Cloud Computing Applications and Services

(Aplicações e Serviços de Computação em Nuvem)

Guide 3: Kubernetes

University of Minho

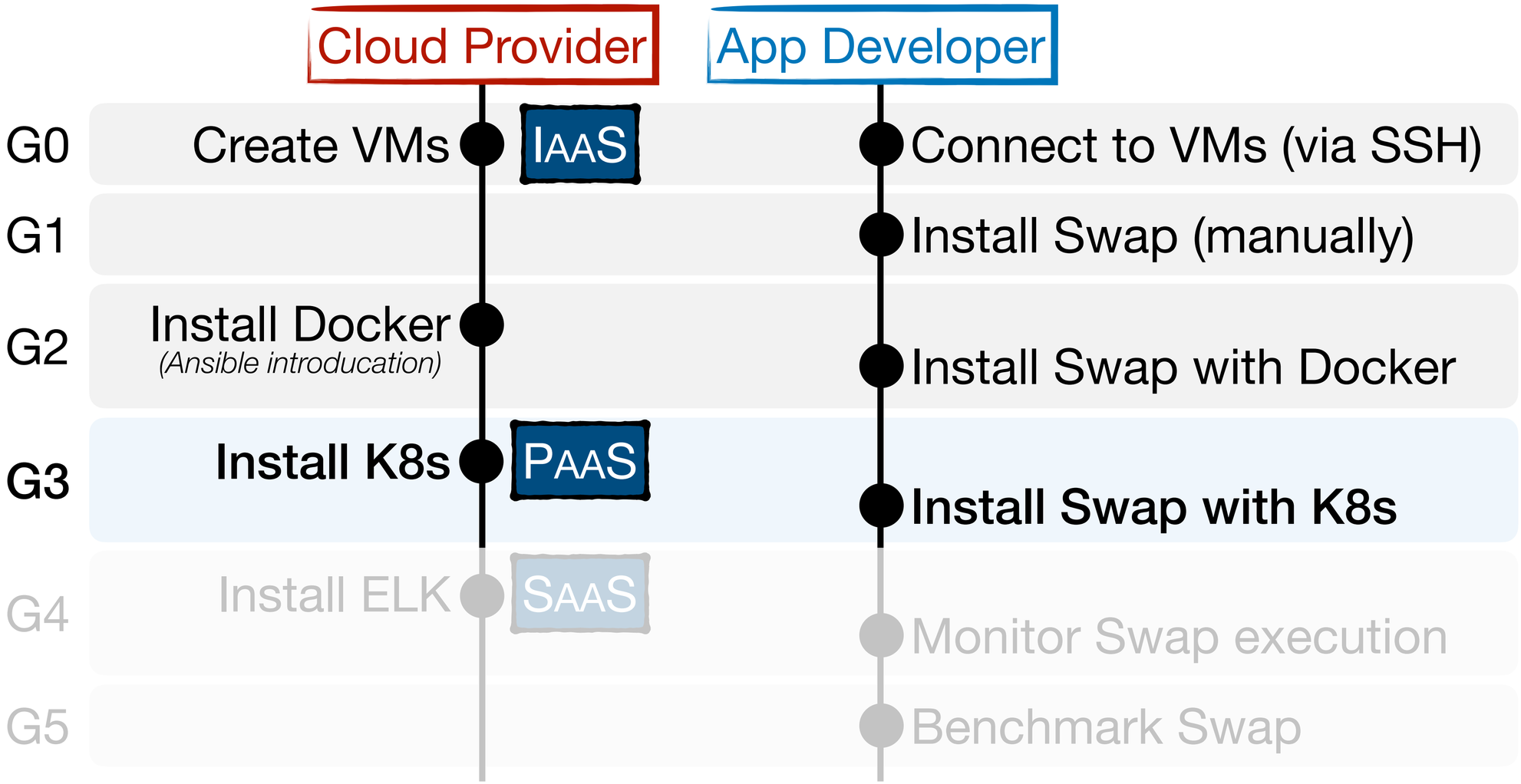
2024-2025

Context

- In Guide 2, you used Docker and Ansible to automate the installation and configuration of Swap.
- What if now you want to increase or decrease the number of replicas of your application according to your demands (i.e., clients requests)?
- To address these challenges in an automated fashion you need to resort to **container orchestration and scheduling platforms**.

Road Map

Where are you on the roadmap?



