

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

Aim : To learn to explore, scale, and update an application with Kubernetes

Scenario : In continuation, light and understanding of the previous practical done by you, where you learned to create a cluster, and deploy and debug an application- continue using the same application, or create a new application and hence perform the following tasks.

Tasks, Codes and Screenshots containing commands and output :

1. Learn about a Service in Kubernetes, and understand how labels and LabelSelector objects relate to a Service.

A Service in Kubernetes is a resource that enables load balancing and service discovery for a set of pods. Labels are key-value pairs assigned to pods, and a LabelSelector is used by a Service to determine which pods to include in the service. Labels and LabelSelectors are crucial for defining the pods associated with a Service, ensuring traffic is correctly directed to them.

2. Expose an application outside a Kubernetes cluster using a Service.

a) **knote.yaml** :

```
apiVersion: v1
kind: Service
metadata:
name: knote
spec:
selector:
app: knote
ports:
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

```
- name: http
protocol: TCP
port: 3000
type: NodePort

---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: knote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: knote
  template:
    metadata:
      labels:
        app: knote
    spec:
      containers:
        - name: app
          image: learnk8s/knote-js:1.0.0
          ports:
            - containerPort: 3000
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

env:

- name: MONGO_URL

value: mongodb://mongo:27017/dev

imagePullPolicy: Always

b) mongo.yaml:

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: mongo-pvc

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 256Mi

apiVersion: v1

kind: Service

metadata:

name: mongo

spec:

selector:

app: mongo

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

ports:

- port: 27017

targetPort: 27017

apiVersion: apps/v1

kind: Deployment

metadata:

name: mongo

spec:

selector:

matchLabels:

app: mongo

template:

metadata:

labels:

app: mongo

spec:

containers:

- name: mongo

image: mongo:3.6.17-xenial

ports:

- containerPort: 27017

volumeMounts:

- name: storage

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

```
mountPath: /data/db

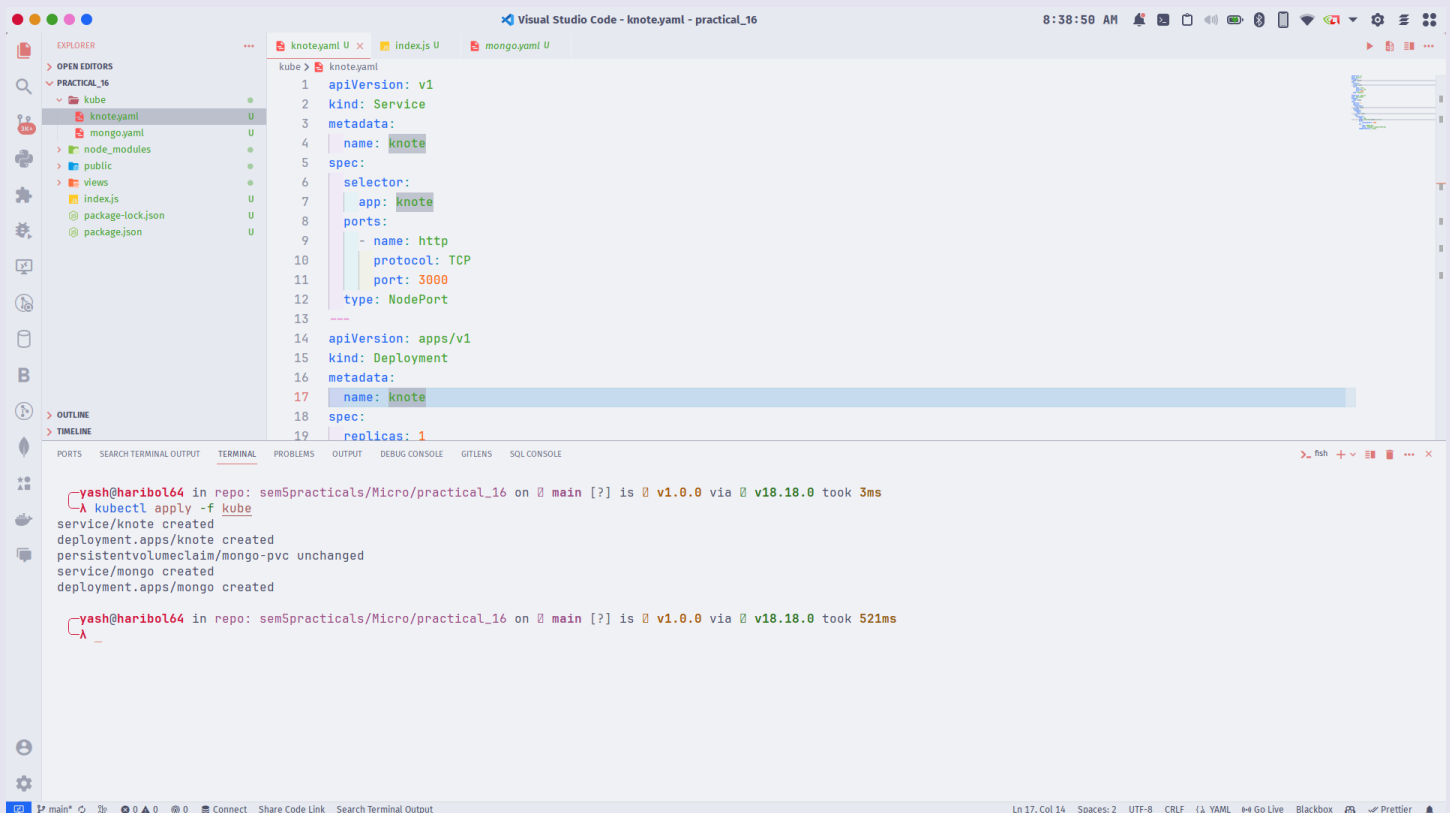
volumes:

- name: storage

persistentVolumeClaim:

claimName: mongo-pvc
```

