Namespace

→ Organise resources in namespaces
→ Virtual cluster inside cluster

Kubernetes provides 4 default namespace

*Kuberenetes-dashboard → only with minikube

Kube-system
→ do not create or modify in Kube-system
→ system processes are present
→ Master & Kubectl processes are present

Kube-public
→ publicely accessible data
→ configMap data, which contains cluster information

Kube-node-lease
→ heartbeats of nodes
→ each node has associated lease object in namespace
→ determines the availability of node

default → resources you create are located here

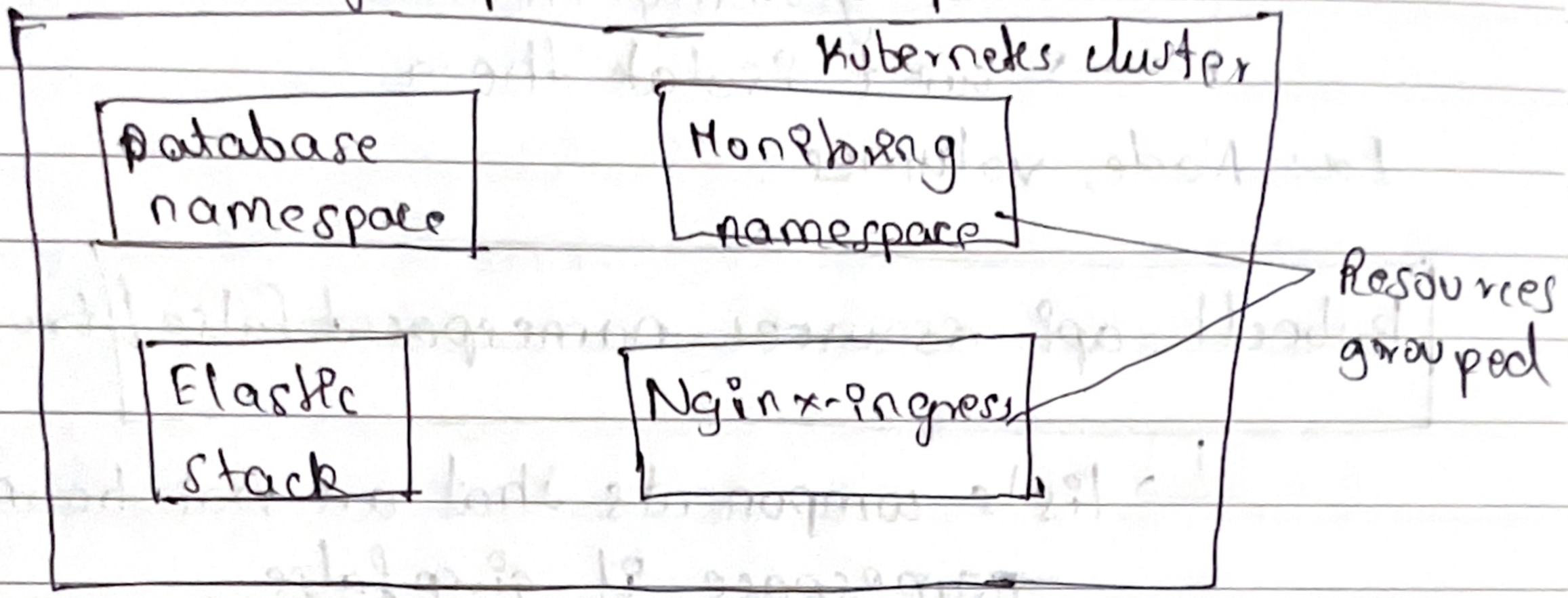
Note:

