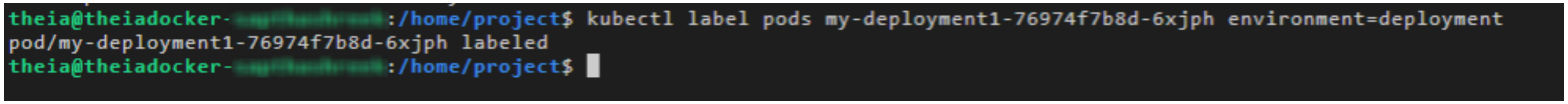
```
theia@theiadocker-[redacted]:/home/project$  kubectl get pod  my-deployment1-76974f7b8d-6xjph --show-labels
NAME                              READY   STATUS    RESTARTS   AGE    LABELS
my-deployment1-76974f7b8d-6xjph   1/1     Running   0          2m9s   app=my-deployment1,pod-template-hash=76
974f7b8d
theia@theiadocker-[redacted]:/home/project$ 
```

This command will list the labels associated with the specified pod, helping you identify its attributes and categorization within your Kubernetes cluster.

3. Label the pod

Replace `<pod-name>` with the actual pod Name:

```
kubectl label pods <pod-name> environment=deployment
```



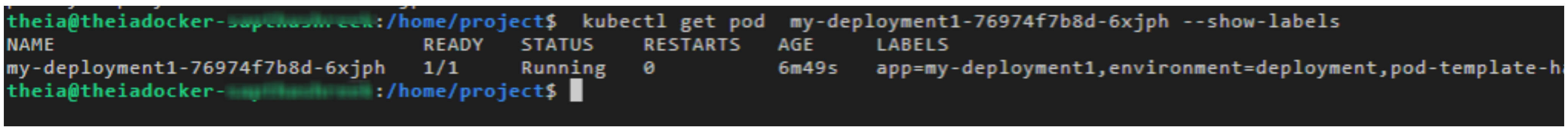
```
theia@theiadocker-[redacted]:/home/project$ kubectl label pods my-deployment1-76974f7b8d-6xjph environment=deployment
pod/my-deployment1-76974f7b8d-6xjph labeled
theia@theiadocker-[redacted]:/home/project$ 
```

The command is used in Kubernetes to label a specific pod with the key-value pair `environment=deployment`. This label helps categorize and manage pods based on their deployment environment, making it easier to organize and select Kubernetes objects within the cluster.

4. Show labels

Replace `<pod-name>` with the actual pod Name:

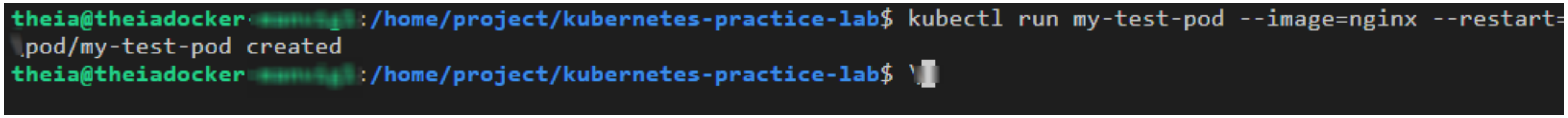
```
kubectl get pod <pod-name> --show-labels
```



```
theia@theiadocker-[redacted]:/home/project$  kubectl get pod  my-deployment1-76974f7b8d-6xjph --show-labels
NAME                              READY   STATUS    RESTARTS   AGE     LABELS
my-deployment1-76974f7b8d-6xjph   1/1     Running   0          6m49s   app=my-deployment1,environment=deployment,pod-template-h
theia@theiadocker-[redacted]:/home/project$
```

