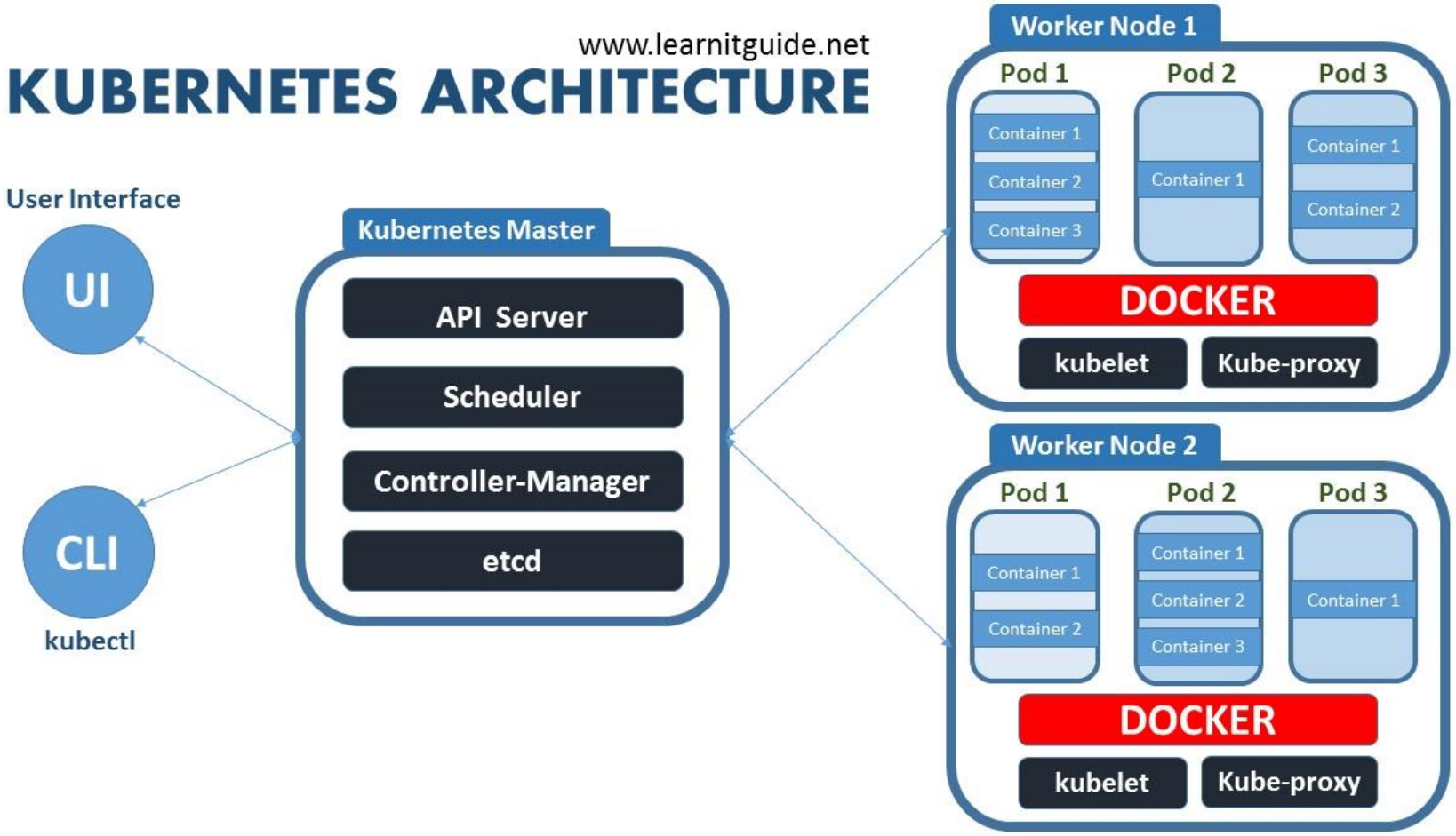
DEPLOIEMENT D'APPLICATION .NET CORE SOUS KUBERNETES

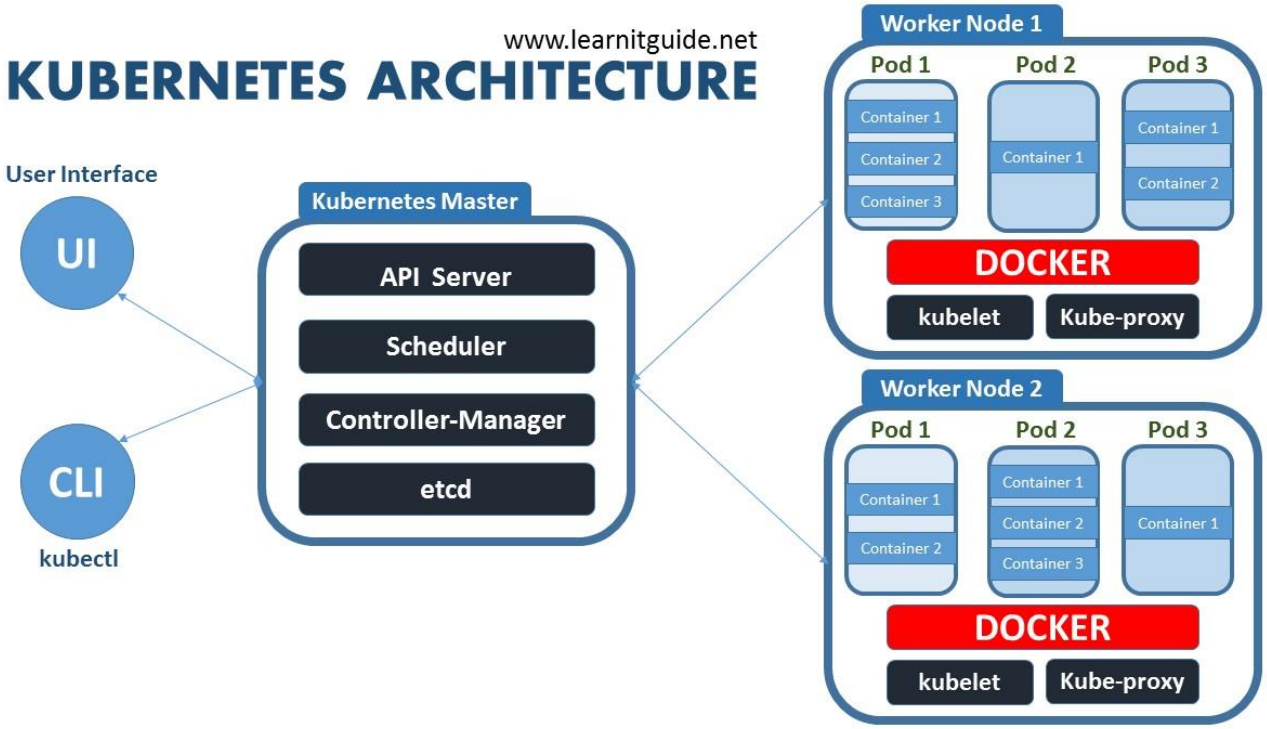
Architecture d'un cluster Kubernetes (multi-nodes)



Terminologies architecturales

- **Nodes:** Machine d'execution (Machine physique ou virtuelle)