- c) Now, create deployments of these two files inside **kube** folder using **kubectl** command and wait till all deployments, services, pods are fully created and in working state. It can be checked graphically using **minikube dashboard**.



The screenshot shows a Visual Studio Code editor with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project structure with a 'kube' folder containing 'knoteyaml' and 'mongoyaml' files. The code editor displays the 'knoteyaml' file, which is a Kubernetes Service and Deployment manifest. The terminal shows the output of the 'kubectl apply -f kube' command, indicating that the service, deployment, and persistent volume claim were successfully created.

```
Visual Studio Code - knoteyaml - practical_16
8:38:50 AM

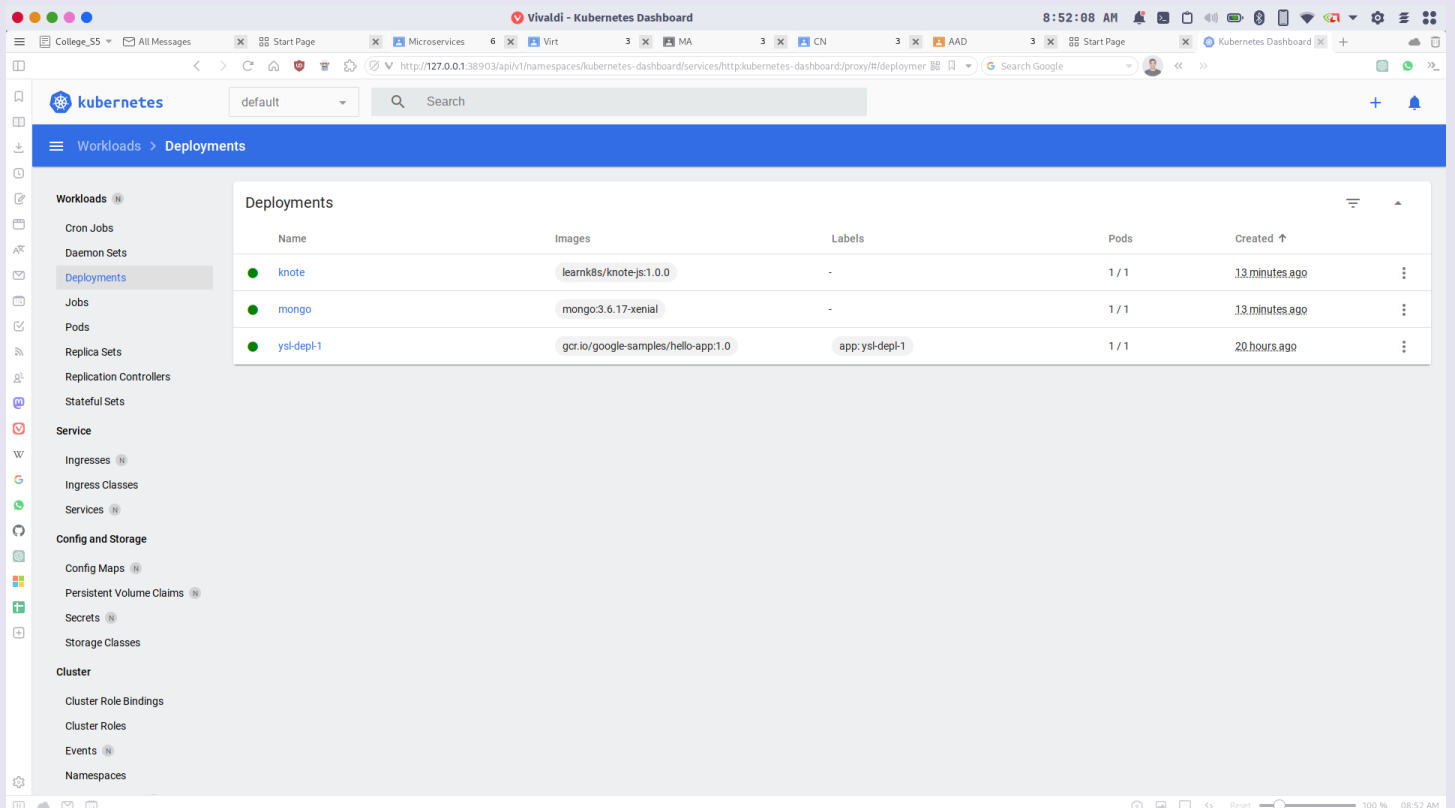
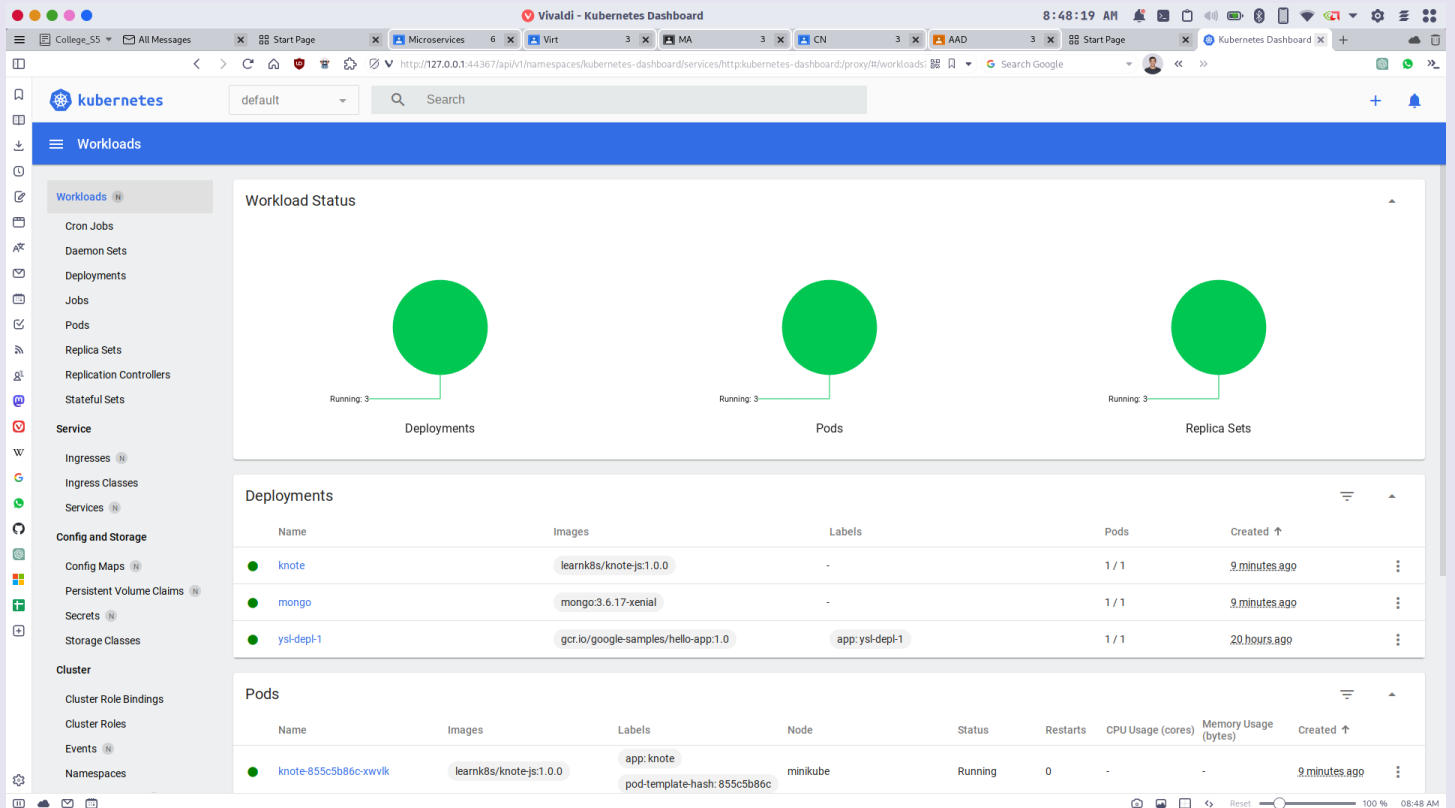
EXPLORER
  PRACTICAL_16
    kube
      knoteyaml
      mongoyaml
      node_modules
      public
      views
      index.js
      package-lock.json
      package.json

knoteyaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: knote
5  spec:
6    selector:
7      app: knote
8    ports:
9      - name: http
10        protocol: TCP
11        port: 3000