Create namespace with configuration file

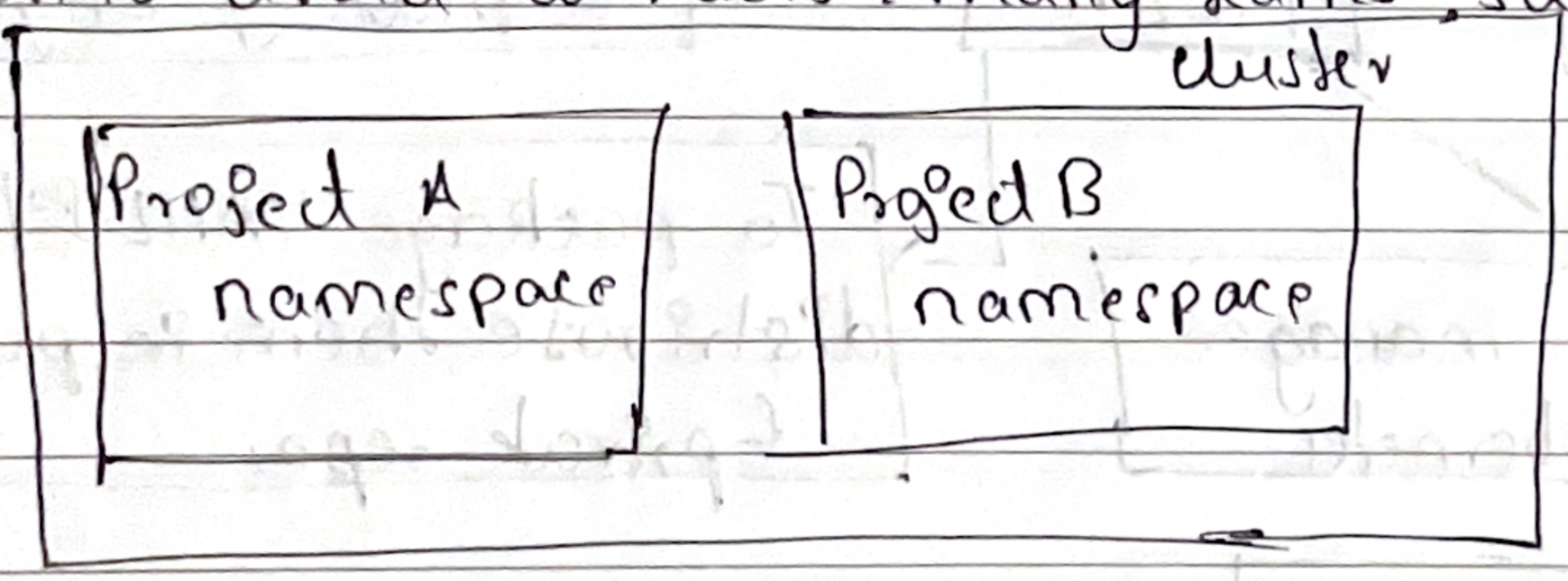
Officially: should not use for smaller projects

Why need for namespace?

i) Resources grouped in namespaces



ii) Can To avoid conflicts: many teams, same application



iii) Resource sharing: staging & development

iv) Blue Green development like different versions of application

v) Access & resource limits on namespace between teams

Note:

i) You can't access most resources from another namespace like you ~~should~~ ~~could~~ can't use or access configMap or secret of other namespace.

Note:

