Task 1 - Preparation

1.1. Choose an Application
For this task, I have chosen an open-source project called “Gitea”, a self-hosted Git service. It is lightweight, easy to deploy, and meets the requirements outlined in the task.

- Project Repository: [Gitea GitHub Repository](https://github.com/go-gitea/gitea)
- Brief Description: Gitea is a community-managed lightweight code hosting solution written in Go. It is published under the MIT license and can be run as a standalone binary or in a Docker container. It provides features similar to GitHub, such as repository management, issue tracking, pull requests, and more.
- Why Gitea?:
  - It uses configuration files in JSON, YAML, or TOML formats.
  - It supports environment variables for configuration.
  - It requires secrets for database credentials, OAuth tokens, etc.
  - It can be exposed to the internet (optional).
  - It has health check endpoints (e.g., `/api/v1/healthz`).

1.2. Get Familiar with Kubernetes (k8s) and Concepts
Kubernetes (k8s) is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. Key concepts to understand include:
- **Pods**: The smallest deployable units in Kubernetes, which can contain one or more containers.
- **Services**: An abstraction that defines a logical set of Pods and a policy to access them.
- **Deployments**: Manage the deployment and scaling of Pods.
- **ConfigMaps and Secrets**: Used to manage configuration data and sensitive information, respectively.
- **Ingress**: Manages external access to services in a cluster.
- **Namespaces**: Provide a scope for resources within a cluster.

1.3. Install and Set Up the Necessary Tools
To work with Kubernetes, you need to install the following tools:

1. **kubectl** :
   - The Kubernetes command-line tool used to interact with the Kubernetes cluster.
   - Installation instructions:
     - For Linux:

```
 curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps$ kubectl
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
  create          Create a resource from a file or from stdin
  expose          Take a replication controller, service, deployment or pod and
expose it as a new Kubernetes service
  run             Run a particular image on the cluster
  set             Set specific features on objects
```

2. **Minikube** :
   - A tool that allows you to run a single-node Kubernetes cluster locally.
   - Installation instructions:
     - For Linux:

```
curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-a
md64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

3. Start Minikube:
   - After installation, start Minikube:

```
minikube start
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps$ minikube start
😄  minikube v1.35.0 on Ubuntu 24.04
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.46 ...
🏃  Updating the running docker "minikube" container ...
```

   - Verify the cluster is running:

```
kubectl get nodes
```

1.4. Get Access to Kubernetes Dashboard
The Kubernetes Dashboard provides a web-based UI for managing your cluster.

1. **Enable the Dashboard** :
   - Minikube includes the Kubernetes Dashboard as an addon. Enable it with:

```
minikube addons enable dashboard
```

2. **Access the Dashboard** :
   - Start the dashboard proxy:

kubectl proxy

- Access the dashboard in your browser:

```
http://127.0.0.1:35247/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/workloads?namespace=default
```

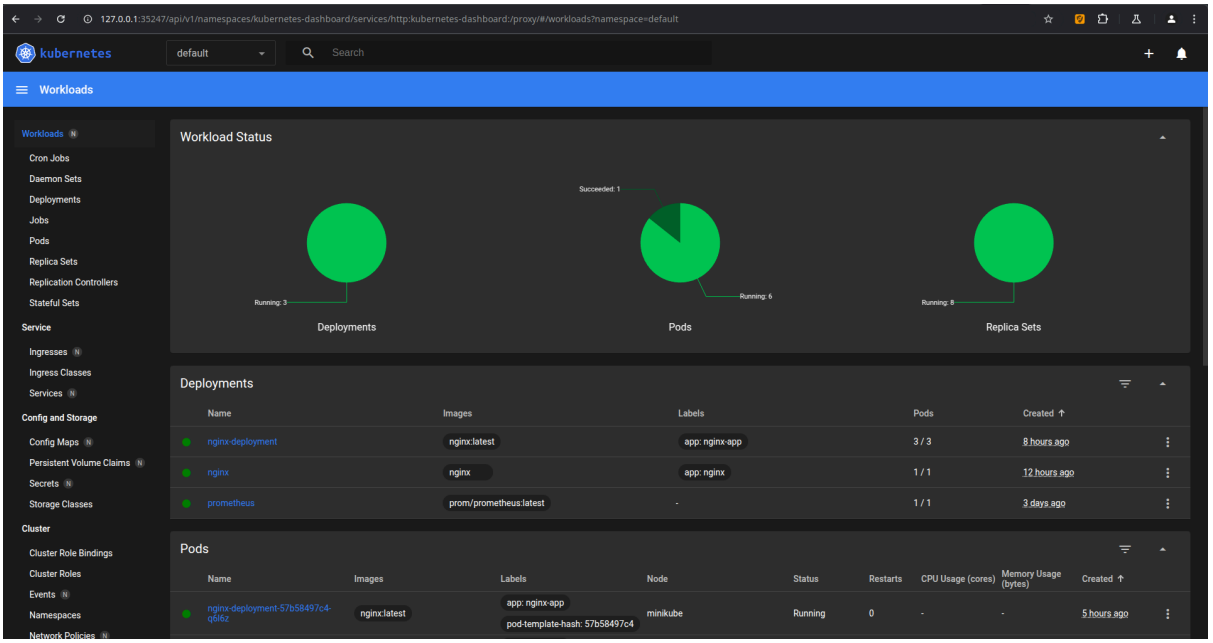
3. **Authenticate** :

- You can use the token from the default service account to log in:

```
kubectl -n kubernetes-dashboard create token admin-user
```

- Alternatively, use the Minikube token:

```
minikube dashboard
```



Task 2 - k8s Nodes

Étapes :

1.

Start a cluster local with Minikube :

```
minikube start
```

2. **Listen and describpe the nodes of cluster**

lists of nodes of my cluster :

```
kubectl get nodes
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   5d17h   v1.32.0
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$
```

get more detail of one node