5. Run a test pod using the `nginx` image

```
kubectl run my-test-pod --image=nginx --restart=Never
```



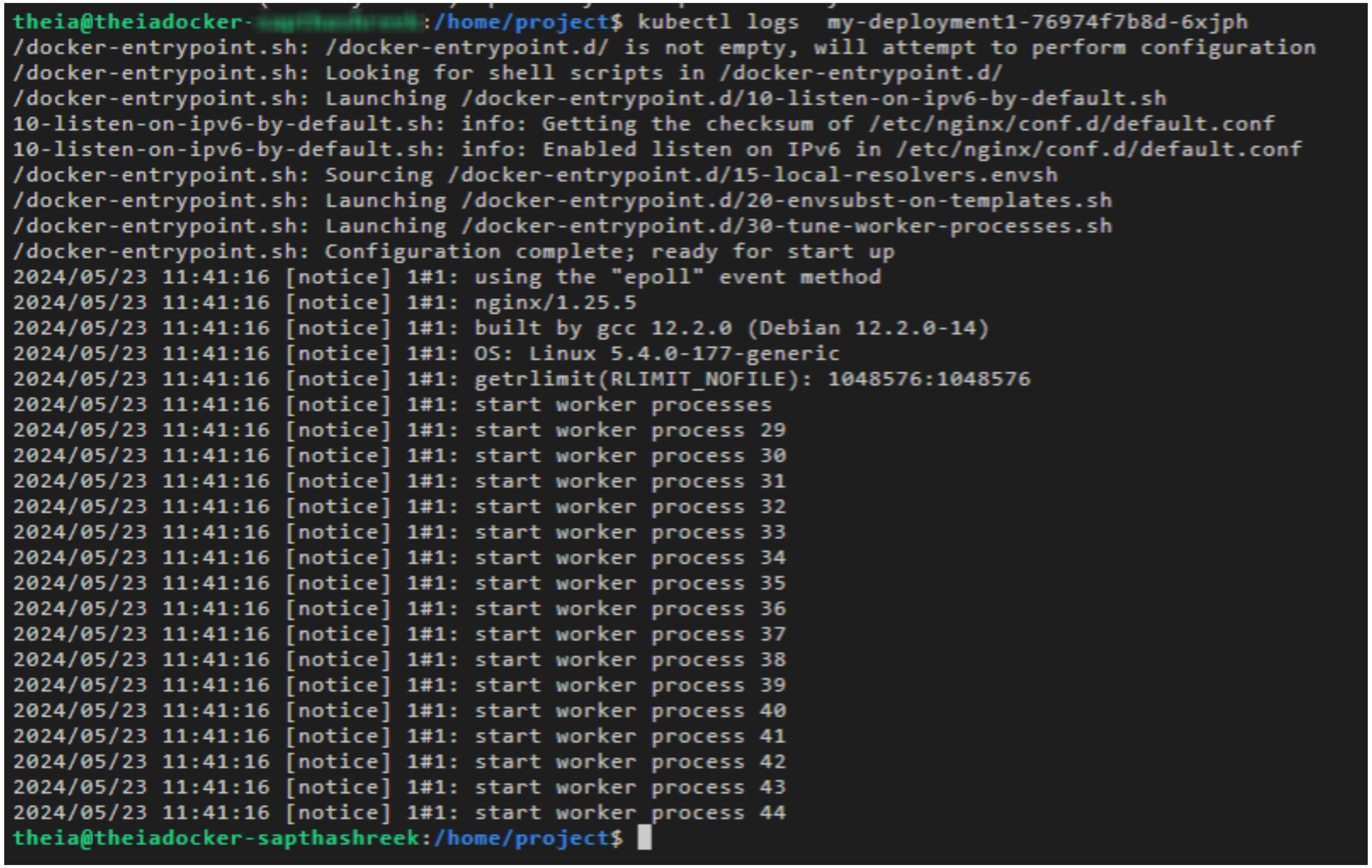
```
theia@theiadocker[redacted]:/home/project/kubernetes-practice-lab$ kubectl run my-test-pod --image=nginx --restart=
pod/my-test-pod created
theia@theiadocker[redacted]:/home/project/kubernetes-practice-lab$
```

This command tells Kubernetes to create a pod named "my-test-pod" using the nginx image, and the pod will not restart automatically if it stops for any reason as we are using `--restart=Never`.