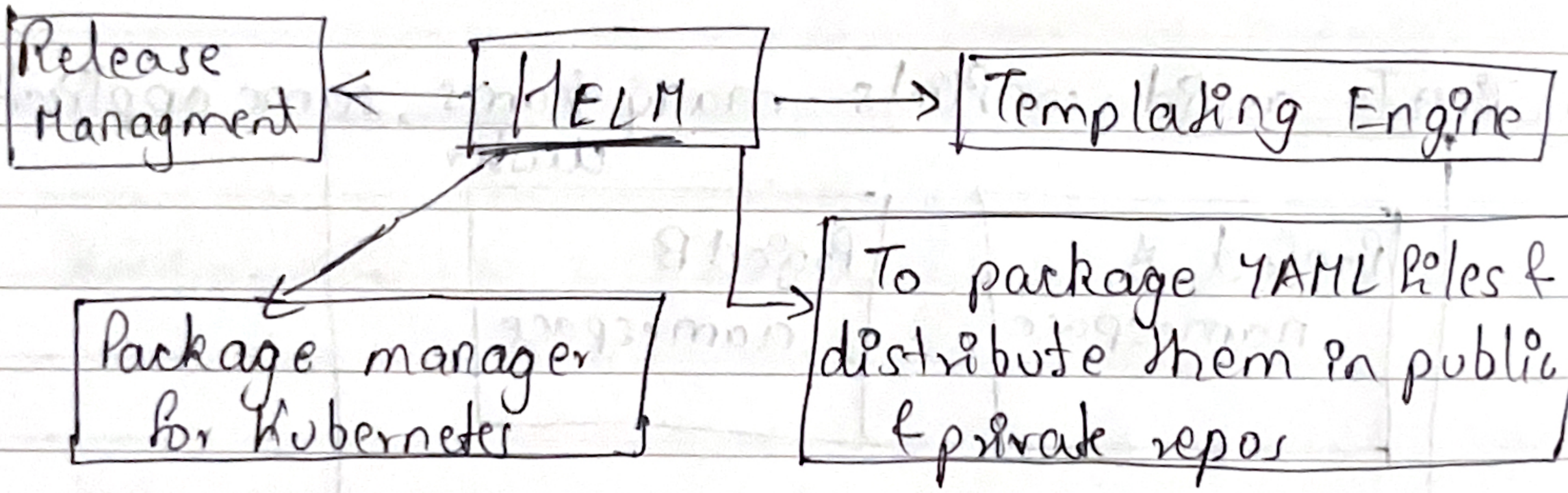
Some components cannot be created within namespace because