12        type: NodePort
13
14  apiVersion: apps/v1
15  kind: Deployment
16  metadata:
17    name: knote
18  spec:
19    replicas: 1

Terminal
yash@haribol64 in repo: sem5practicals/Micro/practical_16 on  main [?] is  v1.0.0 via  v18.18.0 took 3ms
$ kubectl apply -f kube
service/knote created
deployment.apps/knote created
persistentvolumeclaim/mongo-pvc unchanged
service/mongo created
deployment.apps/mongo created

yash@haribol64 in repo: sem5practicals/Micro/practical_16 on  main [?] is  v1.0.0 via  v18.18.0 took 521ms
$ _
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16



Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

Vivaldi - Kubernetes Dashboard

8:52:28 AM

College_S5 All Messages Start Page Microservices 6 Virt 3 MA 3 CN 3 AAD 3 Start Page Kubernetes Dashboard

http://127.0.0.1:38903/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard/proxy/#/service/http:kubernetes-dashboard

kubernetes default Search

Service > Services

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Service

- Ingresses
- Ingress Classes
- Services

Config and Storage

- Config Maps
- Persistent Volume Claims
- Secrets
- Storage Classes

Cluster

- Cluster Role Bindings
- Cluster Roles
- Events
- Namespaces

Services

Name	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
knote	-	NodePort	10.97.239.59	knote:3000 TCP knote:31540 TCP	-	13 minutes ago
mongo	-	ClusterIP	10.104.141.43	mongo:27017 TCP mongo:0 TCP	-	13 minutes ago
ysl-depl-1	app: ysl-depl-1	NodePort	10.111.144.136	ysl-depl-1:8080 TCP ysl-depl-1:31406 TCP	-	20 hours ago
kubernetes	component: apiserver provider: kubernetes	ClusterIP	10.96.0.1	kubernetes:443 TCP kubernetes:0 TCP	-	19 days ago

Vivaldi - Kubernetes Dashboard

8:52:52 AM

College_S5 All Messages Start Page Microservices 6 Virt 3 MA 3 CN 3 AAD 3 Start Page Kubernetes Dashboard

http://127.0.0.1:38903/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard/proxy/#/podname

kubernetes default Search

Workloads > Pods

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Service

- Ingresses
- Ingress Classes
- Services

Config and Storage

- Config Maps
- Persistent Volume Claims
- Secrets
- Storage Classes

Cluster

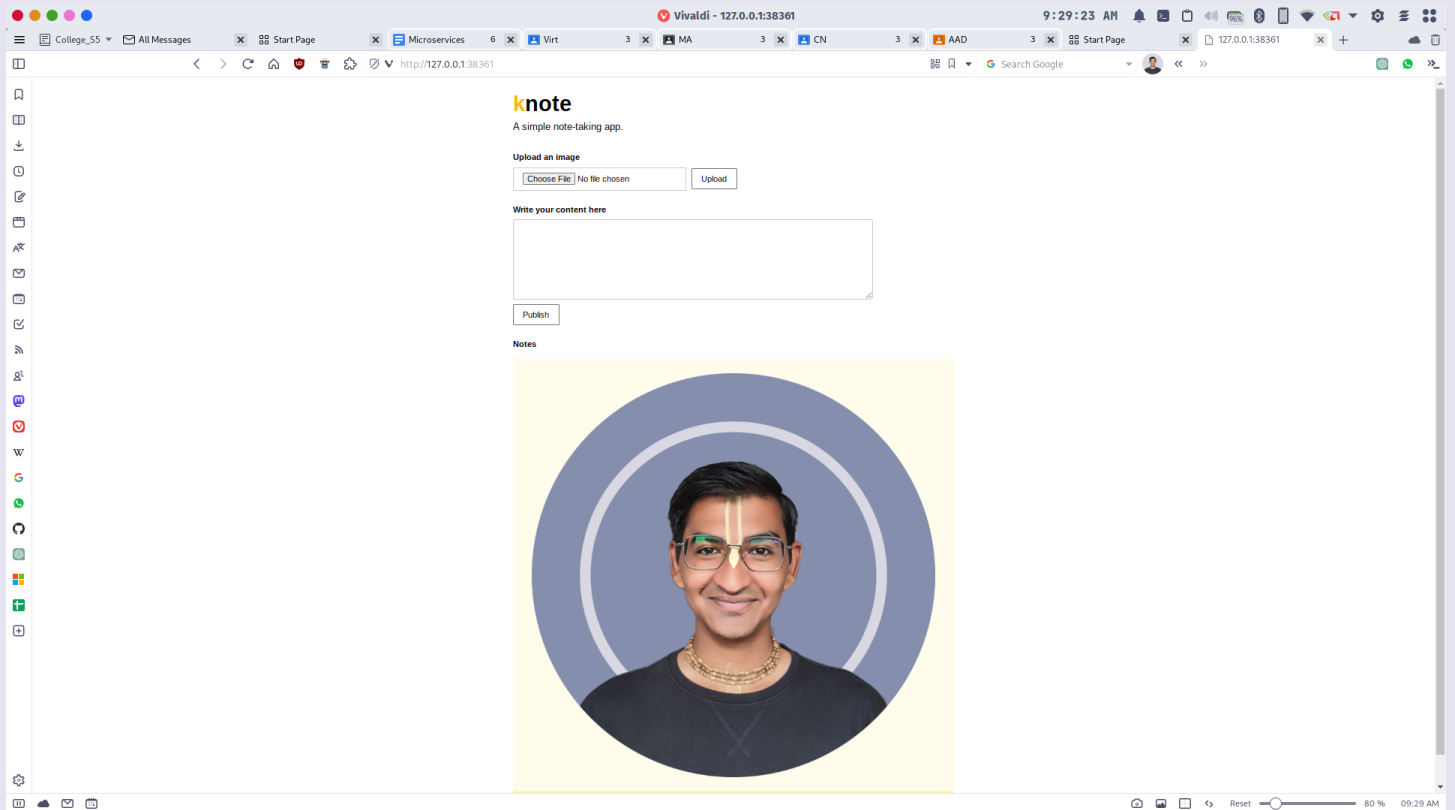
- Cluster Role Bindings
- Cluster Roles
- Events
- Namespaces

Pods

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created ↑
knote-855c5b86c-xwvfk	learnk8s/knote-js:1.0.0	app: knote pod-template-hash: 855c5b86c	minikube	Running	0	-	-	14 minutes ago
mongo-674c55f7b9-zdr2q	mongo:3.6.17-xenial	app: mongo pod-template-hash: 674c55f7b9	minikube	Running	0	-	-	14 minutes ago
ysl-depl-1-6574bdf4f-jgqx4	gcr.io/google-samples/hello-app:1.0	app: ysl-depl-1 pod-template-hash: 6574bdf4f	minikube	Running	2	-	-	20 hours ago

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

d) Run **minikube service knote --url** and on the URL given in command output, upload an image file.



Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

e) Check the database connectivity and existence of the uploaded file via
minikube service mongo --url

The screenshot displays two applications side-by-side. On the left is Visual Studio Code, showing a Kubernetes service definition in a file named `knoteyaml`. The service is named `knote` and is configured to expose port `3000` on the host. The terminal at the bottom shows the command `minikube service mongo --url` being executed, which returns the URL `http://127.0.0.1:38659`. On the right is MongoDB Compass, showing the `dev.notes` collection in the `dev` database. A single document is visible with the following fields: `_id` (ObjectID) and `description` (a long string).