Goal

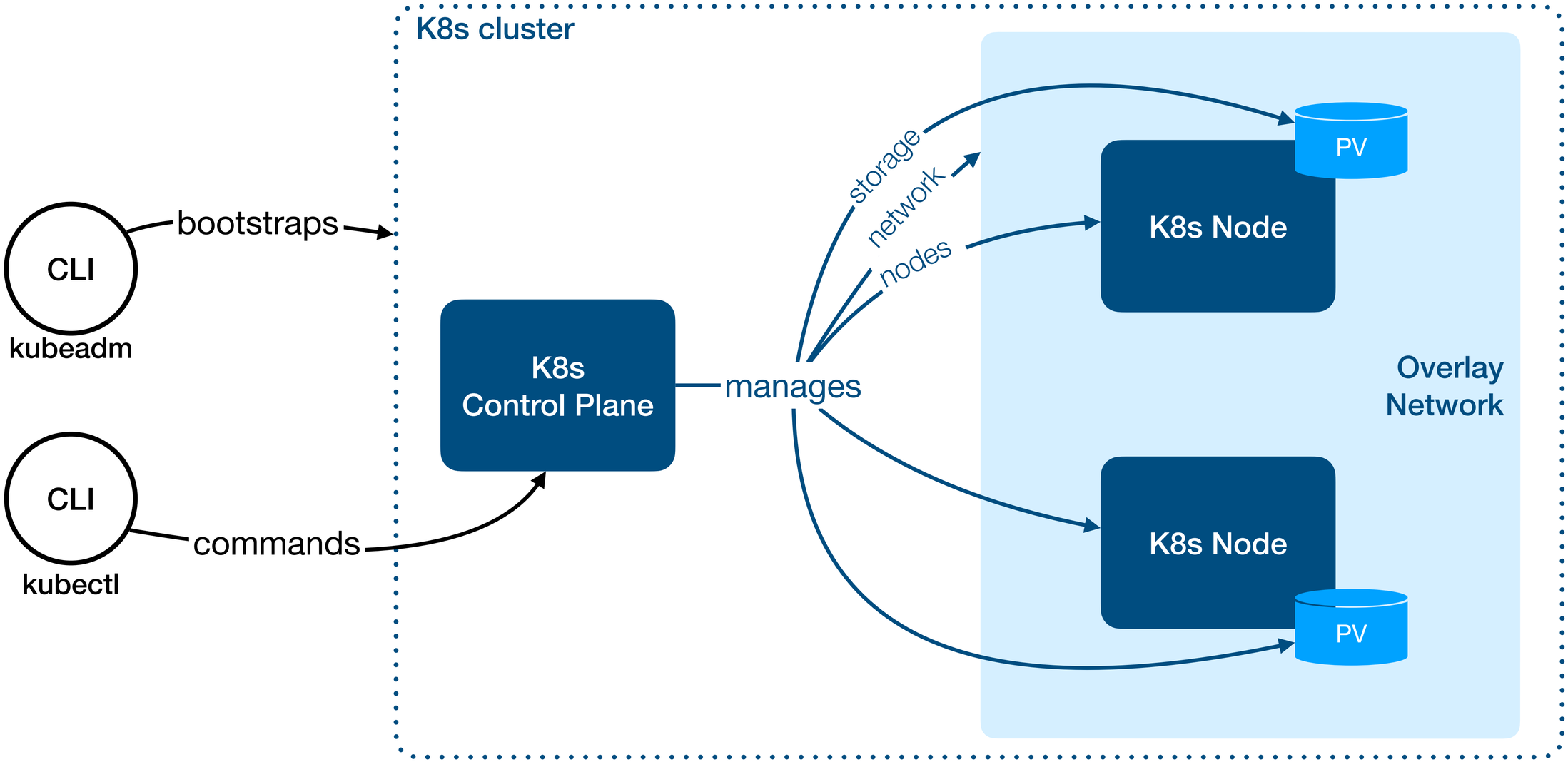
- In this Guide you will use **Kubernetes (K8s)** for **automate the deployment, scaling and management** of Swap.
- The Guide is divided in two parts:
  - **PART I:**
    - Set up a K8s cluster (using Ansible)
    - Use K8s to automate Swap's configuration and deployment.
  - **PART II:**
    - Use Ansible to automate the configuration and deployment of Swap with K8s.

Part I

kubernetes

Kubernetes (K8s) is an open-source container orchestration system for automating software deployment, scaling, and management.