  - **Master node:** machine principale qui gère les autres nodes
  - **Worker nodes:** machines executant nos applications

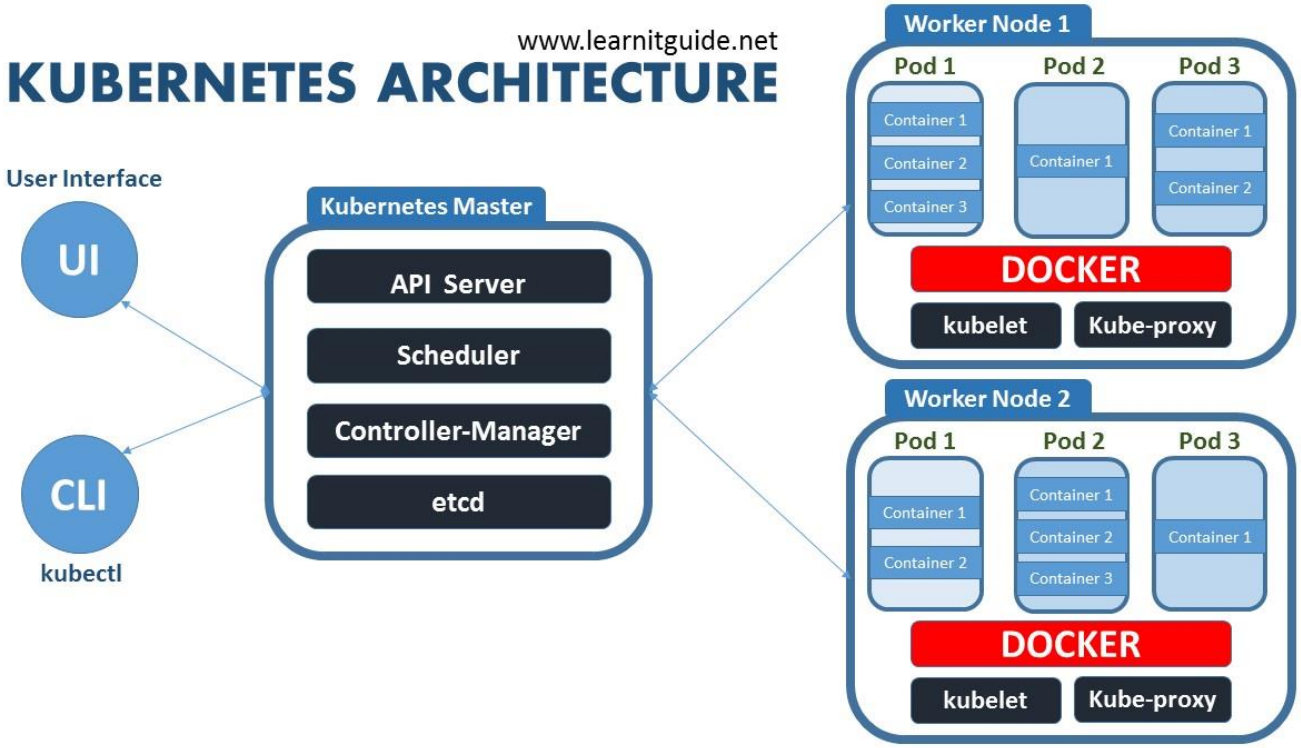
- **Pods:** Unité basique d'exécution sous kubernetes
  - créé ou déployé séparément (et automatiquement)
  - contenu dans les nodes
  - Environnement pour un groupe de containers



