Paso 1: Crear Proyecto Node.js

1. **Inicializar Proyecto Node.js:**

```
mkdir mi-proyecto-nodejs
cd mi-proyecto-nodejs
npm init -y
```

2. **Crear un archivo de servidor básico (index.js):**

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;

app.get('/', (req, res) => {
  res.send('Hola Mundo desde Node.js!');
});

app.listen(PORT, () => {
  console.log(`Servidor corriendo en puerto ${PORT}`);
});
```

3. **Instalar Express:**

```
npm install express
```

Paso 2: Dockerizar la Aplicación

1. **Crear un archivo Dockerfile:**

```
FROM node:14
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "index.js"]
```

2. **Construir la imagen de Docker:**

```
docker build -t [USUARIO DOCKERHUB]/mi-app-nodejs .
```

3. **Subir la imagen de Docker:**

```
docker push [USUARIO DOCKERHUB]/mi-app-nodejs
```

Paso 3: Preparar Kubernetes (usando Docker for Desktop)

1. **Asegurarse de que Kubernetes esté habilitado en Docker for Desktop.**

Paso 4: Desplegar en Kubernetes

1. **Crear un Namespace:**

```
# namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: mi-namespace
```

Aplicar con:

```
kubectl apply -f namespace.yaml
```

2. **Deployment:**

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-deployment
  namespace: mi-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mi-app-nodejs
  template:
    metadata:
      labels:
        app: mi-app-nodejs
    spec:
      containers:
      - name: mi-app-nodejs
        image: [USUARIO DOCKERHUB]/mi-app-nodejs
        ports:
        - containerPort: 3000
```

Aplicar con:

```
kubectl apply -f deployment.yaml
```

3. **ReplicaSet, StatefulSet, DaemonSet, y Job:**

ReplicaSet

```yaml
# replicaset.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: mi-replicaset
  namespace: mi-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mi-app-nodejs
  template:
    metadata:
      labels:
        app: mi-app-nodejs
    spec:
      containers:
      - name: mi-app-nodejs
        image: [USUARIO DOCKERHUB]/mi-app-nodejs
        ports:
        - containerPort: 3000
```

StatefulSet

```yaml
# statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mi-statefulset
  namespace: mi-namespace
spec:
  serviceName: "mi-servicio"
  replicas: 3
  selector:
    matchLabels:
      app: mi-app-nodejs
  template:
    metadata:
      labels:
        app: mi-app-nodejs
    spec:
      containers:
      - name: mi-app-nodejs
        image: [USUARIO DOCKERHUB]/mi-app-nodejs
        ports:
        - containerPort: 3000
```

DaemonSet

```yaml
# daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mi-daemonset
  namespace: mi-namespace
spec:
  selector:
    matchLabels:
      app: mi-app-nodejs
  template:
    metadata:
      labels:
        app: mi-app-nodejs
    spec:
      containers:
      - name: mi-app-nodejs
        image: [USUARIO DOCKERHUB]/mi-app-nodejs
        ports:
        - containerPort: 3000
```

Job

```yaml
# job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: mi-job
  namespace: mi-namespace
spec:
  template:
    spec:
      containers:
      - name: mi-job
        image: [USUARIO DOCKERHUB]/mi-app-nodejs
        command: ["sh", "-c", "echo Hola Mundo desde el Job!"]
      restartPolicy: Never
  backoffLimit: 4
```

> **Nota:** recuerda aplicar cada uno de los archivos yaml usando el comando:

```
kubectl -f [NOMBRE DEL YAML].yaml apply
```

Paso 5: Acceso y Pruebas

1. **Exponer el servicio:**
   Para acceder a tu aplicación desde fuera del clúster de Kubernetes, debes exponerla como un servicio.
   Ejemplo básico:

```
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: mi-servicio
  namespace: mi-namespace
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 3000
  selector:
    app: mi-app-nodejs
```

   Aplicar con:

```
kubectl apply -f service.yaml
```

2. **Verificar el despliegue:**

```
kubectl get all -n mi-namespace
```