

# Kubernetes

# Proč Kubernetes?

- Proč Kubernetes?
  - Abstrakce nad hardware
    - Nestaráme se o to, že nějaká aplikace má běžet na dvou konkrétních serverech v X instancích, ale máme cluster vytvořený ze stávajících uzlů na kterých běží Kubernetes a ten řídí nasazování aplikací na jednotlivé uzly
  - Treat your applications like cattle, not pets
    - <https://thenewstack.io/how-to-treat-your-kubernetes-clusters-like-cattle-not-pets/>
  - S Kubernetes říkáme, jaký požadovaný stav má mít celý systém a je na Kubernetes, aby toho stavu docílil.
  - Když je zapotřebí docílit HA (High Availability)

# Kdy použít Kubernetes?

- Kdy použít Kubernetes?
  - Stateless aplikace
  - Batch processing
  - Webové servery
  - Mobilní backend
- Na co se Kubernetes nehodí?
  - Persistent data storage (typicky databáze)

# Instalace Kubernetes (kubeadm)

- 1. nainstalovat Docker:
  - <https://docs.docker.com/engine/install/ubuntu/>
- 2. nainstalovat Kubernetes (pro produkci):
  - <https://phoenixnap.com/kb/install-kubernetes-on-ubuntu>

# Instalace Minikube (Ubuntu)

- 1. nainstalovat Docker:
  - <https://docs.docker.com/engine/install/ubuntu/>
- 2. nainstalovat Minikube (pro development):
  - <https://phoenixnap.com/kb/install-minikube-on-ubuntu>
- Minikube hello world:
  - <https://kubernetes.io/docs/tutorials/hello-minikube/>

# Instalace Minikube (Windows)

- 1. nainstalovat Docker:
  - <https://docs.docker.com/engine/install/ubuntu/>
- 2. nainstalovat Chocolatey:
  - <https://chocolatey.org/install>
- 3. nainstalovat Minikube (pro development):
  - <https://minikube.sigs.k8s.io/docs/start/>
- Minikube hello world:
  - <https://kubernetes.io/docs/tutorials/hello-minikube/>

**Nebo Docker Desktop se zapnutou podporou pro Kubernetes!**

# Minikube & zrychlení vývoje aplikací I.

- Normálně aby bylo možné používat nějakou Docker image v kubernetes, tak se musí provést následující posloupnost operací:
  1. vytvořit image
  2. provést push do registry
  3. `kubectl apply -f ...`
  4. kubernetes provede pull z registry
  5. kubernetes vytvoří kontejner

# Minikube & zrychlení vývoje aplikací II.

- Co kdyby to šlo pro lokální vývoj zjednodušit a kubernetes by si tu image bral z lokálního Docker registry?
  - <https://stackoverflow.com/questions/42564058/how-to-use-local-docker-images-with-minikube>
  - <https://stackoverflow.com/questions/52310599/what-does-minikube-docker-env-mean>
- Posloupnost operací:
  - minikube start
  - eval \$(minikube docker-env)
  - mvn spring-boot:build-image / docker build ...
  - Musí být nastaveno spec.template.spec.containers.imagePullRegistry: Never
  - kubectl apply -f ...



# Harbor

- Instalace:
  1. Nainstalovat Helm: <https://helm.sh/docs/intro/install/>
  2. Nainstalovat Harbor:

```
helm repo add harbor https://helm.goharbor.io
```

```
helm install my-release harbor/harbor
```