```
kubectl describe node minikube
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl describe node minikube
Name:               minikube
Roles:              control-plane
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=minikube
                    kubernetes.io/os=linux
                    minikube.k8s.io/commit=dd5d320e41b5451cdf3c01891bc4e13d189586ed-dirty
                    minikube.k8s.io/name=minikube
                    minikube.k8s.io/primary=true
                    minikube.k8s.io/updated_at=2025_02_18T15_31_03_0700
                    minikube.k8s.io/version=v1.35.0
                    node-role.kubernetes.io/control-plane=
                    node.kubernetes.io/exclude-from-external-load-balancers=
Annotations:        kubeadm.alpha.kubernetes.io/cri-socket: unix:///var/run/cri-dockerd.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Tue, 18 Feb 2025 15:30:53 +0300
Taints:             <none>
Unschedulable:      false
Lease:
  HolderIdentity:  minikube
  AcquireTime:     <unset>
```

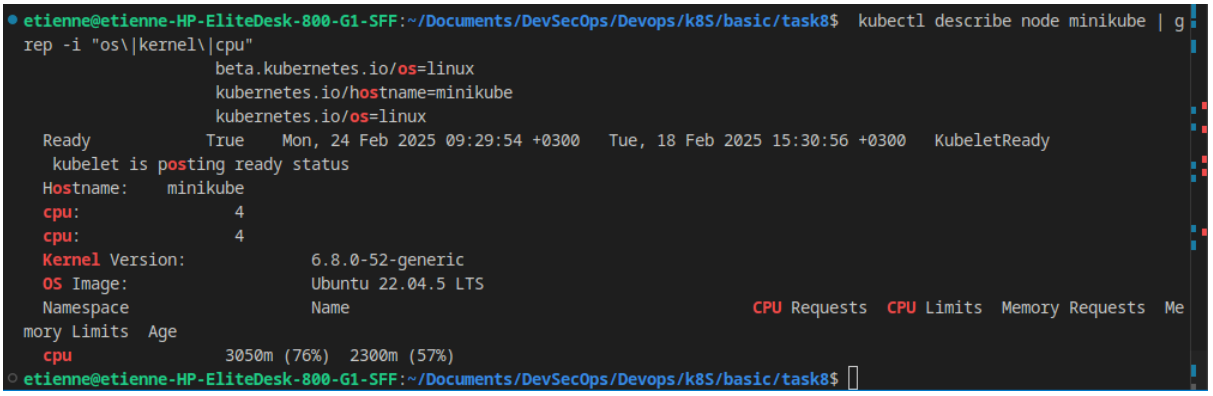
3. **Get more system information about a node**

get info about OS , CPU of node :

```
kubectl get nodes -o wide
```

check détails of OS and ressources CPU :

```
kubectl describe node minikube | grep -i "os\|kernel\|cpu"
```



Task 3 - k8s Pod

Objectif :

Create and manipulate a Kubernetes Pod that runs an application.

Steps :

1. **Understand Structure of Pod**

    - A Pod is a basic unit in Kubernetes. It contains one or more containers.
    - The specifications of a Pod are defined in a YAML file with the main fields :
        - `apiVersion`: Kubernetes API version
        - `kind`: resource type (here, Pod)
        - `metadata`:information about the Pod (name, labels, etc.)
        - `spec`: specifications, including containers and their parameters

Here is an example of a Pod running an Nginx container:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

Create this file, for example pod.yaml, and then apply it:

```
kubectl apply -f pod.yaml
```

Check Pod Deployment

List active Pods:

```
kubectl get pods
```

```
cpu                 3050m (76%)   2500m (57%)
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$  kubectl get pods
NAME                              READY   STATUS      RESTARTS        AGE
busybox                           0/1     Completed   0               2d21h
nginx                             1/1     Running     2 (3m4s ago)    2d20h
```

get more détails of Pod :

```
kubectl describe pod nginx
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$  kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Fri, 21 Feb 2025 12:54:06 +0300
Labels:           name=nginx-app
Annotations:      <none>
Status:           Running
IP:               10.244.0.43
IPs:
  IP:  10.244.0.43
Containers:
  nginx:
    Container ID:  docker://6ff2be162f1aa49ef9cab89ddb56eed03c864a5ed4acfe7d19b383a45500ac66
    Image:         nginx:latest
    Image ID:      docker-pullable://nginx@sha256:91734281c0ebfc6f1aea979cffeed5079cfe786228a71cc6f1f46a228cde6e34
    Port:          60/TCP
    Host Port:     0/TCP
    Command:
      /bin/sh
      -c
    Args:
      apt update && apt install -y iputils-ping && nginx -g 'daemon off;';
    State:          Running
      Started:      Mon, 24 Feb 2025 09:30:51 +0300
    Last State:     Terminated
      Reason:       Error
      Exit Code:    100
      Started:      Mon, 24 Feb 2025 09:28:54 +0300
      Finished:     Mon, 24 Feb 2025 09:30:25 +0300
    Ready:          True
    Restart Count:  2
    Limits:
      cpu:     500m
      memory:  128Mi
    Requests:
      cpu:        500m
      memory:     128Mi
```

Look at the container logs:

```
kubectl logs nginx
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl logs nginx

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:3 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 Packages [8792 kB]
Get:5 http://deb.debian.org/debian bookworm-updates/main amd64 Packages [13.5 kB]
Get:6 http://deb.debian.org/debian-security bookworm-security/main amd64 Packages [246 kB]
Fetched 9306 kB in 5s (1814 kB/s)
Reading package lists...
Building dependency tree...
Reading state information...
2 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

Open shell inside a Pod :

```
kubectl exec -it nginx -- /bin/sh
```

Check if Nginx work :

```
curl http://localhost
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl exec -it nginx -- /bin/sh
#  curl http://localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
#
```

Task 4 - k8s Service

A Service in Kubernetes allows to :

- Provide a stable IP to access Pods.
- Enable internal/external communication.
- Offer a load balancing mechanism.

Key fields in a YAML specification file of a Service :

- `apiVersion`: Version of the Kubernetes API.
- `kind`: Resource type (here Service).
- `metadata`: Name of the Service.
- `spec`:
  - `selector`: Labels used to identify the targeted Pods.
  - `ports`: Exposed ports (targetPort, port, nodePort if needed).
  - `type`: Service type (ClusterIP, NodePort, LoadBalancer).

---

2. Write a Service for your Pod and deploy it

Assume that your Pod uses an Nginx container.

Service Definition (`service.yaml`)

This Service exposes the application via an internal IP with ClusterIP :

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: my-nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Déploiement of Service

Apply the file with :

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

Check that the Service is createdé :

```
kubectl get services
kubectl describe service nginx-service
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl get services
NAME               TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP        5d18h
my-nginx-external  NodePort    10.109.224.41   <none>        80:30080/TCP   2d20h
nginx-service      NodePort    10.98.120.249   <none>        80:30090/TCP   2d21h
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl describe service nginx-serv
ice
Name:                     nginx-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.98.120.249
IPs:                      10.98.120.249
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30090/TCP
Endpoints:                10.244.0.45:80
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$
```

---

3. Check communication between Pods

List Pods and their IP addresses :

```
kubectl get pods -o wide
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl get pods -o wide
NAME                                READY   STATUS      RESTARTS       AGE     IP            NODE       NOMINATED NODE
READINESS GATES
busybox                             0/1     Completed   0              2d21h   10.244.0.12   minikube   <none>
<none>
nginx                               1/1     Running     2 (8m8s ago)   2d20h   10.244.0.43   minikube   <none>
<none>
nginx-5869d7778c-td9jp              1/1     Running     1 (11m ago)    2d21h   10.244.0.45   minikube   <none>
<none>
nginx-deployment-57b58497c4-nw2hc   1/1     Running     1 (11m ago)    2d14h   10.244.0.37   minikube   <none>
<none>
nginx-deployment-57b58497c4-q6l6z   1/1     Running     1 (11m ago)    2d14h   10.244.0.47   minikube   <none>
<none>
nginx-deployment-57b58497c4-x79pn   1/1     Running     1 (11m ago)    2d14h   10.244.0.41   minikube   <none>
<none>
prometheus-76f97bf89f-5n5rc         1/1     Running     2 (11m ago)    5d16h   10.244.0.42   minikube   <none>
<none>
```

Each Pod has an IP assigned by Kubernetes. You can test the DNS resolution between them :

```
kubectl exec -it nginx -- ping nginx-service
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl exec -it nginx -- ping ngin
x-service
PING nginx-service.default.svc.cluster.local (10.98.120.249) 56(84) bytes of data.
```

If everything works fine, the Pod should be able to reach the Service by its DNS name.