Terminologies architecturales

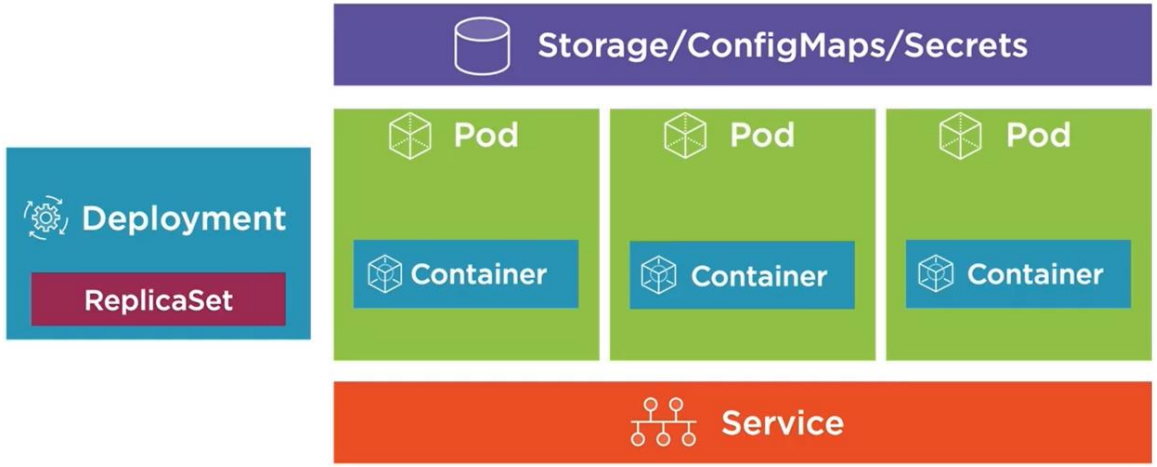
- **Containers:** Ressource d'exécution isolée pour un service

- **API Server, Scheduler, Controller-Manager, etcd**: Daemon exécuté sur le master node pour la gestion du cluster Kubernetes
- **Kubelet, Kube-proxy**: Daemon sur les workers nodes pour l'exécution et l'adressage réseau