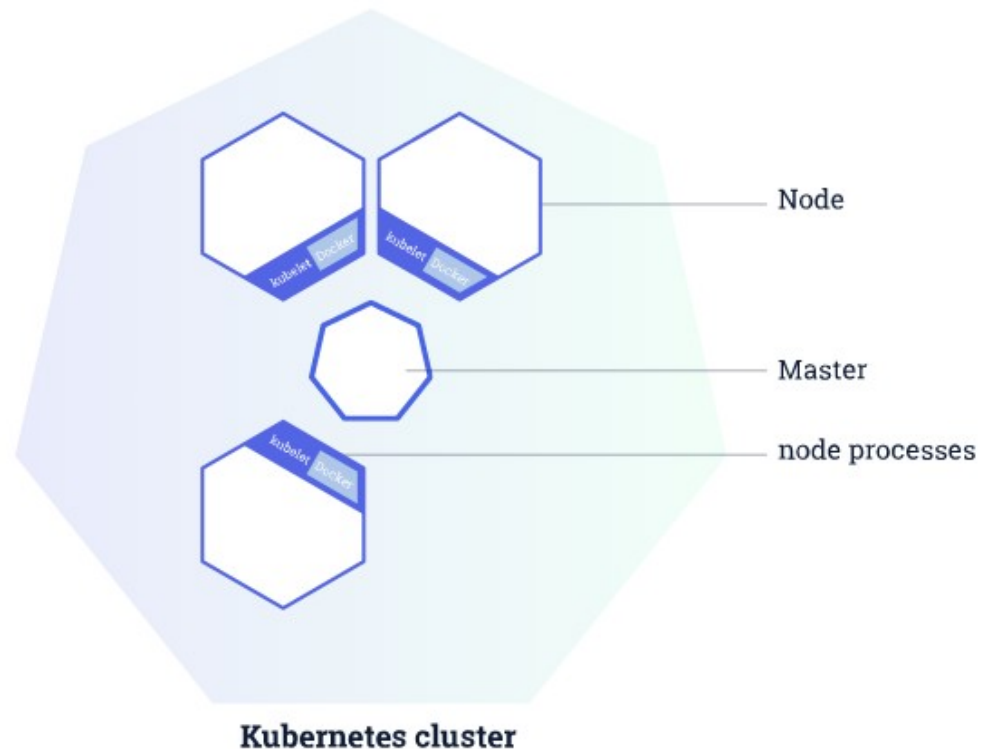
    - <https://github.com/goharbor/harbor-helm>
- Odinstalování:
  - `helm delete my-release`

# Minikube vs. kind vs. K3S

- Pro vývoj se v dnešní době nemusí používat jenom Minikube:
    - <https://brennerm.github.io/posts/minikube-vs-kind-vs-k3s.html>
  - Poznámka: K3S je “minifikovaný” kubernetes, který může běžet i na produkci!
    - Spuštění jedné instance:
      - `curl -sfL https://get.k3s.io | sh -`
      - `k3s kubectl get node`
    - Spuštění další instance:
      - `cat /var/lib/rancher/k3s/server/node-token`
      - `curl -sfL https://get.k3s.io | K3S_URL=https://serverip:6443 K3S_TOKEN=mytoken sh -`
- Tohle zavolat na první instanci, vrátí mytoken
- IP adresa první instance
- Tohle zavolat na druhé instanci