---

4. Deleting and recreating a Pod

delete the Pod :

```
kubectl delete pod nginx
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task8$ kubectl delete pod nginx
pod "nginx" deleted
```

Kubernetes will automatically recreate a new Pod if a Deployment is used. Check the IP of the new Pod and make sure it can still be reached by the Service :

```
kubectl get pods -o wide
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task4$ kubectl get pods -o wide
NAME                                READY   STATUS      RESTARTS      AGE     IP            NODE       NOMINATED NODE
EADINESS GATES
busybox                             0/1     Completed   0             2d21h   10.244.0.12   minikube   <none>
none>
nginx                               1/1     Running     0             109s    10.244.0.48   minikube   <none>
none>
nginx-5869d7778c-td9jp              1/1     Running     1 (18m ago)   2d21h   10.244.0.45   minikube   <none>
none>
nginx-deployment-57b58497c4-nw2hc   1/1     Running     1 (17m ago)   2d14h   10.244.0.37   minikube   <none>
none>
nginx-deployment-57b58497c4-q6l6z   1/1     Running     1 (17m ago)   2d14h   10.244.0.47   minikube   <none>
none>
nginx-deployment-57b58497c4-x79pn   1/1     Running     1 (17m ago)   2d14h   10.244.0.41   minikube   <none>
none>
prometheus-76f97bf89f-5n5rc         1/1     Running     2 (18m ago)   5d16h   10.244.0.42   minikube   <none>
none>
```

The Pod IP will have changed, but the connection via the Service should still work.

---

5. Learn about LoadBalancer and NodePort

Kubernetes offers several types of Services :

- **ClusterIP** (défaut) : Accessible only from the cluster
- **NodePort** Exposes the Service on a specific port of each node.
- **LoadBalancer** : Creates an external Load Balancer (requires a cloud provider like AWS, GCP, Azure).

You can test a NodePort by modifying `service.yaml` :

type: NodePort

Apply the changes :

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

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task4$ kubectl get services
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP          5d18h
my-nginx-external  NodePort    10.109.224.41    <none>        80:30080/TCP     2d20h
nginx-service      NodePort    10.98.120.249    <none>        80:30090/TCP     2d21h
```

The assigned port will be in the range **30000-32767**.
You can access it via `http://10.98.120.249:30090`.

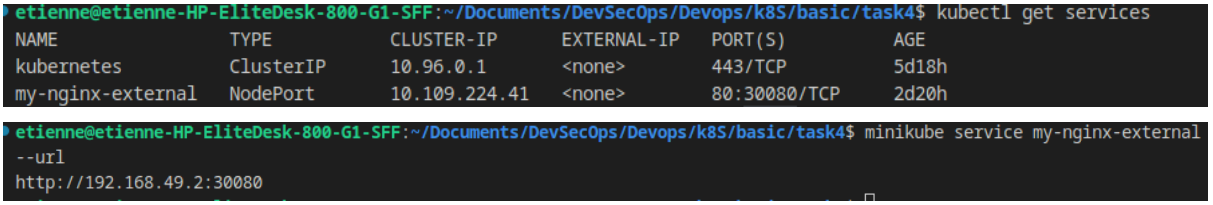
---

6. Deploy an External Service to access from the local host

If you want to test access from your computer:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-external
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
  type: NodePort
```

```
kubectl apply -f service.yaml
minikube service my-nginx-external --url
```



Bonus: Labels et Sélecteurs

Labels in Kubernetes

Labels are tags attached to Kubernetes resources (Pods, Services, Deployments, etc.).

Example of a Pod with a label:

```yaml
metadata:
  labels:
    app: nginx
```

The **Service** selects the Pods that have this label:

```yaml
selector:
  app: nginx
```

Test 1: No Pod Matches the Label

If a **Service** has this selector:

```
selector:
  app: missing-label
```

But no Pod has this label, then the command:

kubectl get endpoints my-nginx-service

Will return **0 endpoints**, meaning that no Pod is accessible through this Service.

Test 2: Multiple Pods Match the Label

If you have multiple Pods with the same label:

```
kubectl scale deployment nginx --replicas=3
kubectl get endpoints nginx-service
```

The **Service** will automatically distribute traffic between these Pods. 🚀

Task 5

---

1. Identifying the Required Fields in a Deployment Spec

The key fields in a **Deployment** manifest include :

- **apiVersion** : For example, `apps/v1`
- **kind** : Must be `Deployment`
- **metadata** : Contains the name, labels, etc.
- **spec** :
    - **replicas** : Number of desired Pods.
    - **selector** : Criteria (labels) to associate the Deployment with the Pods.
    - **template** : The Pod template that will be created, which itself consists of :
        - **metadata** (labels, annotations)
        - **spec** (list of containers, ports, volumes, environment variables, etc.)
    - *(Optional)* **strategy**: Defines the update strategy (e.g., RollingUpdate) with parameters like maxSurge and maxUnavailable..
    - *(Optional)* **revisionHistoryLimit**: Number of old versions to retain for rollback.

minimal YAML Deployment :