```
apiVersion: v1
kind: Service
metadata:
  name: knote
spec:
  selector:
    app: knote
  ports:
    - name: http
      protocol: TCP
      port: 3000
  type: NodePort
```

```
minikube service mongo --url
http://127.0.0.1:38659
! Because you are using a Docker driver on linux, the terminal needs to
be open to run it.
```

```
{
  "_id": "6530a995db0e606e67c4994",
  "description": " ![] (/uploads/a8b354a5b71f13cfe700221b6a29bf0)"
}
```

Name - Yash Lakhtariya
Enrollment number - 21162101012
Branch - CBA Batch - 51
Microservices Practical 16

3. Scale an app using kubectl.

The application can be scaled by editing replicas also. For example, to increase the replicas of the deployment knote, use command : **kubectl scale --replicas=3 deployment/knote**

The screenshot displays a development environment with Visual Studio Code on the left and the Kubernetes Dashboard on the right.

Visual Studio Code:

- Explorer:** Shows a file tree for 'practical_16' with folders like 'kube', 'node_modules', 'public', 'views', and files like 'knote.yaml', 'mongoyaml', 'package-lock.json', and 'package.json'.
- Editor:** Displays the 'knote.yaml' file with the following content:

```
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: knote
5 spec:
6   selector:
7     app: knote
8   ports:
9     - name: http
10     protocol: TCP
11     port: 3000
12   type: NodePort
```
- Terminal:** Shows the execution of the command `kubectl scale --replicas=3 deployment/knote` and its output: `deployment.apps/knote scaled`. Below it, the command `ls -la` is shown with its output.

Kubernetes Dashboard:

- Workloads > Pods:** A table lists the pods for the 'knote' deployment. The table has columns: Name, Images, Labels, and Node.

Name	Images	Labels	Node
knote-855c5b86c-7g2vw	learnk8s/knote-js:1.0.0	app: knote pod-template-hash: 855c5b86c	minikube
knote-855c5b86c-887pb	learnk8s/knote-js:1.0.0	app: knote pod-template-hash: 855c5b86c	minikube
knote-855c5b86c-lkx9g	learnk8s/knote-js:1.0.0	app: knote pod-template-hash: 855c5b86c	minikube