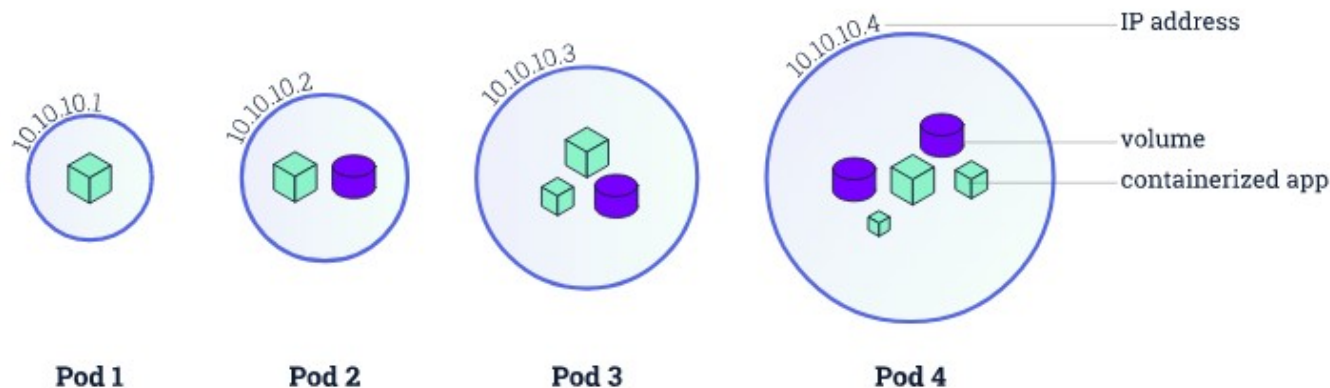
# Cluster

- Cluster se skládá minimálně z jednoho master serveru a volitelně z jednoho nebo víc uzlů (node)
- Poznámka: Minikube obsahuje jenom jeden uzel (mastera)
- Základní Kubernetes tutorial (nevyžaduje ani instalaci Minikube):
  - <https://kubernetes.io/docs/tutorials/kubernetes-basics/>



# Pods

- Uvnitř clusteru na node běží pody. Pod se skládá minimálně z jednoho kontejneru, ale může obsahovat i více kontejnerů a volumů:
  - <https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/>



**Poznámka:** Každý pod má svoji IP adresu!

# Pods

- Pokud jeden pod obsahuje více kontejnerů, pak je garantované, že všechny budou na jednom worker node.
- Když Kubernetes provádí škálování, pak škáluje celé pody, nikdy jednotlivé kontejnery.
- Většinou je nejlepší 1 pod = 1 container

# Deployment

- Pody se nenasazují na cluster “napřímo”, k tomu se používá deployment:
  - <https://stackoverflow.com/questions/41325087/what-is-the-difference-between-a-pod-and-a-deployment>
  - <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>
- Jak ovládat Kubernetes:
  - Pomocí ad-hoc příkazu (kubectl nebo dashboard)
    - Příklad: `kubectl create deployment nginx --image nginx`
    - Dashboard: zavolat “minikube dashboard” (v samostatném okně)
  - Pomocí yaml (yml) souboru (kubectl)
    - Příklad: `kubectl create -f deployment.yml`
    - NEBO: `kubectl apply -f deployment.yml`
  - <https://kubernetes.io/docs/concepts/overview/working-with-objects/object-management/>

# Deployment: příklad

- Vytvoří 1 deployment:
  - `kubectl create -f deployment.yml`
- Vrátí 1 deployment:
  - `kubectl get deployments`
- Vrátí 3 pody (tento deployment obsahuje 3 pody):
  - `kubectl get pods`
- Smaže deployment:
  - `kubectl delete -f deployment.yml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx-deployment
  labels:
```