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: nginx:latest
        ports:
        - containerPort: 80
```

---

2. Clean Up Previous Pod Manifests and Deploy the Application

Before creating the **Deployment**, make sure to delete or clean up any old Pods or manifests (created via kubectl run or individual YAML manifests).

Then, create a YAML file (e.g., deployment.yaml) with the Deployment manifest adapted to your application.

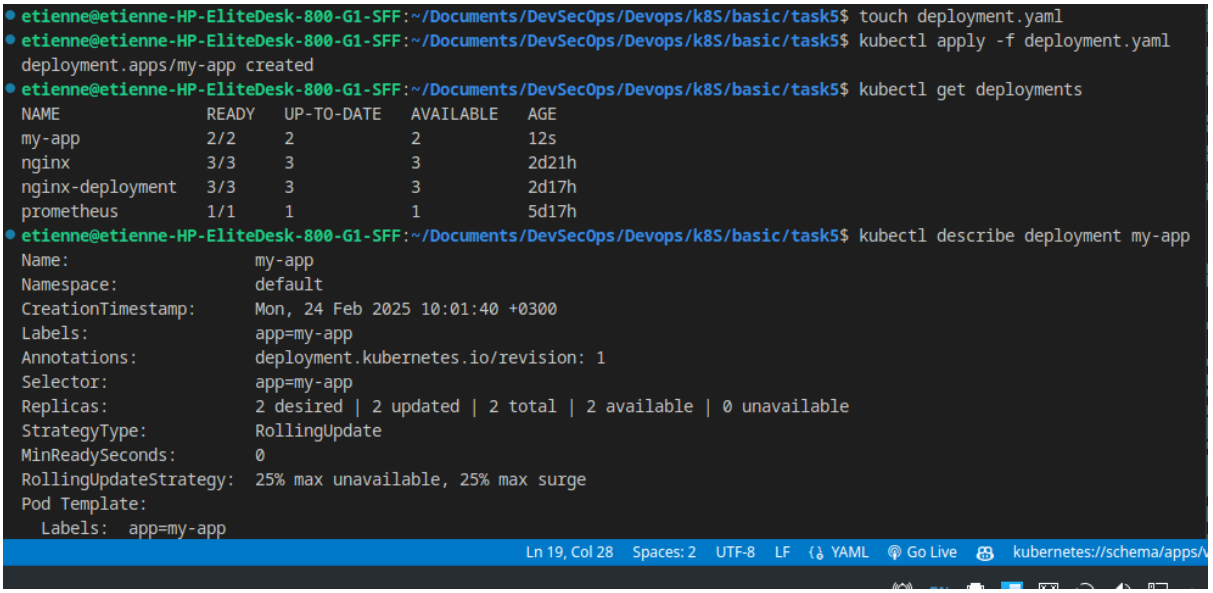
Apply it using:

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

3. List and Describe Deployments

To verify that the **Deployment** has been created and check its status, use:

```
kubectl get deployments
kubectl describe deployment my-app
```



The describe command provides details about the **ReplicaSet**, recorded events (useful for debugging), and update history.

4. Scale the Deployment to Three Replicas

To increase the number of Pods to **3**, you can either edit the YAML file:

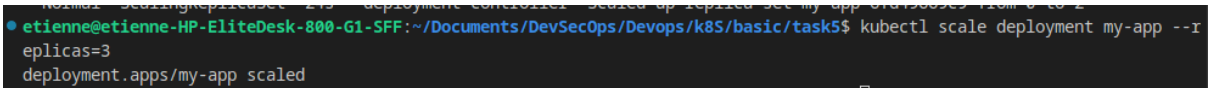
```
spec:
  replicas: 3
```

Then apply the update:

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

Or use the command:

```
kubectl scale deployment my-app --replicas=3
```

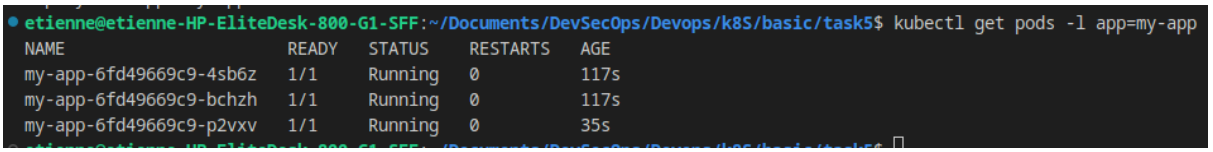


5. Access the Shell and Logs of Pods via Deployment Labels

The Pods created by the Deployment inherit the **labels** defined in the template.

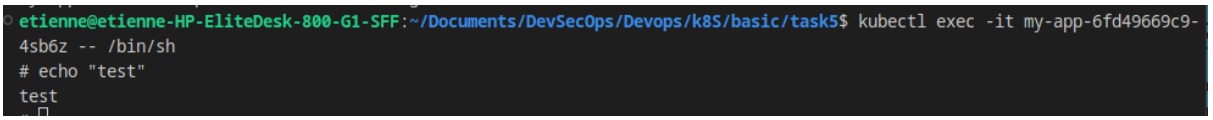
- **List Pods with a specific label**:

```
kubectl get pods -l app=my-app
```



- **Access a Pod's shell (if the image supports it)**:

```
kubectl exec -it my-app-6fd49669c9-4sb6z -- /bin/sh
```



- **View logs of a specific Pod**:

```
kubectl logs my-app-6fd49669c9-4sb6z
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl logs my-app-6fd49669c9-4sb6
z
/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
2025/02/24 07:01:45 [notice] 1#1: using the "epoll" event method
2025/02/24 07:01:45 [notice] 1#1: nginx/1.27.4
2025/02/24 07:01:45 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/02/24 07:01:45 [notice] 1#1: OS: Linux 6.8.0-52-generic
2025/02/24 07:01:45 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/02/24 07:01:45 [notice] 1#1: start worker processes
2025/02/24 07:01:45 [notice] 1#1: start worker process 29
2025/02/24 07:01:45 [notice] 1#1: start worker process 30
2025/02/24 07:01:45 [notice] 1#1: start worker process 31
2025/02/24 07:01:45 [notice] 1#1: start worker process 32
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$
```

- **View logs using labels (e.g., for all matching Pods)**:

```
kubectl logs -l app=my-app --tail=50
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl logs -l app=my-app --tail=5
0
/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
2025/02/24 07:01:45 [notice] 1#1: using the "epoll" event method
2025/02/24 07:01:45 [notice] 1#1: nginx/1.27.4
2025/02/24 07:01:45 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/02/24 07:01:45 [notice] 1#1: OS: Linux 6.8.0-52-generic
2025/02/24 07:01:45 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/02/24 07:01:45 [notice] 1#1: start worker processes
2025/02/24 07:01:45 [notice] 1#1: start worker process 29
2025/02/24 07:01:45 [notice] 1#1: start worker process 30
2025/02/24 07:01:45 [notice] 1#1: start worker process 31
```

6. Update the Application Configuration and Observe the Behavior

Modify part of the **Deployment manifest** (e.g., update the image version or environment variables).

For example, change the image:

image: my-app-image:2.0

Then apply the changes:

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

Use --watch to follow the update process:

```
kubectl get pods --watch
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl get pods --watch
NAME                                READY   STATUS      RESTARTS      AGE
busybox                             0/1     Completed   0             2d22h
my-app-6fd49669c9-4sb6z             1/1     Running     0             13m
my-app-6fd49669c9-bchzh             1/1     Running     0             13m
my-app-6fd49669c9-p2vxv             1/1     Running     0             12m
nginx                               1/1     Running     0             32m
nginx-5869d7778c-k6gpz              1/1     Running     0             20m
nginx-5869d7778c-q6vsr              1/1     Running     0             20m
nginx-5869d7778c-td9jp              1/1     Running     1 (48m ago)   2d21h
nginx-deployment-57b58497c4-nw2hc   1/1     Running     1 (48m ago)   2d14h
nginx-deployment-57b58497c4-q6l6z   1/1     Running     1 (48m ago)   2d14h
nginx-deployment-57b58497c4-x79pn   1/1     Running     1 (48m ago)   2d14h
prometheus-76f97bf89f-5n5rc         1/1     Running     2 (48m ago)   5d17h
```

You will see Kubernetes perform a **rolling update**, gradually replacing old Pods with new ones.

7. Perform a Rollback to the Previous Version

If the update causes issues, rollback to the previous version with:

```
kubectl rollout undo deployment my-app
```

To check the update history:

```
kubectl rollout history deployment my-app
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl rollout history deployment
my-app
deployment.apps/my-app
REVISION  CHANGE-CAUSE
1         <none>
```

8. Define CPU and Memory Requests & Limits

Set resource constraints in the **containers** section of the Deployment template:

```
resources:
  requests:
    cpu: "100m"
    memory: "128Mi"
  limits:
    cpu: "500m"
    memory: "256Mi"
```

Behavior When Limits Are Exceeded

- **CPU Throttling**: If the app uses more than 500m (0.5 CPU), it will be slowed down.

- **Memory OOM Kill**: If the app exceeds 256Mi, Kubernetes will terminate the Pod (OOMKilled) and restart it based on the restart policy.

To test this, generate a high load on the app and observe **throttling** or **OOM kills** using:

kubectl top pods

Task 6

---

1. Fields needed in a Secret specification

A Secret manifest contains the following essential fields:

- **apiVersion** : Usually `v1`.
- **kind** : Must be `Secret`.
- **metadata** : Name, labels, and annotations.
- **type** : The type of the secret (often Opaque for a generic secret).
- **data** : A dictionary of key/value pairs where each value must be Base64 encoded.
  → Alternatively, you can use stringData which allows you to specify the values in plain text; Kubernetes will take care of the encoding.

---

2. Creating and Applying a Secret Manifest

For example, let's create a secret.yaml file to store a username and password:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  # "myuser" encodé en base64 : echo -n "myuser" | base64 → bXl1c2Vy
  username: bXl1c2Vy
  # "mypassword" encodé en base64 : echo -n "mypassword" | base64 →
bXlwYXNzd29yZA==
  password: bXlwYXNzd29yZA==
```

You can also use the stringData section to avoid doing the encoding manually:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
stringData:
  username: myuser
  password: mypassword
```

apply secret to the cluster :

```
kubectl apply -f secret.yaml
```

---

3. Retrieve and describe the Secret with kubectl

List your secrets:

```
kubectl get secrets
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl get secrets
NAME          TYPE     DATA   AGE
json-secret   Opaque   1      2d16h
my-secret     Opaque   2      2d16h
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$
```

describe secret

```
kubectl describe secret my-secret
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ kubectl describe secret my-secret
Name:         my-secret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  10 bytes
username:  6 bytes
```

---

4. Decode Secrets

To check the value of a field, you can copy the Base64 value and decode it locally. For example:

```
echo 'bXl1c2Vy' | base64 --decode
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$ echo 'bXl1c2Vy' | base64 --decode
myuseretienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task5$
```

---

5. Update your Deployment to reference the Secret as an environment variable

Modify your Deployment manifest to inject the Secret data into the containers. For example, in your deployment.yaml file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
   app: nginx-app
spec:
 replicas: 3  # Deploy 2 instances of the pod
 selector:
   matchLabels:
     app: nginx-app  # Ensures Deployment manages the right Pods
 template:
   metadata:
     labels:
       app: nginx-app
   spec:
     containers:
     - name: nginx
       image: nginx:latest
       command: ["/bin/sh", "-c"]
       args:
         - apt update && apt install -y iputils-ping  net-tools &&
nginx -g 'daemon off;';
       resources:
         limits:
           memory: "128Mi"
           cpu: "600m"
       ports:
         - containerPort: 60
       env:
       - name: APP_USERNAME
         valueFrom:
           secretKeyRef:
             name: my-secret
```

```
            key: username
      - name: APP_PASSWORD
        valueFrom:
          secretKeyRef:
            name: my-secret
            key: password
```

Apply the update :

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

---

6. Verify that the Secret is available in the Pod

You can run a command in one of the pods to display the environment variables:

List the pods with the Deployment label:

```
kubectl get pods -l app=nginx-app
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl get pods -l app=nginx-app
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-d45596dc8-828xc    1/1     Running   0          80s
nginx-deployment-d45596dc8-gf69f    1/1     Running   0          117s
nginx-deployment-d45596dc8-rs5lk    1/1     Running   0          2m2s
```

go to the shell of pod

```
kubectl exec -it nginx-deployment-d45596dc8-rs5lk -- /bin/sh
```

check the variable :

```
 echo $APP_USERNAME
echo $APP_PASSWORD
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$  kubectl exec -it nginx-deployment-
d45596dc8-rs5lk -- /bin/sh
#  echo $APP_USERNAME
echo $APP_PASSWORDmyuser
#
mypassword
#
```

Bonus: Creating a Secret from a JSON File

Suppose you have a file named credentials.json containing:

```
{
  "username": "jsonuser",
  "password": "jsonpass"
}
```

You can create a **Secret** from this file using a similar approach to ConfigMaps (but specifically for secrets):

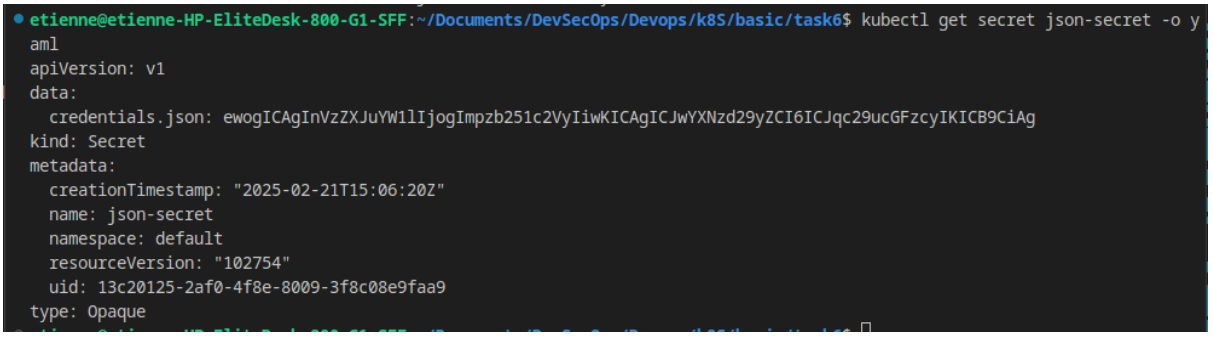
```
kubectl create secret generic json-secret --from-file=credentials.json
```