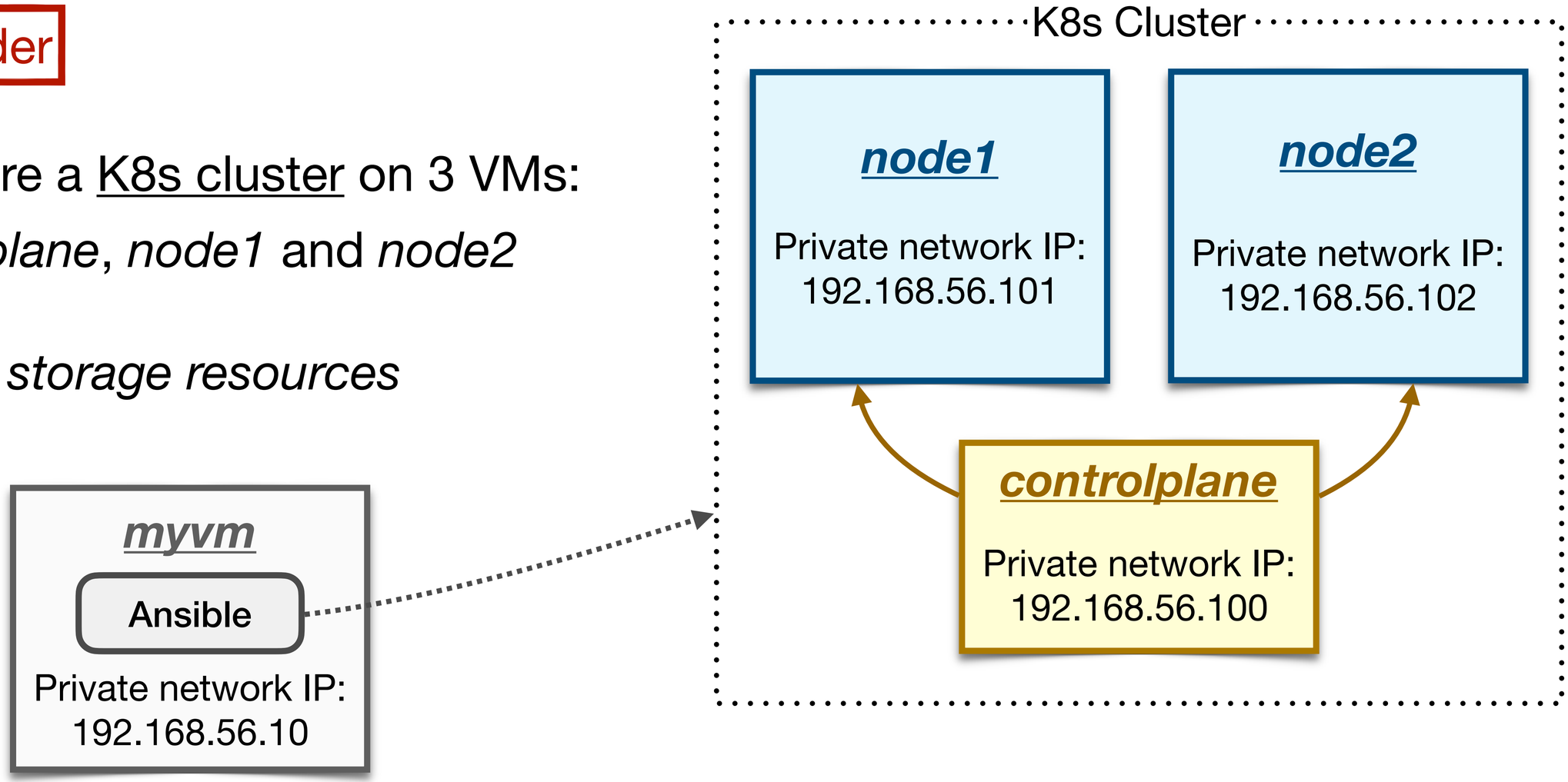
Kubernetes Cluster



Part I - Goal

Cloud Provider

- Configure a K8s cluster on 3 VMs: *controlplane*, *node1* and *node2*
- *Provide storage resources*



CloudProvider Project

(Updates)