```
    app: nginx
```

Počet replik podu

```
spec:
```

```
  replicas: 3
```

```
  selector:
```

Tímto způsobem deployment hledá pody, které spravuje

```
    matchLabels:
```

```
      app: nginx
```

```
  template:
```

Metadata a specifikace podu

```
    metadata:
```

```
      labels:
```

```
        app: nginx
```

```
    spec:
```

```
      containers:
```

```
        - name: nginx
```

```
          image: nginx:1.15.11
```

```
          ports:
```

```
            - containerPort: 80
```

# Deployment stateless aplikace

- Příklad na deployment stateless aplikace (nginx):
  - <https://kubernetes.io/docs/tasks/run-application/run-stateless-application-deployment/>
    - Obsahuje navíc:
      - Aktualizace podu
      - Scaling aplikace
- Rollback deploymentu:
  - <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/#rolling-back-a-deployment>



# Labels

- Labels jsou v Kubernetes klíčové pro organizování podů, je to jednoduchý pár “klíč: hodnota”:

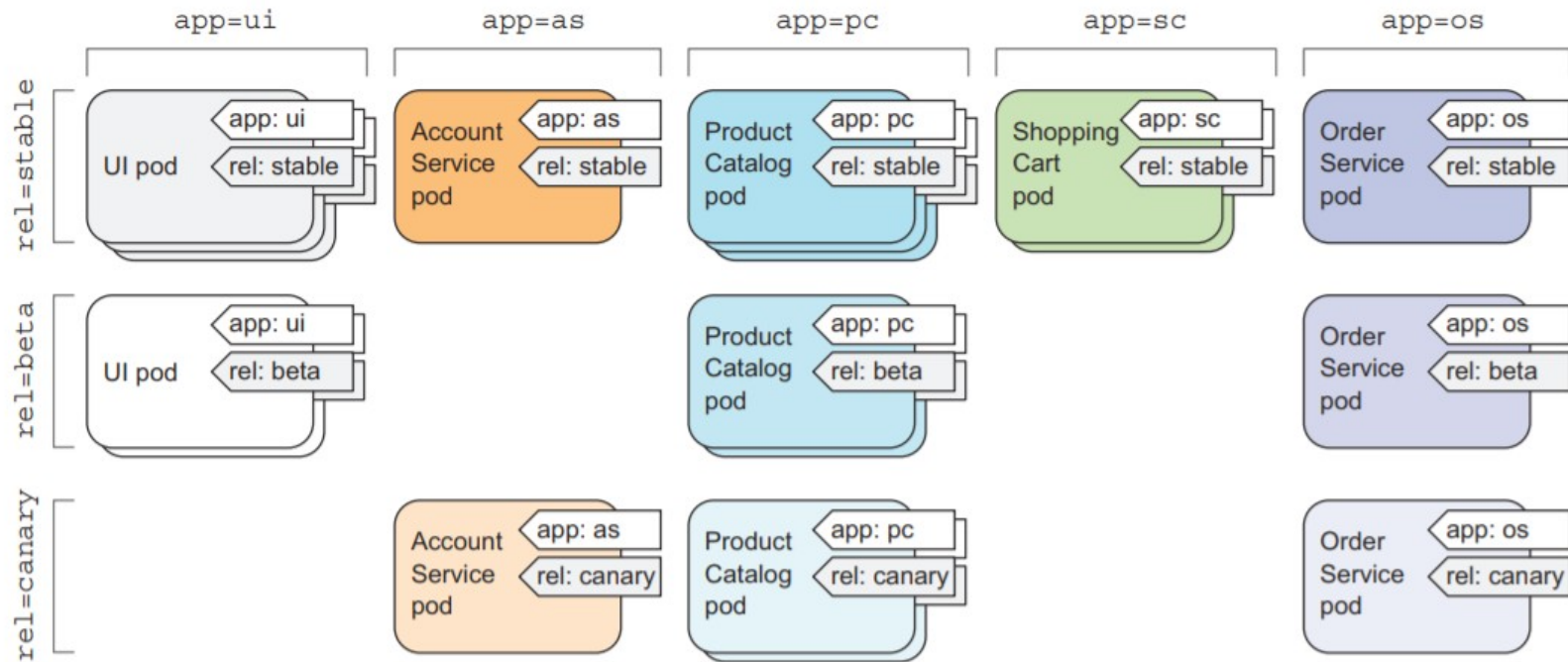


Figure 3.7 Organizing pods in a microservices architecture with pod labels

# Labels


Poznámka: S větším množstvím labelů se pracuje přes čárku, například:  
app=ui,rel=beta

- Výpis podů s labely:
  - `kubectl get pods --show-labels`
- Labely je možné měnit i za běhu:
  - `kubectl label pods POD_NAME env=debug --overwrite`
- Výpis podů s nějakým konkrétním labelem:
  - `kubectl get pods -l app`
- Výpis podů s nějakým konkrétním labelem a hodnotou:
  - `kubectl get pods -l app=nginx`
- Výpis podů bez nějakého konkrétního labelu:
  - `kubectl get pods -l '!app'`

# Labels

- Seznam uzlů:
  - `kubectl get nodes --show-labels`
- K uzlu “A” se přidá label “server=ssd”
  - `kubectl label nodes A server=ssd`

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    server: ssd
```