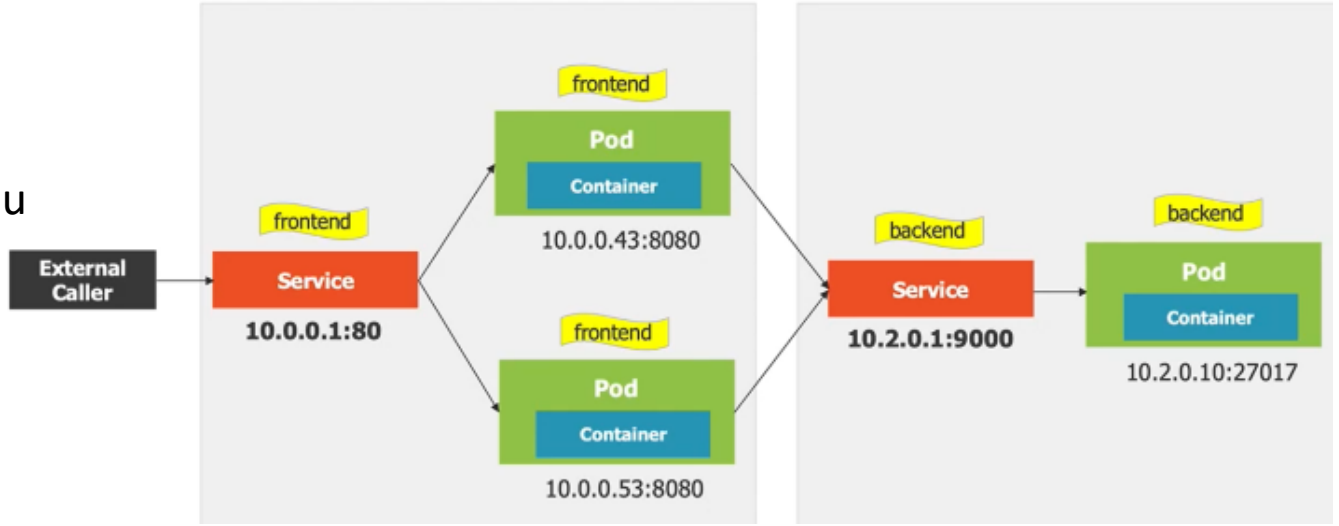


Terminologies fonctionnelles

- **Deployment et replicaSet:**
  - manière declarative de définir l'état des pods et de les gérer.
  - Deployment utilise les replicaSets
  - Assure que les pods marchent parfaitement selon l'état défini
  - Gère le scaling (en créant, supprimant, replaçant les pods)

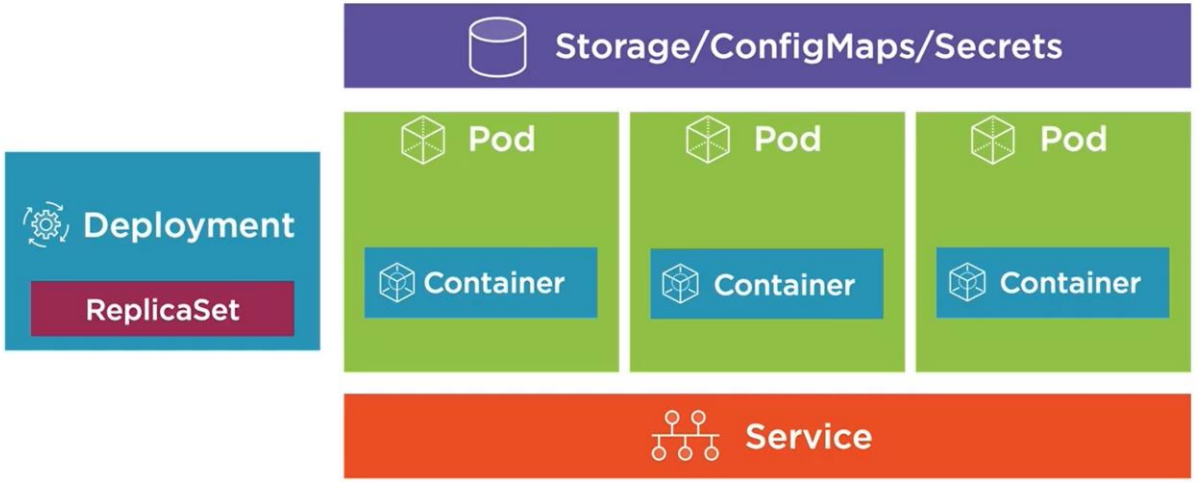


- **Service:**
  - Définit un point d'entrée pour l'accès à un ou plusieurs Pods (Pods crées et détruits dynamiquement)
  - Abstraction de l'adressage IP des pods
  - Load balancing entre les pods
  - Plusieurs types: clusterIP, nodePort, LoadBalancer,