6. Show logs

```
kubectl logs <pod-name>
```

Replace `<pod-name>` with the actual name of the pod.



```
theia@theiadocker-[redacted]:/home/project$ kubectl logs  my-deployment1-76974f7b8d-6xjph
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/05/23 11:41:16 [notice] 1#1: using the "epoll" event method
2024/05/23 11:41:16 [notice] 1#1: nginx/1.25.5
2024/05/23 11:41:16 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/05/23 11:41:16 [notice] 1#1: OS: Linux 5.4.0-177-generic
2024/05/23 11:41:16 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/05/23 11:41:16 [notice] 1#1: start worker processes
2024/05/23 11:41:16 [notice] 1#1: start worker process 29
2024/05/23 11:41:16 [notice] 1#1: start worker process 30
2024/05/23 11:41:16 [notice] 1#1: start worker process 31
2024/05/23 11:41:16 [notice] 1#1: start worker process 32
2024/05/23 11:41:16 [notice] 1#1: start worker process 33
2024/05/23 11:41:16 [notice] 1#1: start worker process 34
2024/05/23 11:41:16 [notice] 1#1: start worker process 35
2024/05/23 11:41:16 [notice] 1#1: start worker process 36
2024/05/23 11:41:16 [notice] 1#1: start worker process 37
2024/05/23 11:41:16 [notice] 1#1: start worker process 38
2024/05/23 11:41:16 [notice] 1#1: start worker process 39
2024/05/23 11:41:16 [notice] 1#1: start worker process 40
2024/05/23 11:41:16 [notice] 1#1: start worker process 41
2024/05/23 11:41:16 [notice] 1#1: start worker process 42
2024/05/23 11:41:16 [notice] 1#1: start worker process 43
2024/05/23 11:41:16 [notice] 1#1: start worker process 44
theia@theiadocker-sapthashreek:/home/project$ 
```

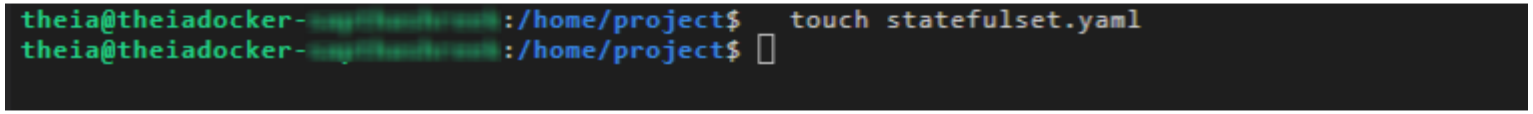
This command retrieves and displays the logs generated by the specified pod, allowing you to troubleshoot issues, monitor activity, and gather information about the pod's behavior.

Task 3: Deploying a StatefulSet

A StatefulSet manages the deployment and scaling of a set of pods, and maintains a sticky identity for each of their Pods, ensuring that each Pod has a persistent identity and storage.

1. **Create and open a file named `statefulset.yaml` in edit mode.**

```
touch statefulset.yaml
```



```
theia@theiadocker-[redacted]:/home/project$   touch statefulset.yaml
theia@theiadocker-[redacted]:/home/project$
```