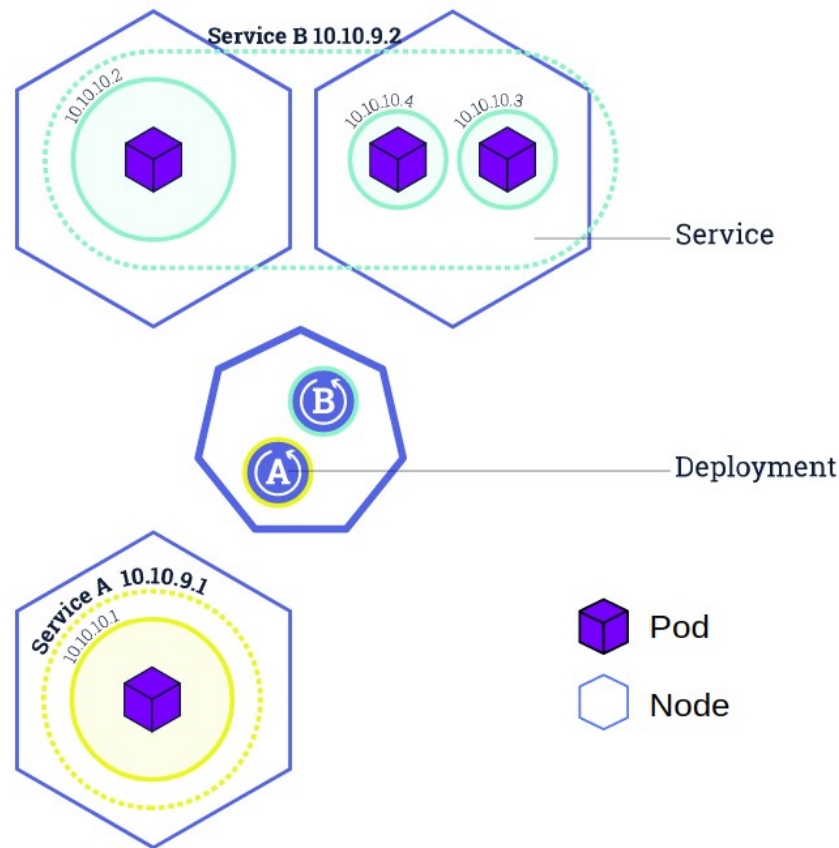
Tento pod bude běžet pouze na nodech,  
které mají label server=ssd

# Namespaces

- Pody je také možné organizovat do namespaces:
  - `kubectl get namespaces`
  - `kubectl get pods --namespace kube-system`
- Je možné vlastní namespaces vytvářet i mazat.
- Poznámka: Pody v různých namespaces nejsou nijak izolované.

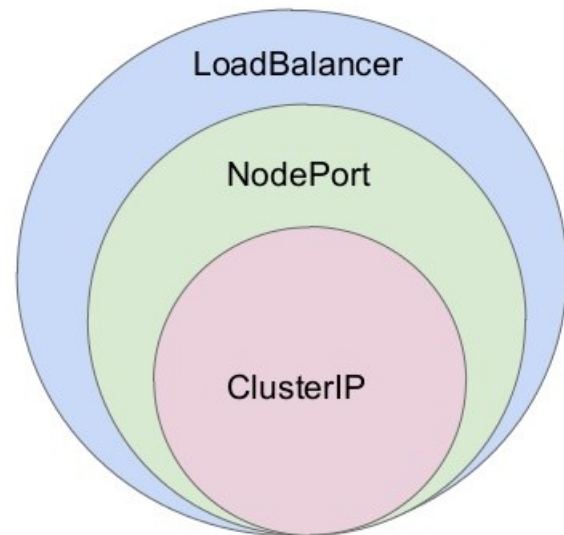
# Service

- Service slouží ke zpřístupnění podů mimo svůj cluster.
- Servicy mohou zpřístupňovat pody různým způsobem:
  - <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>