Variables for the k8s group

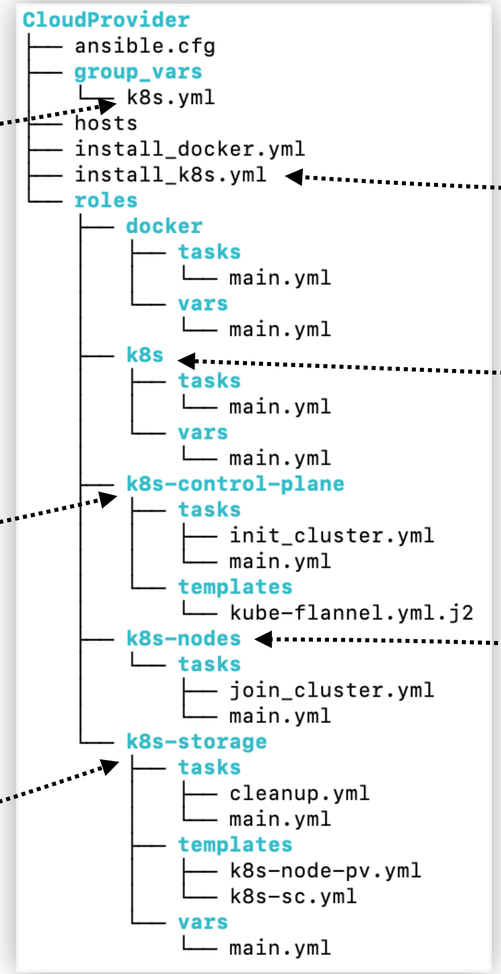
Playbook for installing K8s

Role for configuring the K8s Control Plane

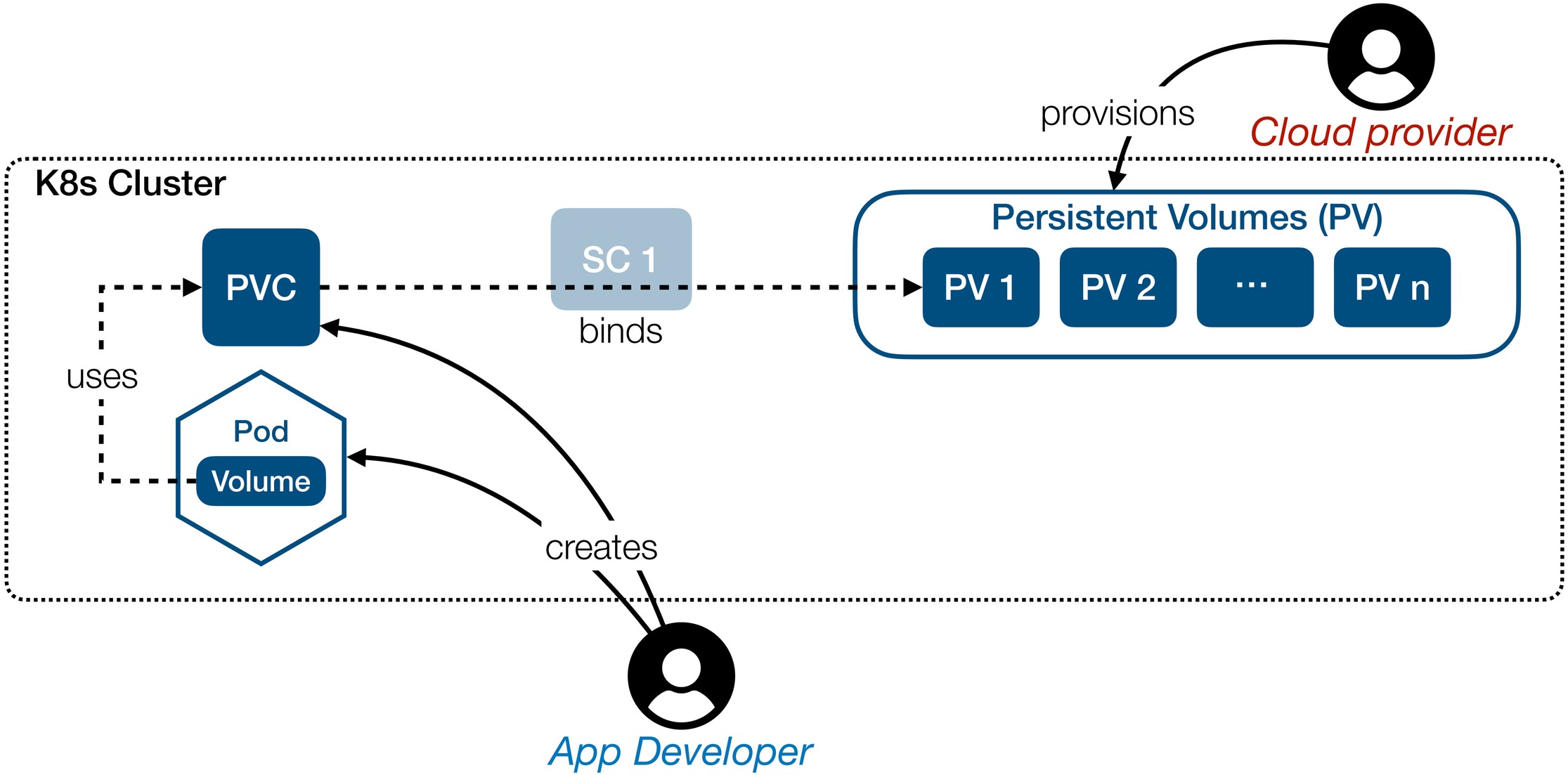
Role for installing K8s packages

Role for configuring storage resources

Role for configuring the K8s Nodes



Storage Resources



Storage Resources

- We (as a cloud provider) need to provide storage resources (through PVs) in the cluster so clients can request them (through PVCs) to ensure persistent storage for their Pods.
- For that, we will configure **Local Persistent Volumes**, which are local storage devices mounted on nodes. The K8s scheduler ensures a pod using this volume type is always scheduled to the same node.

Note: Depending on your setup/goal, there are other types of persistent volumes that may be more adequate: *nfs* (for network FS storage), *csi* (for cloud platforms such as GCP), etc.

Find more about Persistent Volumes Types at:
https://kubernetes.io/docs/concepts/storage/persistent-volumes/#types-of-persistent-volumes

PersitentVolume (PV)

A piece of storage

The example shows the specification of a **PV** named pv-1.