2. **Open `statefulset.yaml`, and add the following code, and save the file:**

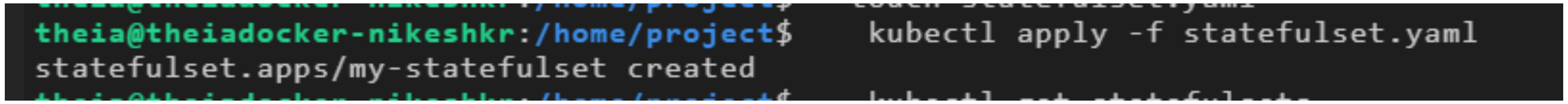
```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-statefulset
spec:
  serviceName: "nginx"
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

Explanation:

- `apiVersion: apps/v1` & `kind: StatefulSet`: Establishes that this resource is a StatefulSet, leveraging the stable, production-ready apps/v1 API.
- `metadata.name: my-statefulset`: Assigns a human-readable identifier to the StatefulSet.
- `spec.serviceName: "nginx"`: Binds the StatefulSet to a headless Service named “nginx,” ensuring each pod has a stable network identity.
- `spec.replicas: 3`: Orchestrates three pod replicas, each with its own persistent identity.
- `spec.selector.matchLabels`: Directs the StatefulSet to manage pods labeled app: nginx.
- `spec.template`: Defines the pod blueprint:
    - `metadata.labels`: app: nginx ensures new pods carry the matching label.
    - `spec.containers` configures the `nginx` container on port 80, named “web.”
- `volumeClaimTemplates`: Automates the creation of a PersistentVolumeClaim named www for each replica, each requesting 1 Gi of storage with ReadWriteOnce access.

3. Apply the StatefulSet configuration.

```
kubectl apply -f statefulset.yaml
```

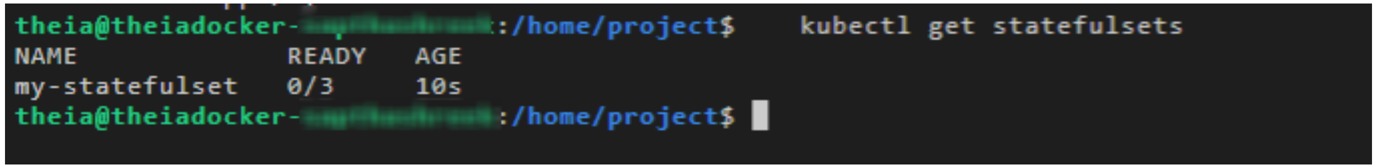


```
theia@theiadocker-nikeshkr:/home/project$   kubectl apply -f statefulset.yaml
statefulset.apps/my-statefulset created
```

This command tells Kubernetes to create the resources defined in the YAML file.

4. **Verify that the StatefulSet is created.**

```
kubectl get statefulsets
```



```
theia@theiadocker-[redacted]:/home/project$   kubectl get statefulsets
NAME             READY   AGE
my-statefulset   0/3     10s
theia@theiadocker-[redacted]:/home/project$
```

After applying the StatefulSet, you should verify that the StatefulSet has been created and is running. This can be done using the kubectl get command.

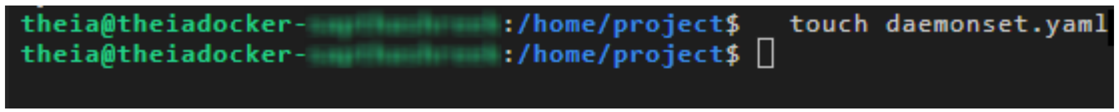
By following these steps, you can successfully apply a StatefulSet in Kubernetes. The kubectl apply command is used to create the StatefulSet, and the kubectl get command helps you verify that the StatefulSet is running as expected.

Task 4: Implementing a DaemonSet

A DaemonSet ensures that a copy of a specific Pod runs on all (or some) nodes in the cluster. It is particularly useful for deploying system-level applications that provide essential services across the nodes in a cluster, such as log collection, monitoring, or networking services.

1. **Create a file named `daemonset.yaml` and open it in edit mode:**