- live globally in cluster
- can't isolate them

Ex: Node, volumes

```
kubectl api-resources --namespaced=false / true
```

↳ lists components that are not bound to namespace if given false



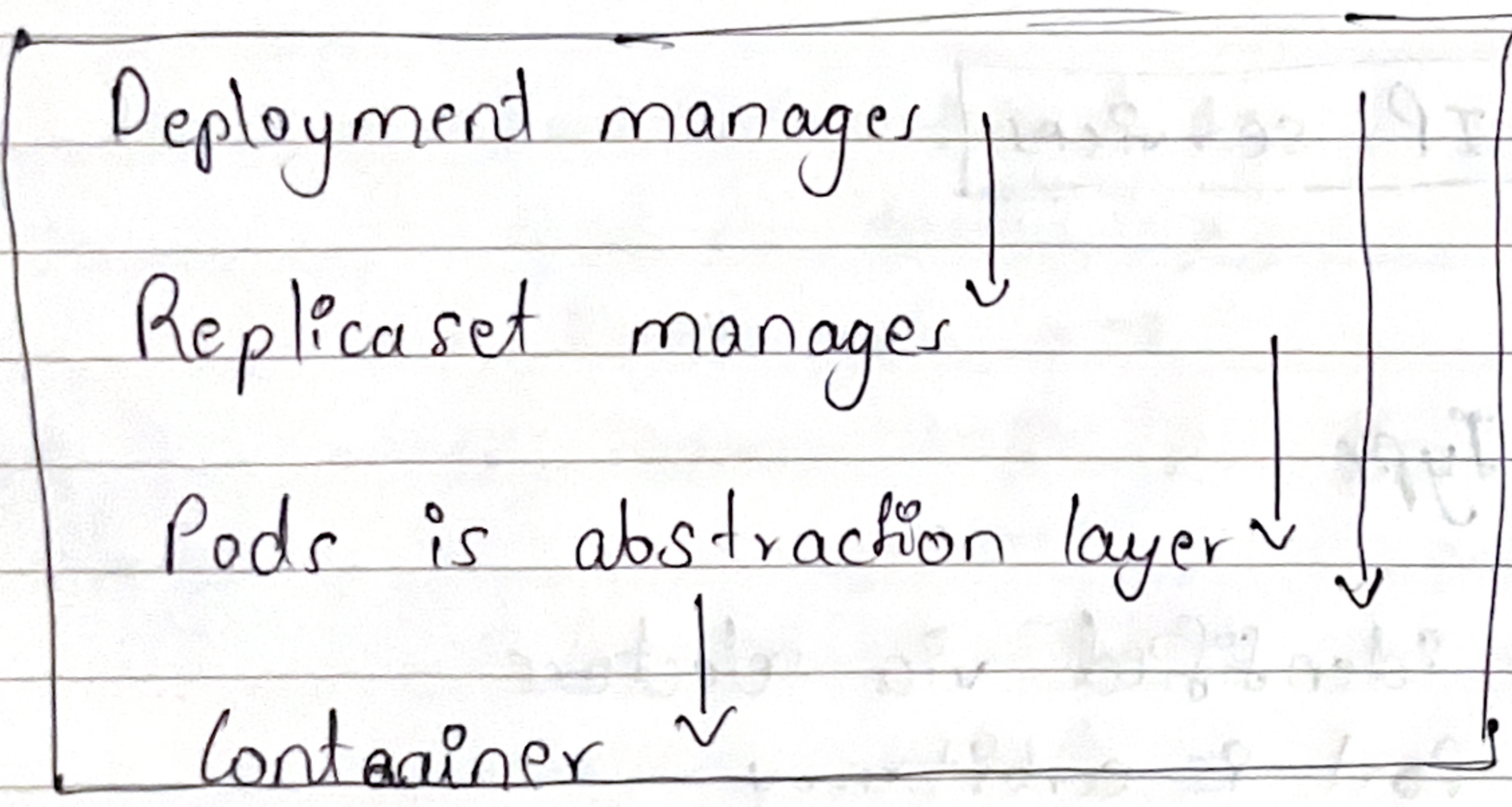
Helm Charts

- Bundle of yaml files
- create your own Helm charts with helm
- Push them to helm repo
- download & use existing ones

YAML file ——> strict indentation

Each configuration file has 3 parts:

- i) Meta data
- Specification
- Status (automatically generated & added by Kubernetes)



Labels & selectors
↳ for connection establisment

Array / Lists

Fruits:
- Orange
- Apple

Vegetables:
- Carrot
- Tomato

Dictionary / Map

Banana:
Calories: 105
Fat: 0.4g

Grapes:
Calories: 62
Fat: 0.2g

Dictionary
↳ Unordered

List
↳ Ordered

Kubernetes services

why

* Stable IP address
* Loadbalancing
* Loose coupling within & outside cluster

Types

i) ClusterIP
ii) Headless
iii) NodePort
iv) LoadBalancer

ClusterIP services

Default Type

* Pods are identified via selectors
* Service Port is arbitrary
* Target port must match port, the container is listening at