Terminologies fonctionnelles

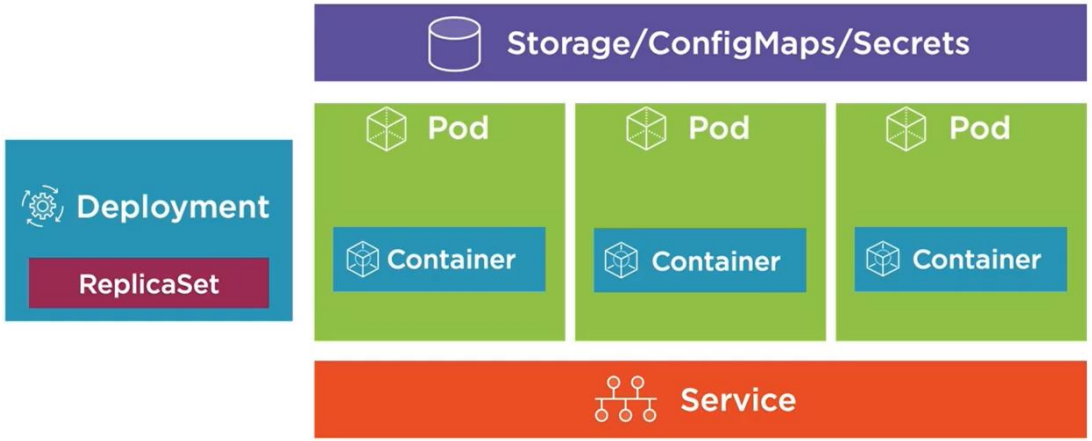
- **Stockage**:
  - **Volume:** Utilisé pour stocker les données utiles au Pods/Containers
  - **Volume type:**
    - **emptyDir:** Stockage au sein d'un Pod, partagé entre les containers du meme Pod, détruit quand le pod est supprimé
    - **hostPath:** Stockage monté sur le systeme de fichier du node contenant les Pods
    - **nfs (network file system):** monté dans un Pod mais partagé

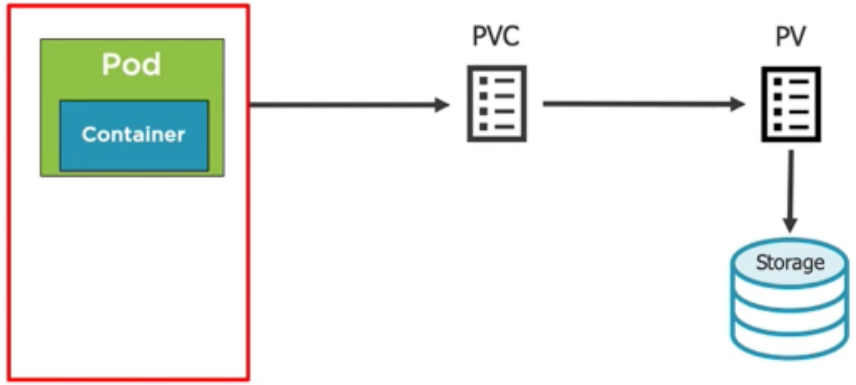


- **ConfigMaps:** stockage centralisé des configurations (clé/valeur) pour les Pods/Containers
- **Secrets:** stockage sécurisé de configuration sensitive requis par les Pods/Containers

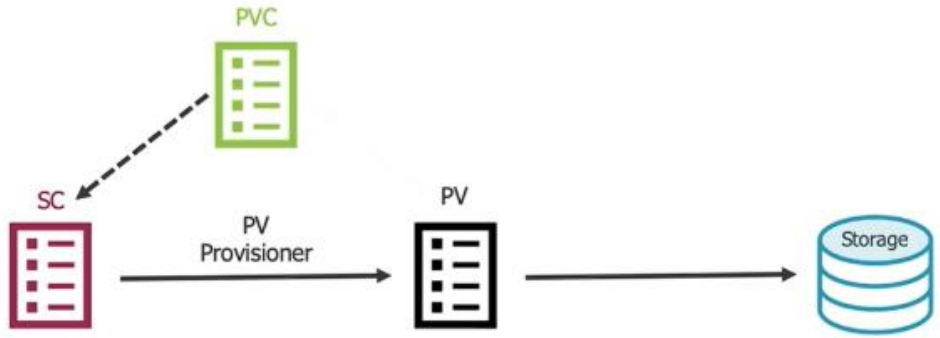
Terminologies fonctionnelles

- **Stockage**:
  - **PersistentVolume (PV):** Unité de stockage (NAS: Network-Attached Storage) centralisé au sein du cluster independent du cycle de vie des Pods
  - **PersistentVolumeClaim (PVC):** Requete pour l'accès à une unite de stockage dans le PersistentVolume depuis les Pods





- **StorageClass (SC):** template pour l'approvisionnement dynamique de stockage PV



Pods: approche imperative (commandes)

- Lister les ressources:

```
$ kubectl get all
```

- Lister les pods

```
$ kubectl get pods
```

- Demarrer manuellement un pod

```
$ kubectl run [nom] –image=nginx:alpine
```

- Ouvrir un pod pour l’accès externe (8080:port externe et 80:port interne au pod):

```
$ kubectl port-forward pod/[nom] 8080:80
```

- Suppression du pod

```
$ kubectl delete pod [nom]
```

Deployment: approche imperative (commandes)

- Lister les deployments:

```
$ kubectl get deployments
```

- Lister les deployments avec les labels

```
$ kubectl get deployments --show-labels
```

- Lister les deployments avec un label specific

```
$ kubectl get deployment –l app=nginx
```

- Ecoute sur le port 8080 :

```
$ kubectl port-forward deployment/[nom] 8080
```