```
touch daemonset.yaml
```



```
theia@theiadocker-[redacted]:/home/project$   touch daemonset.yaml
theia@theiadocker-[redacted]:/home/project$ 
```

2. **Create and open a file named `daemonset.yaml` in edit mode.**

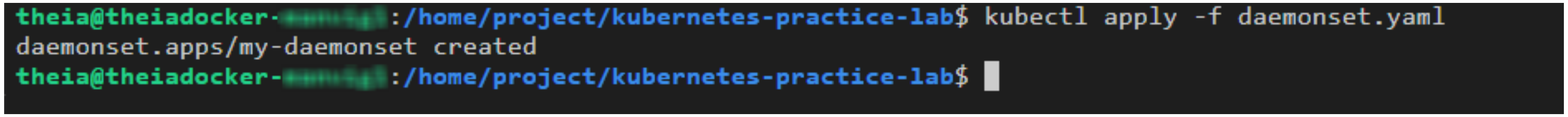
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
spec:
  selector:
    matchLabels:
      name: my-daemonset
  template:
    metadata:
      labels:
        name: my-daemonset
    spec:
      containers:
      - name: my-daemonset
        image: nginx
```

Explanation

- `apiVersion: apps/v1` & `kind: DaemonSet`: Declares this resource as a DaemonSet under the stable apps/v1 API.
- `metadata.name: my-daemonset`: Provides the DaemonSet's unique name.
- `spec.selector.matchLabels`: Targets pods labeled name: my-daemonset for scheduling.
- `spec.template.metadata.labels`: Labels the pod template so it matches the selector.
- `spec.template.spec.containers`: Defines a single container named my-daemonset using the nginx image.

3. Apply the DaemonSet

   ```
   kubectl apply -f daemonset.yaml
   ```

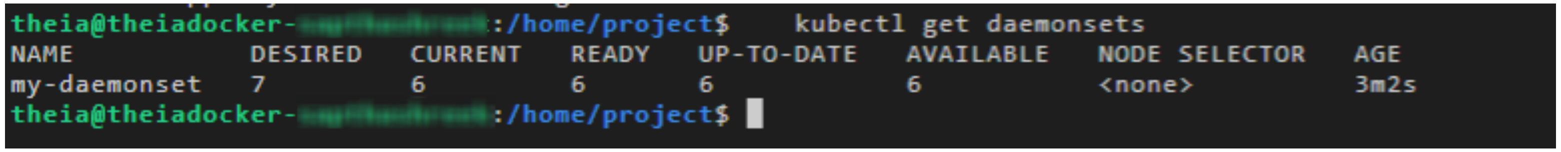


```
theia@theiadocker-[redacted]:/home/project/kubernetes-practice-lab$ kubectl apply -f daemonset.yaml
daemonset.apps/my-daemonset created
theia@theiadocker-[redacted]:/home/project/kubernetes-practice-lab$
```

This command tells Kubernetes to apply the configuration defined in the daemonset.yaml file. The apply command is used to create or update Kubernetes resources based on the configuration provided in the YAML file.

4. Verify that the DaemonSet has been created

```
kubectl get daemonsets
```



```
theia@theiadocker-[redacted]:/home/project$   kubectl get daemonsets
NAME           DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
my-daemonset   7         6         6       6            6           <none>          3m2s
theia@theiadocker-[redacted]:/home/project$
```

This output from `kubectl get daemonsets` provides information about the DaemonSet named "my-daemonset" in your Kubernetes cluster.

- **NAME**: The name of the DaemonSet, which is "my-daemonset" in this case.
- **DESIRED**: The desired number of DaemonSet pods. In your case, it's set to 7.
- **CURRENT**: The current number of DaemonSet pods running. It shows 6 pods are currently running.
- **READY**: The number of DaemonSet pods that are ready and available for use. All 6 running pods are ready.
- **UP-TO-DATE**: The number of DaemonSet pods that are up-to-date with the latest configuration.

- **AVAILABLE**: The number of DaemonSet pods that are available for use.
- **NODE SELECTOR**: Specifies which nodes in the cluster the DaemonSet should run on. In this case, it's set to `<none>`, meaning the DaemonSet is not restricted to specific nodes.
- **AGE**: The age of the DaemonSet, indicating how long it has been running.

Conclusion

Congratulations! You have completed the practice lab on Kubernetes. You created a Kubernetes Service, used various kubectl commands, deployed StatefulSets for stateful applications, and implemented DaemonSets for uniform pod deployment across cluster nodes.

Author(s)

Manvi Gupta