Headless services

Client wants to communicate with a specified pod directly

Pods want to talk directly with a specified pod

Use case:
Stateful applications like databases

We can set ClusterIP to none to use headless service

3 Service type attributes

ClusterIP

NodePort

External traffic has access to fixed port on each worker node

nodePort attribute is used for declaring fixed port range: 30000 - 32767

Accessible via browser request

* cluster-ip is automatically created
* NodePort services not secure

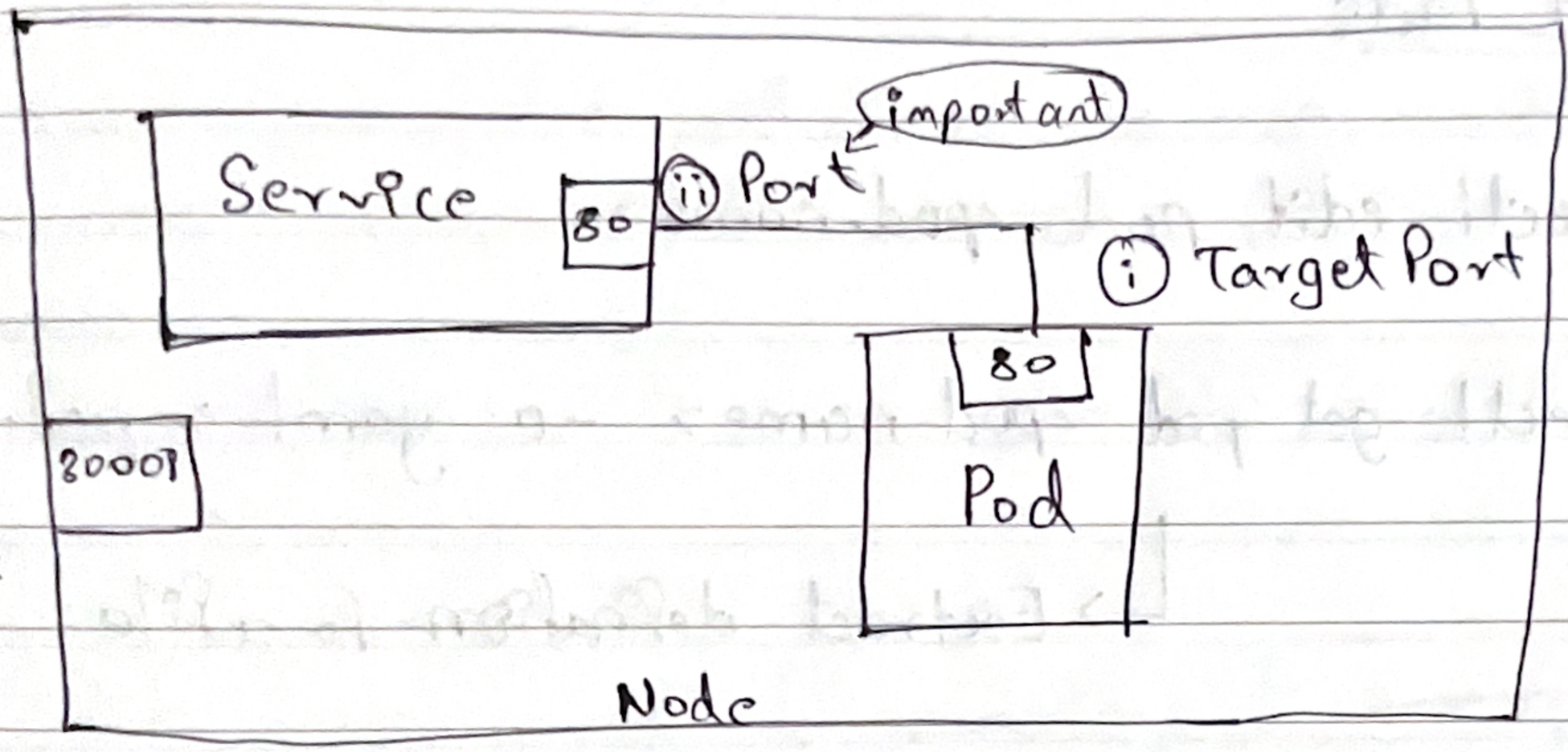
LoadBalancer

Becomes accessible externally through cloud providers loadbalancer

NodePort & clusterIP are automatically created

* LoadBalancer service is extension of NodePort
* NodePort is extension of ClusterIP
* Configure Ingress or LoadBalancer for production environments

Service - NodePort



svc-def.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-svc
spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80*
      nodePort: 30008
  selector:
    app: myapp
    type: Frontend
```

pod-def.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: Frontend
spec:
  containers:
    -name:
      image:
```

* Required

Update & Rollback deployment commands

```
k create -f <FileName>
k get deployments
```
— { Create, Get }

```
k apply -f <FileName>
k set image deployment/app-deployment
    nginx = nginx:1.9.1
```
— { Update }

```
k rollout status deployment/<name>
k rollout history  "
```
— { Status }

```
k rollout undo deployment/<name>
```
— Rollback

Networking criteria in Cluster ⟶ Should be set by us

* All containers/PODs can communicate to one another without NAT
* All nodes can communicate with all containers & vice-versa without NAT

* We can check status of each revision individually by using `--revision=<>` flag
* We can use --record flag to save command against the revision number
* To rollback to specific version we will use --to-revision = <revision number>