- Augmenter les nombres de replica:

```
$ kubectl scale deployment [nom]
–replicas=5
```

- Suppression d’un deployment

```
$ kubectl delete deployment [nom]
```

Service: approche imperative (commandes)

- Lister les services:

```
$ kubectl get services
```

- Lister les services avec les labels

```
$ kubectl get services --show-labels
```

- Lister les services avec un label specific

```
$ kubectl get service –l app=nginx
```

- Ecoute sur le port 8080 :

```
$ kubectl port-forward service/[nom] 8080
```

- Executer des commandes dans un pod

```
$ kubectl exec [nom-pod] –it sh
> curl –s http://podId
```

- Suppression d’un service

```
$ kubectl delete service [nom]
```

- Suppression d’un service

```
$ kubectl delete service [nom]
```

Approche declarative (YAML)

Pod: Approche declarative (YAML)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
```

- Kubernetes API version
- Type of Kubernetes resource
- Metadata about the Pod
- The spec/blueprint for the Pod
- Information about the containers that will run in the Pod

Deployment: Approche declarative (YAML)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: my-nginx
    tier: frontend
spec:
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: my-nginx
        image: nginx:alpine
```

- Kubernetes API version and resource type (Deployment)
- Metadata about the Deployment
- The selector is used to "select" the template to use (based on labels)
- Template to use to create the Pod/Containers (note that the selector matches the label)

Service: Approche declarative (YAML)

```yaml
apiVersion: v1
kind: Service
metadata:
  ...
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

◀ Set Service *type* to NodePort

◀ Optionally set NodePort value (defaults between 30000-32767)

Approche declarative (YAML)

Créer une ou des ressources

```
$ kubectl create –f [nom-fichier-declarative].yml
```

Appliquer des changements