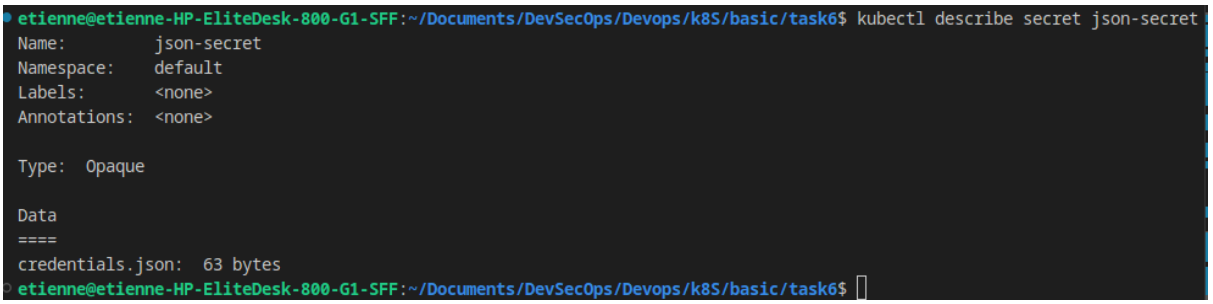
This command creates a **secret** named json-secret with a key credentials.json, where the value is the file's content encoded in **Base64**.

Verifying the Secret

To check the created secret:

```
kubectl get secret json-secret -o yaml
kubectl describe secret json-secret
```





To **decode** the stored data, retrieve the Base64 value and use:

```
echo 'encoded_value' | base64 --decode
```

Injecting Secret Values into a Deployment

There are two ways to inject these secrets into your **Deployment**:

1️⃣ Direct Injection into Environment Variables

If you want to extract specific keys from the JSON file, it's better to create the **secret** with multiple key-value pairs. You can use a tool like jq to extract values:

```
kubectl create secret generic json-secret \
  --from-literal=username=$(jq -r '.username' credentials.json) \
  --from-literal=password=$(jq -r '.password' credentials.json)
```

Then, reference them in your **Deployment** as environment variables.

2️⃣ Mounting the Secret as a Volume

You can mount the **Secret** as a volume inside the Pod so that the JSON file is accessible from within the container.

🚀 This allows Kubernetes applications to securely manage sensitive data like credentials!

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: my-app-image:1.0
        volumeMounts:
        - name: json-secret-volume
```

```
        mountPath: /etc/secrets
        readOnly: true
    volumes:
    - name: json-secret-volume
      secret:
        secretName: json-secret
```

1. In this case, the /etc/secrets/credentials.json file will be available in the pod and you can use it in your application.

Task 7

---

1. The Required Fields in a ConfigMap Manifest

The essential elements of a **ConfigMap** manifest are:

- **apiVersion**: usually v1
- **kind**: must be ConfigMap
- **metadata**: includes the name, labels, and optional annotations
- **data**: a dictionary of key-value pairs

→ You can also use **stringData** to provide unencoded values (Kubernetes will handle the Base64 conversion).

Minimal Example:

(Example YAML not provided in the text.)

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
data:
  APP_ENV: "production"
  APP_DEBUG: "false"
```

---

2. Modifying the Deployment to Inject Environment Variables

To allow your application to receive this configuration via **environment variables**, update the **Deployment** manifest by adding a reference to the **ConfigMap**.

For example, in the **envFrom** section of the container:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
```

```
  labels:
    app: my-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app-container
        image: my-app-image:1.0
        ports:
        - containerPort: 80
        envFrom:
        - configMapRef:
            name: my-app-config
```

---

3. Creating a ConfigMap with Key/Value Pairs

Create a YAML file (e.g., configmap.yaml) that contains the desired configuration data:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config
data:
  APP_ENV: "production"
  APP_DEBUG: "false"
```

Apply it using:

```
kubectl apply -f configmap.yaml
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl get configmap
NAME               DATA   AGE
kube-root-ca.crt   1      5d19h
my-app-config      2      2d15h
my-app-config-json 1      2d15h
```

Update your **Deployment** (as shown in step 2) to use this **ConfigMap**.

4. Creating a JSON Configuration File

Create a file named, for example, config.json, containing your JSON data:

```
{
  "log_level": "info",
  "max_connections": 100
}
```

---

5. Creating a ConfigMap from a JSON File

You have two options:

Option A – Directly in the YAML manifest:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-app-config-json
data:
  config.json: |
    {
      "log_level": "info",
      "max_connections": 100
    }
```

Option B – Using the Command Line:

```
kubectl create configmap my-app-config-json --from-file=config.json
```

6. Updating the Deployment to Mount the ConfigMap as a Volume

To make the JSON file accessible within the container's filesystem, modify your **Deployment** to add a **volume** and a **volumeMount**:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
```

```
  labels:
    app: my-app
spec:
  replicas: 2
```

```
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
      - name: my-app-container
        image: my-app-image:1.0
        ports:
        - containerPort: 80
        envFrom:
        - configMapRef:
            name: my-app-config
        volumeMounts:
        - name: config-volume
          mountPath: /etc/config   # Le chemin où sera monté le fichier
JSON
      volumes:
      - name: config-volume
        configMap:
          name: my-app-config-json
```

apply the changes with :

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

---

7. Check ConfigMap with kubectl

To view the contents of the ConfigMap and verify that it is stored in clear text, use

```
kubectl get configmap my-app-config -o yaml
kubectl describe configmap my-app-config
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl get configmap my-app-config
 -o yaml
apiVersion: v1
data:
  APP_DEBUG: "false"
  APP_ENV: production
kind: ConfigMap
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"APP_DEBUG":"false","APP_ENV":"production"},"kind":"ConfigMap","metadata":{"annotations"
:{},"name":"my-app-config","namespace":"default"}}
  creationTimestamp: "2025-02-21T16:02:44Z"
  name: my-app-config
  namespace: default
  resourceVersion: "105281"
  uid: 72474e7b-9139-4677-8ab5-111aaf925c73
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl describe configmap my-app-c
onfig
Name:         my-app-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
APP_DEBUG:
----
false

APP_ENV:
----
production


BinaryData
====

Events:  <none>
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$
```

.