The **reclaim policy** is set to Retain, which means that the volume will not be automatically deleted.

The local field indicates this is a Local PV, which ties your application to a specific node on a specific path.

The nodeAffinity field is how the K8s scheduler understands that this PV is tied to a specific node (node1).

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-1
spec:
  capacity:
    storage: 50Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /mnt/data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node1
```

The storage **capacity** is set to 50 GiB.

The volume **access mode** is set to ReadWriteOnce, which allows being mounted by a single node in read-write mode.

The storageClassName field specifies the **StorageClass** (local-storage) that will dynamically bind the PV to a PVC.

StorageClass (SC)

Classes of storage offered

The example shows the specification of a **Storage Class** named local-storage.

The volumeBindingMode field controls when volume binding and dynamic provisioning should occur.

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

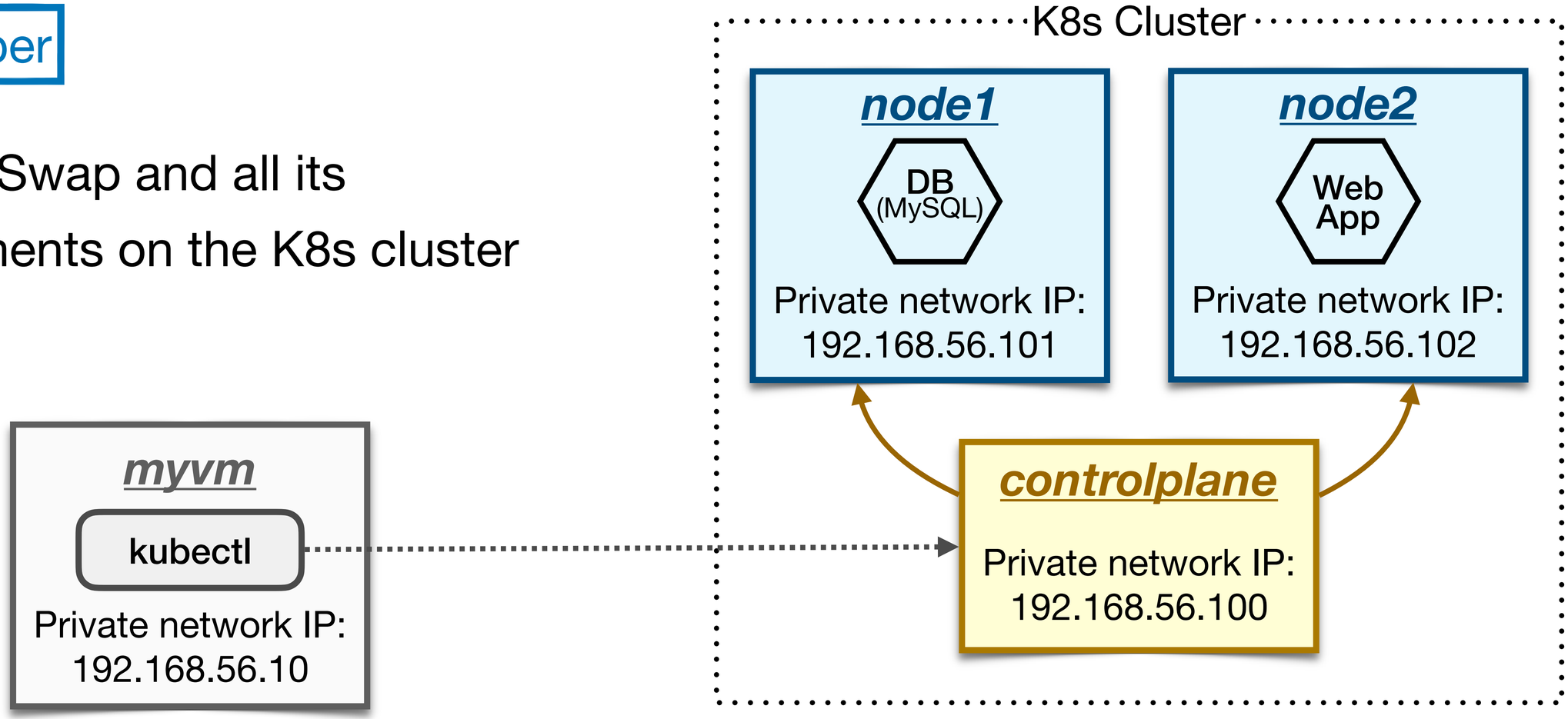
The provisioner field identifies the volume plugin (e.g., kubernetes.io/azure-file, kubernetes.io/gce-pd) used for provisioning PVs.

Local volumes do not support dynamic provisioning. However, a StorageClass should still be created to delay volume binding until the Pod is scheduled. This is done by specifying the WaitForFirstConsume volume binding mode. In this case, the provisioner is set to no-provisioner.

Part I - Goal

App Developer

- Deploy Swap and all its components on the K8s cluster



Pod

Computing unit composed by one or more containers

The example shows the specification of a **Pod** (kind field), which is named nginx (metadata.name field).

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
      - containerPort: 80
```

The Pod is composed of one container (named nginx) and running the image nginx:1.14.2.