```
$ kubectl apply –f [nom-fichier-declarative].yml
```

Supprimer une ou des ressources

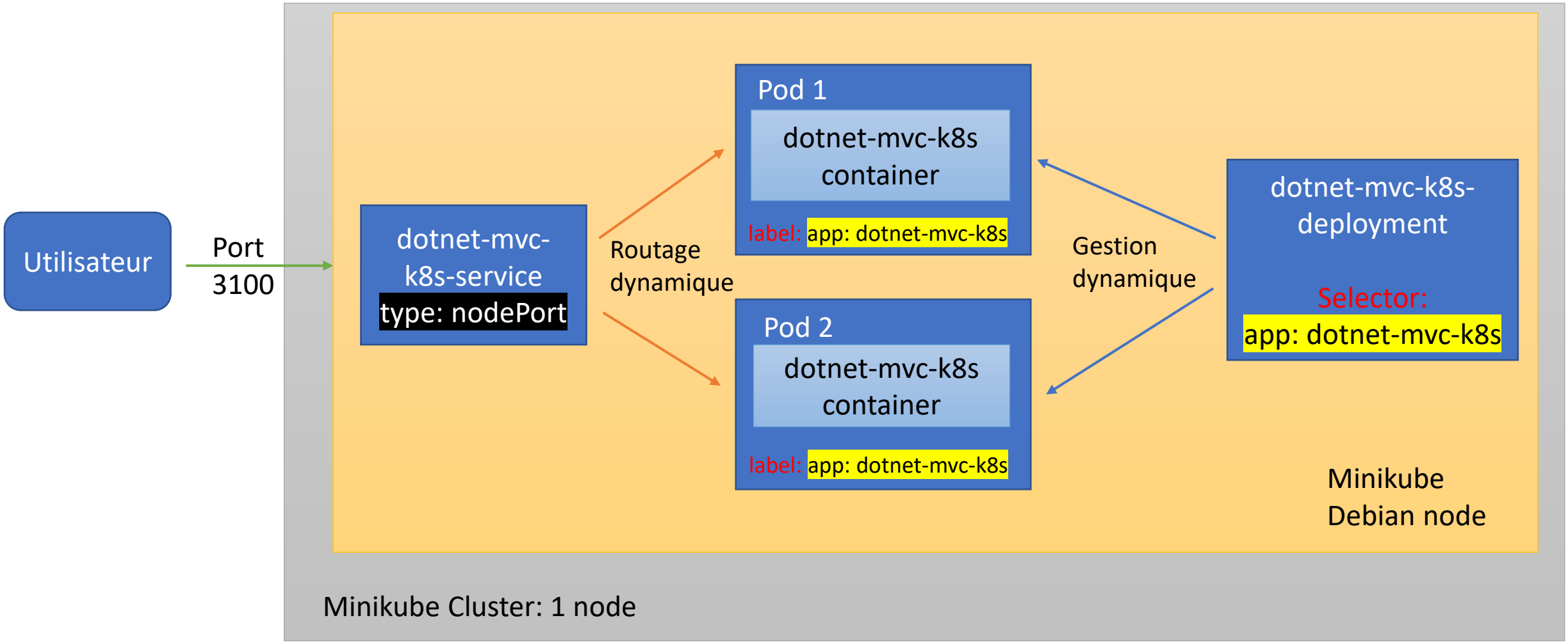
```
$ kubectl delete –f [nom-fichier-declarative].yml
```

Cas d’usage: Outils et environnements

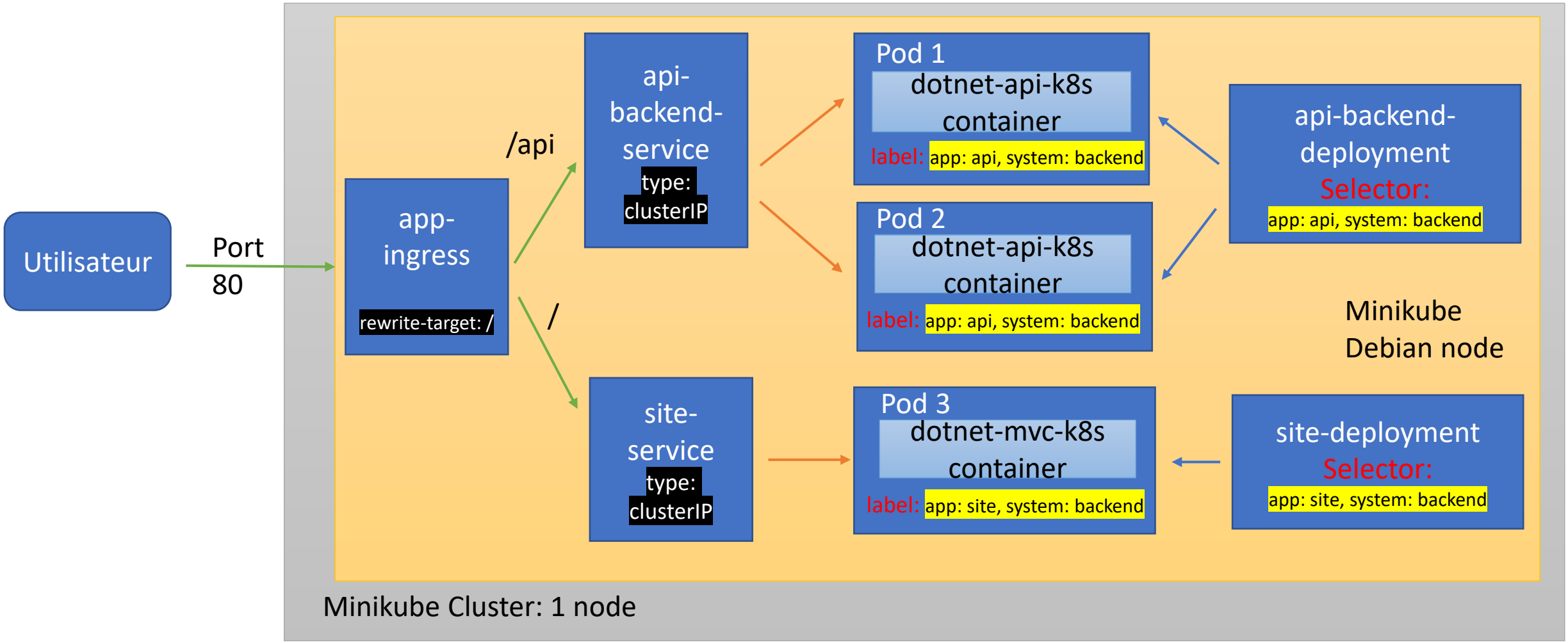
Cluster composé d’un seul node jouant à la fois le role de master et de worker node (Installation en local)

- *Kubernetes version:* minikube
- *Node:* Debian 10 buster (installé en VM dans hyper-V sous win10)
- *Container:* docker (sous dockerhub)
- *Application stack:* ASP.NET Core 3.1

1er Cas d'usage: Application simple avec service type nodePort (deploiement avec kubectl)



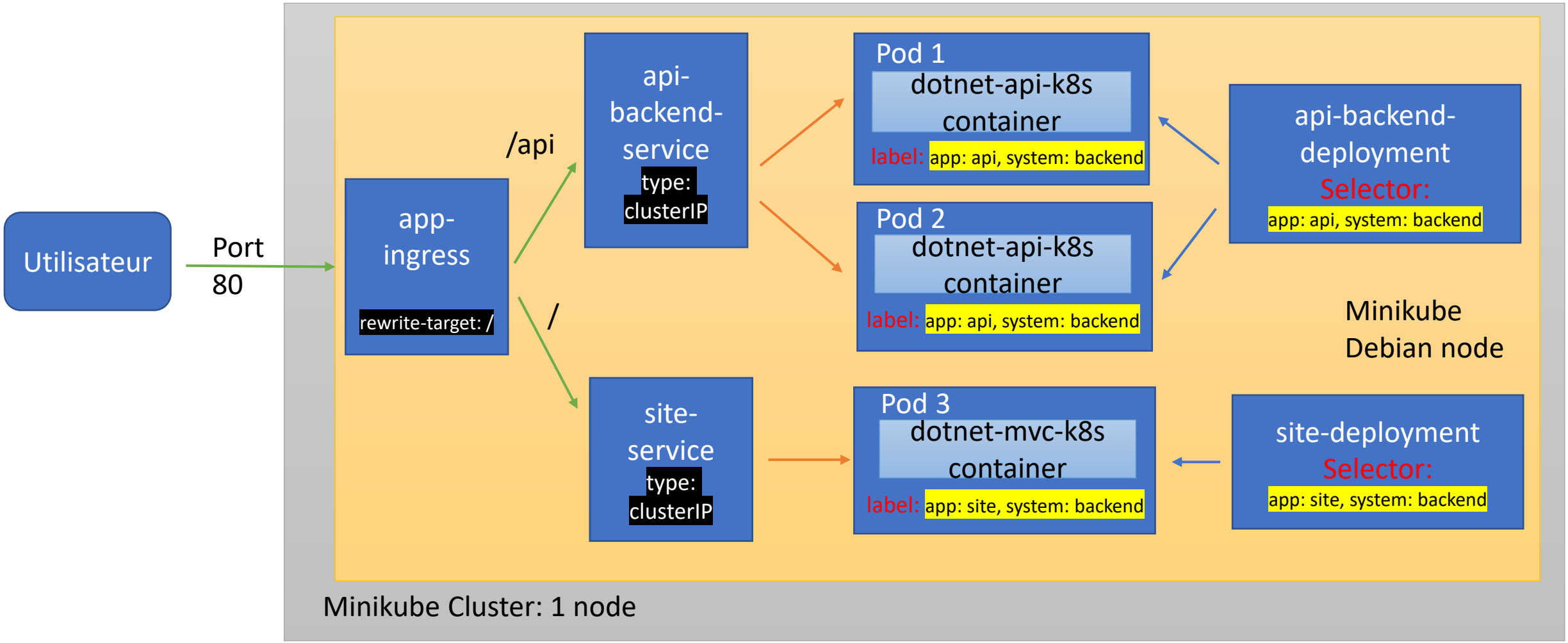
2eme Cas d'usage: Application à 2 microservices avec service type clusterIP et ingress (deploiement avec kubectl)



HELM

- Outil de gestion des “Kubernetes charts”
- Charts: Paquets de ressources Kubernetes pré-configures
- Deploiement d’application sur Kubernetes => bcp de ressources à deployer (deployment, service, ingress, etc.)
- Assure que tous les ressources requises sont installés (en les regroupant en paquets: les charts)
- Permet de parametrer les manifests YAML avec les Helm Templates
- Versionner les releases kubernetes