---

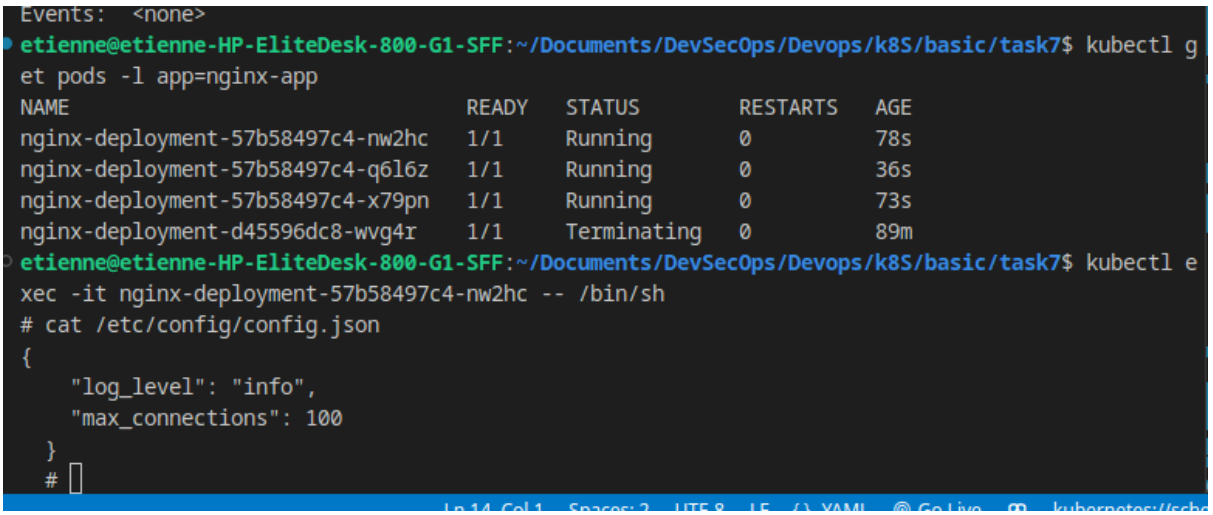
8. Verification in the container

To verify that the JSON file has been mounted, select a pod and open a shell:

```
kubectl get pods -l app=nginx-app
kubectl exec -it <pod-name> -- /bin/sh
```

Once in the shell, display the contents of the mounted file:

```
cat /etc/config/config.json
```


Events: <none>
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task7$ kubectl g
et pods -l app=nginx-app
NAME   READY   STATUS   RESTARTS   AGE
nginx-deployment-57b58497c4-nw2hc   1/1   Running   0   78s
nginx-deployment-57b58497c4-q6l6z   1/1   Running   0   36s
nginx-deployment-57b58497c4-x79pn   1/1   Running   0   73s
nginx-deployment-d45596dc8-wvg4r   1/1   Terminating   0   89m
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task7$ kubectl e
xec -it nginx-deployment-57b58497c4-nw2hc -- /bin/sh
# cat /etc/config/config.json
{
    "log_level": "info",
    "max_connections": 100
  }
  #

Task 8

---

1. Required Fields for a Namespace Manifest

A **Namespace** manifest is very simple. The essential elements are:

- **apiVersion**: v1
- **kind**: Namespace
- **metadata**: Contains at least the **name** field (optionally, labels or annotations).

Minimal Example:

(Example YAML not provided in the text.)

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: app1-namespace
```

---

2. Creating Two Namespaces in the Cluster

You can create two separate YAML files or a single combined file. For example:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: app1-namespace
---
apiVersion: v1
kind: Namespace
metadata:
  name: app2-namespace
```

Apply the config :

```
kubectl apply -f namespaces.yaml
```

---

3. Retrieve and describe your Namespaces with kubectl

Lists of Specifics :

```
kubectl get namespaces
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl get namespaces
NAME                STATUS   AGE
app1-namespace      Active   2d15h
app2-namespace      Active   2d15h
```

Description of a specifics Namespace :

```
kubectl describe namespace app1-namespace
kubectl describe namespace app2-namespace
```

```
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$ kubectl describe namespace app1-nam
espace
kubectl describe namespace app2-namespace
Name:         app1-namespace
Labels:       kubernetes.io/metadata.name=app1-namespace
Annotations:  <none>
Status:       Active

No resource quota.

No LimitRange resource.
Name:         app2-namespace
Labels:       kubernetes.io/metadata.name=app2-namespace
Annotations:  <none>
Status:       Active

No resource quota.

No LimitRange resource.
etienne@etienne-HP-EliteDesk-800-G1-SFF:~/Documents/DevSecOps/Devops/k8S/basic/task6$
```

---

4. Deploying Two Different Applications in Separate Namespaces

You can deploy, for example, an **Nginx** application in app1-namespace and an **Apache** application in app2-namespace.

Here is an example of a combined manifest:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: app1-namespace
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
  namespace: app2-namespace
  labels:
    app: apache
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - containerPort: 80
```

apply the deployment ：

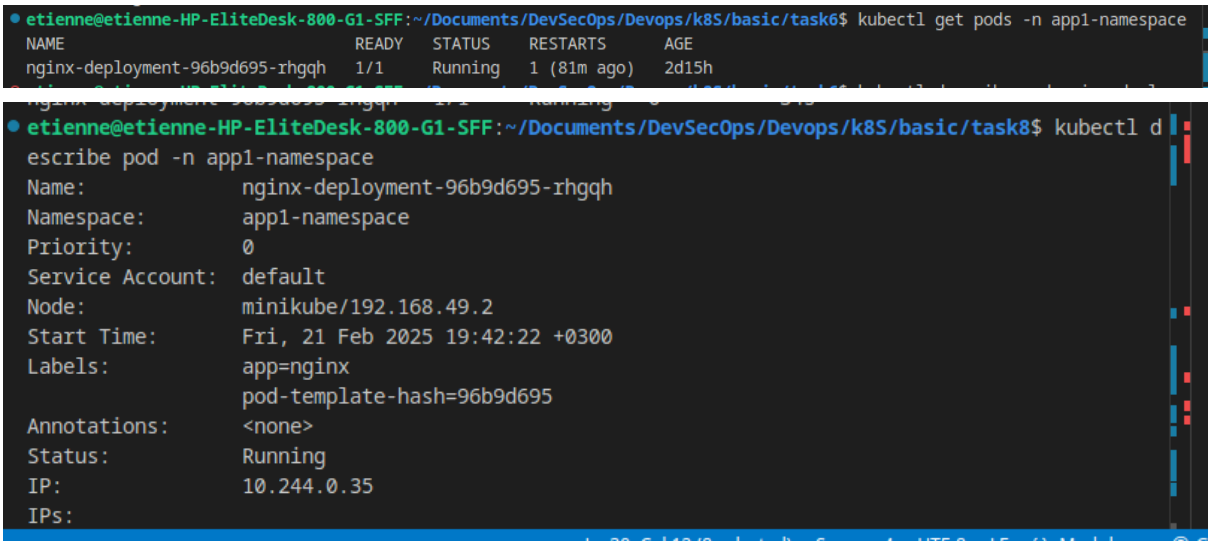
```
kubectl apply -f deployments.yaml
```

---

5. Récupérer et décrire les Pods dans différents Namespaces

To list pods in the app1-namespace namespace:

```
kubectl get pods -n app1-namespace
```



---

6. Visibility and Connectivity Between Resources in Different Namespaces

By default, **Namespaces** isolate resources.

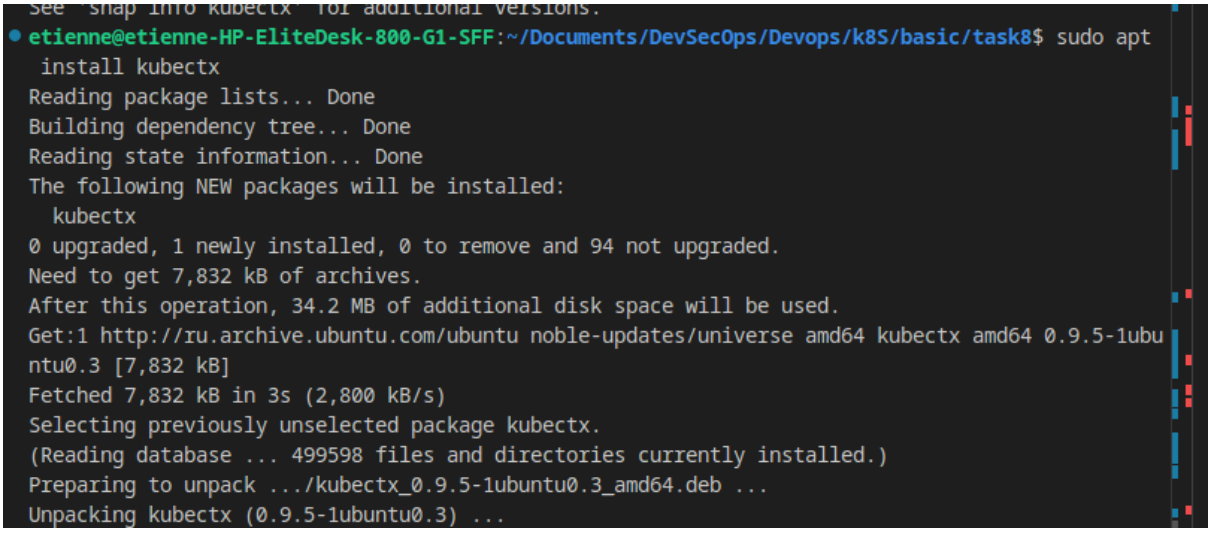
- You **cannot list** resources from another Namespace unless you use the -n flag or --all-namespaces.
- To connect two services in different **Namespaces**, you must use **fully qualified DNS names** (e.g., service-name.namespace.svc.cluster.local) or configure specific routing.

This means that, by default, an application in app1-namespace **will not automatically have access** to a resource (Pod, Service) in app2-namespace without explicit configuration.

Bonus: Using kubectx/kubens

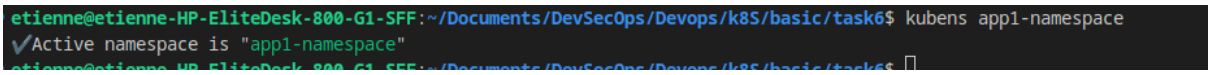
The tools kubectx and kubens allow you to quickly switch between contexts or namespaces without needing to specify the -n flag in each command.

You can install them from GitHub: github.com/ahmetb/kubectx.



For example, to switch to the app1-namespace namespace:

```
kubens app1-namespace
```



Once the namespace is changed, you can simply run:

```
kubectl get pods
```

to view the pods in the current namespace.

Similarly, you can use kubectx to switch between contexts if you're managing multiple clusters.

In Summary, This Task Allows You To:

- Create and isolate environments via namespaces.
- Deploy applications in different namespaces.
- View and monitor resources using the -n flag.
- Use tools like kubens to make multi-namespace management easier.

Rancher for Centralized Kubernetes Cluster Management in Production

Context and Objectives

In a complex production environment, having a centralized graphical interface allows you to:

- Quickly visualize the global state of the cluster (nodes, pods, services, deployments, etc.).
- Manage and control resources (adding/removing clusters, updating workloads, managing users).
- Monitor performance and metrics in real-time.
- Simplify multi-cluster management and security administration.

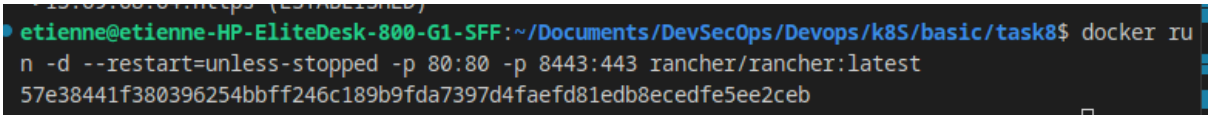
Rancher is one of the most popular and comprehensive solutions to meet these needs.

1. Rancher Installation

Installation with Docker (Quick method for a test environment)

You can install Rancher by simply running a Docker container. For example, on a Linux machine:

```
docker run -d --restart=unless-stopped -p 80:80 -p 443:443
rancher/rancher:latest
```



Production Installation

For a production setup, it's recommended to install Rancher on a Kubernetes cluster itself, for example, using Helm:

1. Add the Rancher Helm repository:

```
helm repo add rancher-latest
https://releases.rancher.com/server-charts/latest
helm repo update
```

2. Install Rancher in a dedicated namespace (e.g., cattle-system):

```
kubectl create namespace cattle-system
helm install rancher rancher-latest/rancher \
  --namespace cattle-system \
  --set hostname=rancher.mycompany.com
```

You'll need to configure DNS so the hostname points to the cluster's IP or use an Ingress Controller.

2. Access and Practical Actions via Rancher Interface

Once Rancher is installed, you can access the interface via your browser at an address like https://rancher.mycompany.com.

Typical Actions Available in Rancher:

- **Add and Manage Clusters**

Rancher allows you to add new clusters (local or cloud) and group them into projects to better organize your resources.

- **Visualize and Control Resources**

You can view the real-time status of nodes, pods, services, and deployments. The interface provides dashboards with metrics (CPU, memory, etc.) and detailed logs.

- **Deploy Workloads**

From the Rancher interface, you can create deployments, StatefulSets, or jobs without typing kubectl commands. Simply fill out forms to define replicas, images, environment variables, etc.

- **Updates and Rollbacks**

Rancher provides the ability to easily update deployments and rollback to a previous version if needed. You can view the update history and initiate a rollback with a few clicks.

- **User and Role Management**

You can define access policies for users or teams, restricting access to specific clusters or projects.

- **Access to Shell Interface**

Rancher includes a web terminal that lets you run kubectl commands directly from the interface, making checks and troubleshooting easier.

- **Context Switching Tools**

Rancher simplifies multi-cluster management by clearly displaying which cluster or project is currently selected and allows you to switch between them easily.