Usually, you don't need to create Pods directly.

Instead, create them using workload resources such as **Deployment**.

Deployment

Manages the desired state and lifecycle of Pods

The example shows the specification of a **Deployment** named nginx-deployment.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

The .spec.replicas field indicates to create a **ReplicaSet** to bring up three Pods.

The .spec.selector field defines how the created ReplicaSet finds which Pods to manage (i.e., through the label defined in the Pod template app: nginx).

The **Pod template**'s specification (.template.spec field) indicates that the Pods run one container, named nginx, which runs the image nginx:1.14.2.

PersistentVolumeClaim (PVC)

A request for storage resources

The example shows the specification of a **PVC** named mysql-pvc.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc
  namespace: default
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 20Gi
```

This claim is requesting a PV with the access mode set to ReadWriteOnce and a storage capacity of at least 20 GiB.

The claim can request a particular class by specifying the name of a StorageClass. Only PVs of the local-storage class can be bound to this PVC.

Service

- Once we (as an application developer) deploy Swap's components on different Pods, we must create Services so they can communicate:
  - Web App server needs to access the Database **(internal communication)**
  - Clients need to access the Web App server (**external communication)**
- Different types of Services:
  - ClusterIP (internal communication)
  - NodePort (external communication)
  - LoadBalancer (external communication)
    - Uses an external load balancer
    - Useful in Cloud Platforms

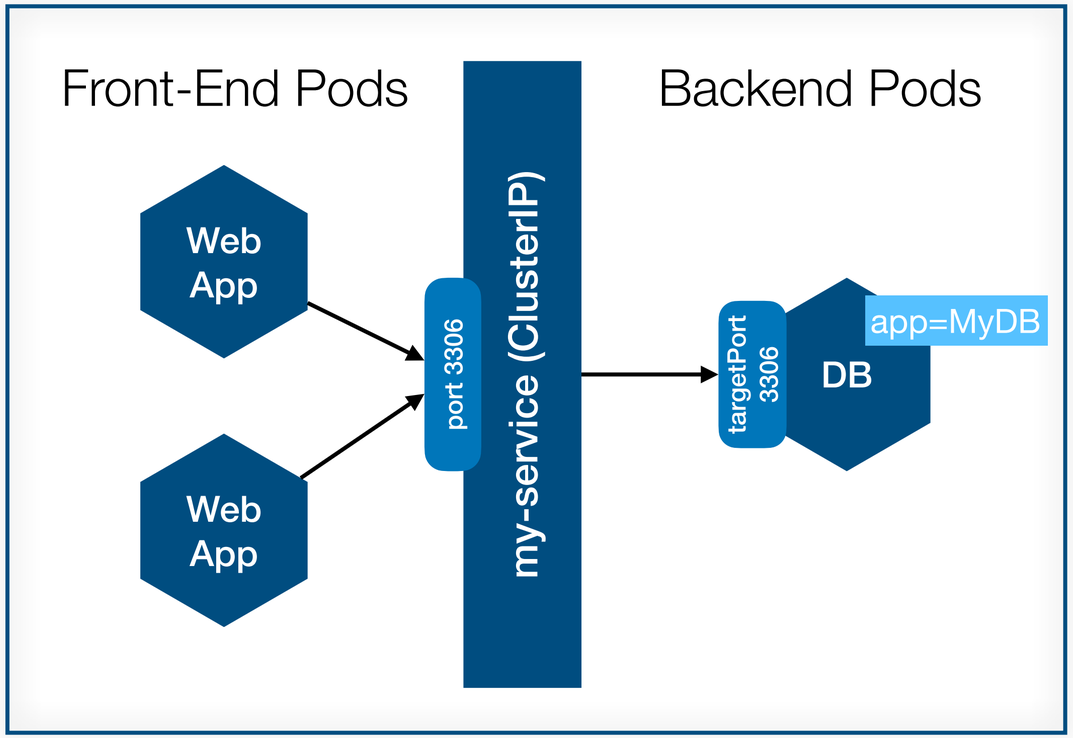
Find more about Services at: https://kubernetes.io/docs/concepts/services-networking/service/

ClusterIP Service

Exposes the Service on a cluster-internal IP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: db-service
spec:
  type: ClusterIP
  selector:
    app: MyDB
  ports:
    - targetPort: 3306
      port: 3306
```

The example shows the specification of a **ClusterIP Service** named db-service that targets pods labeled as app: MyDB.