3eme Cas d'usage: Application à 2 microservices avec service type clusterIP et ingress (deploiement avec helm)



HELM: Utilisation

- Lister les releases:

```
$ helm list
```

- Lancer un deployment:

```
$ helm upgrade --install [releaseName] . \
                        --set api.image.tag="nightly" \
                       --set site.image.tag="firsttry" \
                       --debug \
```

- Annuler une release:

```
$ helm rollback [releaseName]
```

- Supprimer une release:

```
$ helm uninstall [releaseName]
```

Troubleshooting Kubernetes

Visionner les logs de Pod

```
$ kubectl logs [nomPod]
```

Logs de container specific dans un Pod

```
$ kubectl logs [nomPod] –c [nomContainer]
```

Decrire un Pod

```
$ kubectl describe pod [nomPod]
```

Utiliser le format YAML

```
$ kubectl get pod [nomPod] –o yaml
```

Executer une commande shell dans le container d’un Pod

```
$ kubectl exec [nomPod] –it sh
```

Troubleshooting Kubernetes

Un mode graphique disponible avec le dashboard

```
$ minikube dashboard
```

Aller plus loin dans Kubernetes

- ConfigMap et Secrets
- Volume / PersistentVolume / PersistentVolumeClaim / StorageClass
- Horizontal Pod Autoscaler, Loadbalancer service
- Stateful Sets, Jobs, Daemon Sets
- RBAC (Service Accounts)
- Etc.