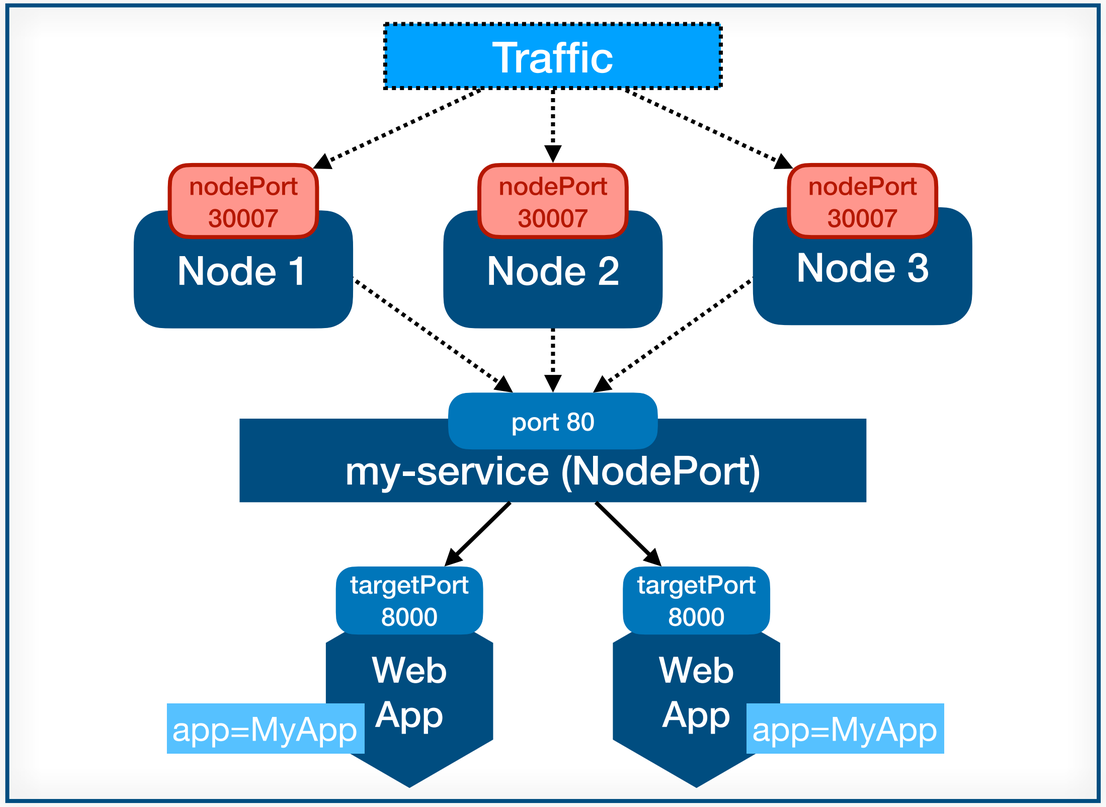
The port field corresponds to the port on which the service is exposed (3306).
The targetPort field indicates the port on which your application is running inside the container (3306).

NodePort

Exposes the Service on each Node's IP at a static port

```yaml
apiVersion: v1
kind: Service
metadata:
  name: app-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
    - targetPort: 8000
      port: 80
      nodePort: 30007
```

The example shows the specification of a **NodePort Service** named app-service that targets pods labeled as app: MyApp.



The nodePort field corresponds to the port on each node where external traffic will come in (30007).

Kubernetes
Useful Commands

- Execute commands at pods:
  - kubectl exec –it <pod_name> -- <command>
- Check Pod logs:
  - kubectl logs <pod_name>
- Deploy and delete objects:
  - kubectl apply –f <file.yml>
  - kubectl delete –f <file.yml>
  - kubectl delete <type> [name]
- Check resources/objects:
  - kubectl get all
  - kubectl get <object_type> [name]
  - Kubectl describe <object_type> [name]
- Filter resources by label:
  - kubectl get all --selector=app=MyApp

Part II

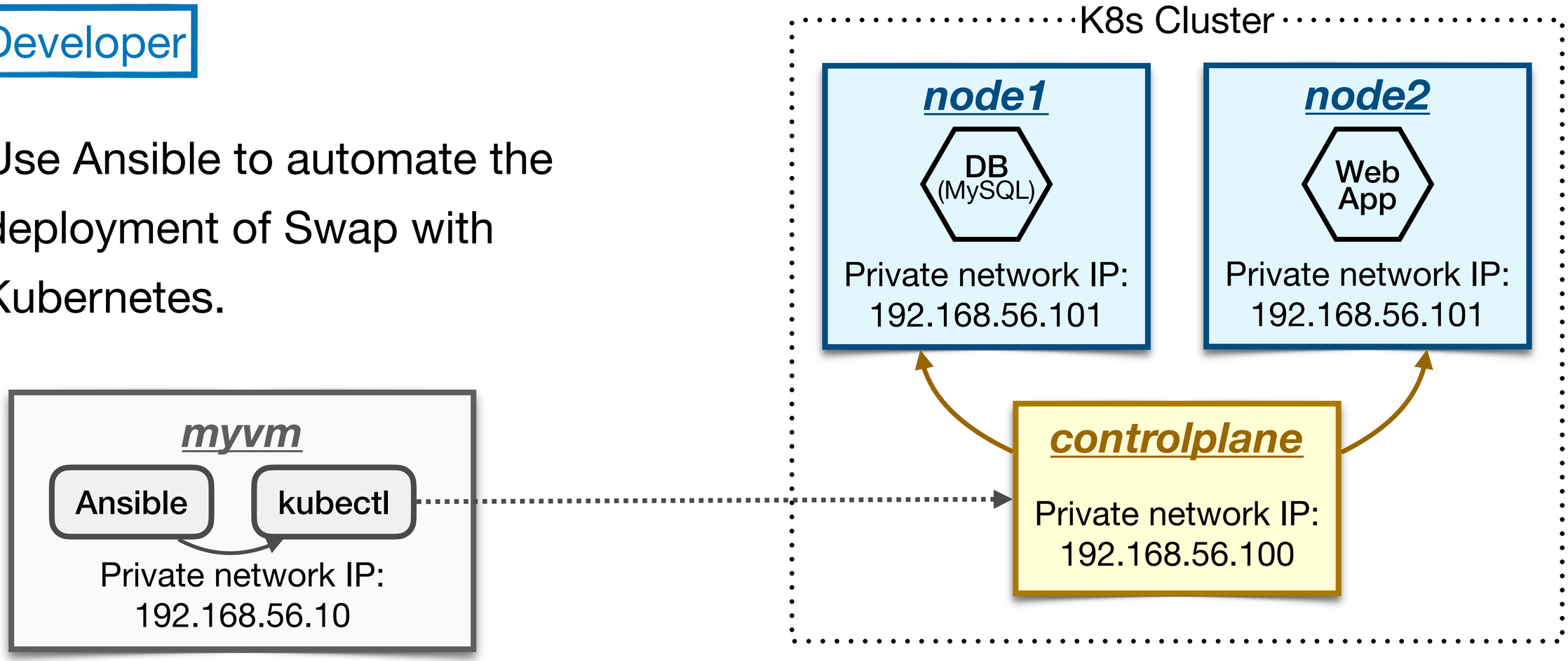
Part II - Context

- Now, the installation and scaling of Swap is automated with Kubernetes.
- However, what if we wanted to deploy Swap on **different K8s clusters**? (Remember that you need to do this for all Portuguese Universities…)
  - You would have to:
    - manually configure and apply all Kubernetes objects (e.g., deployments, services, etc) in all the clusters.
- Once again, **Ansible** can further automate these manual steps.

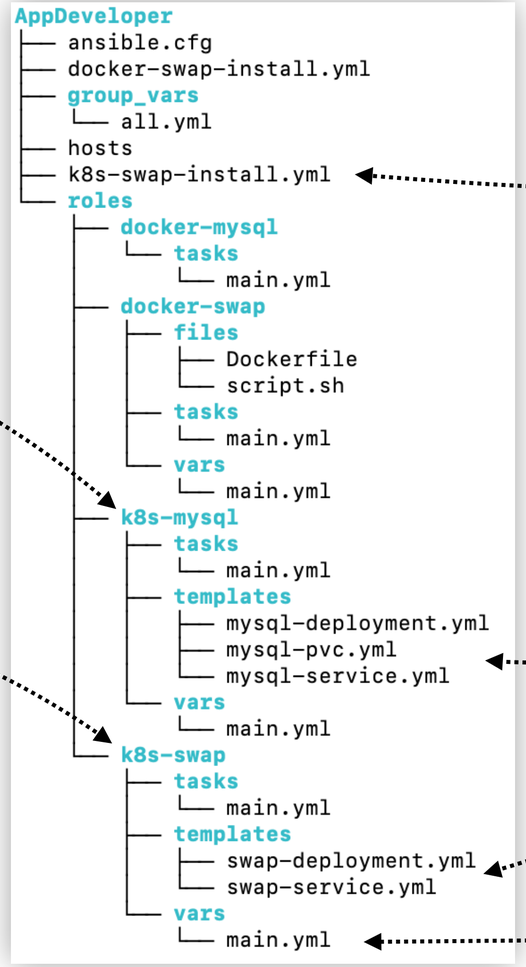
Part II - Goal

App Developer

- Use Ansible to automate the deployment of Swap with Kubernetes.



AppDeveloper Project
(Updates)



Playbook for installing Swap with K8s

Role for installing Swap's Database with K8s

Role for installing Swap's Web App server with K8s

Templates with the description of K8s objects

Role's variables