# ClusterIP vs NodePort vs LoadBalancer

- 
- <https://stackoverflow.com/questions/41509439/whats-the-difference-between-clusterip-nodeport-and-loadbalancer-service-types>
- 



# Service

apiVersion: v1

kind: Service

metadata:

name: nginx

spec:

type: LoadBalancer

selector:

app: nginx

Všechny pody s “app: nginx” labelem budou součástí této service

ports:

- name: http

port: 80

Port, na kterém bude tato service dostupná (navenek)

targetPort: 80

Port podu, na který se budou posílat požadavky

# Service: příklad na ClusterIP

- Stáhnout a uložit jako service.yml:
  - <https://gist.github.com/jirkapinkas/6795eaf74dc37d5a84e68ed22b623764>
- `kubectl apply -f service.yml`
- Tohle vytvoří ad-hoc pod, ve které se nachází curl:

```
kubectl run curl-nginx --image=radial/busyboxplus:curl -i --tty --rm
```
- Uvnitř podu je možné zavolat curl a tím se v rámci interního clusteru dostaneme z jednoho podu na druhý:
  - `curl http://nginx`



Tohle je hodnota metadata.name

Poznámka: service se “spáruje” s pody pomocí spec.selector.app



# Service: příklad na NodePort I.

- Vypíše seznam services:
  - `kubectl get services`
- Proveďte expose portu 80 (na něm běží nginx) na náhodný port:
  - `kubectl expose deployment/nginx-deployment --type="NodePort" --port 80`
- Získá IP adresu kde běží minikube (například 192.168.99.100):
  - `minikube ip`
- Vrátí na co je namapovaný port 80 (hodnota CLUSTER-IP, například 32701):
  - `kubectl get services`
- Mělo by fungovat něco jako:
  - <http://192.168.99.100:32701>

# Service: příklad na NodePort II.

- Vrátí název podu (například nginx-deployment-865bf46d65-6wn4n):
  - `kubectl get pods`
- Vypíše log:
  - `kubectl logs nginx-deployment-865bf46d65-6wn4n`
- Smaže service:
  - `kubectl delete service nginx-deployment`

# Service: Příklad na LoadBalancer

- Stáhnout a uložit jako service.yml:
  - <https://gist.github.com/jirkapinkas/3abe9e52811295f985bba21c1e5ce804>
  - `kubectl apply -f service.yml`
- Uvnitř podu je možné stejným způsobem jako na předcházejícím snímku přistoupit na: `http://nginx:8080`
- Když se zavolá:
  - `minikube tunnel` (tohle v jiném okně)
  - `kubectl get svc` (tady se získá hodnota EXTERNAL-IP)
- Tak se může přejít na:
  - `http://external-ip:8080` (na hostitelském počítači) a dostaneme se k service

# Ingress I.

- Základní nevýhoda LoadBalancer servis je, že každá LB service vyžaduje svůj vlastní load balancer se svojí public IP adresou. Ingress oproti tomu vyžaduje pouze jednu public IP adresu i když zprostředkovává přístup k X servisům.
- V Minikube je Ingress out-of-the-box disabled. Jak ho odblokovat?  
minikube addons list  
minikube addons enable ingress

# Ingress II.

**ingress.yml:**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myingress
  labels:
    name: myingress
spec:
  rules:
  - host: nginx.example.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: nginx
            port:
              number: 80
```

```
kubectl apply -f ingress.yml
```

```
kubectl get ingress
```

Tohle vrátí ADDRESS (ip adresa kde běží Ingress, například 192.168.49.2)

**/etc/hosts:**

```
# Když se přejde na nginx.example.com, tak se bude
# traffic směřovat na 192.168.49.2 (tam běží Ingress):
192.168.49.2  nginx.example.com
```

A nakonec vyzkoušíme:

```
curl nginx.example.com
```

Tady může být i “/podadresar”

Nginx service

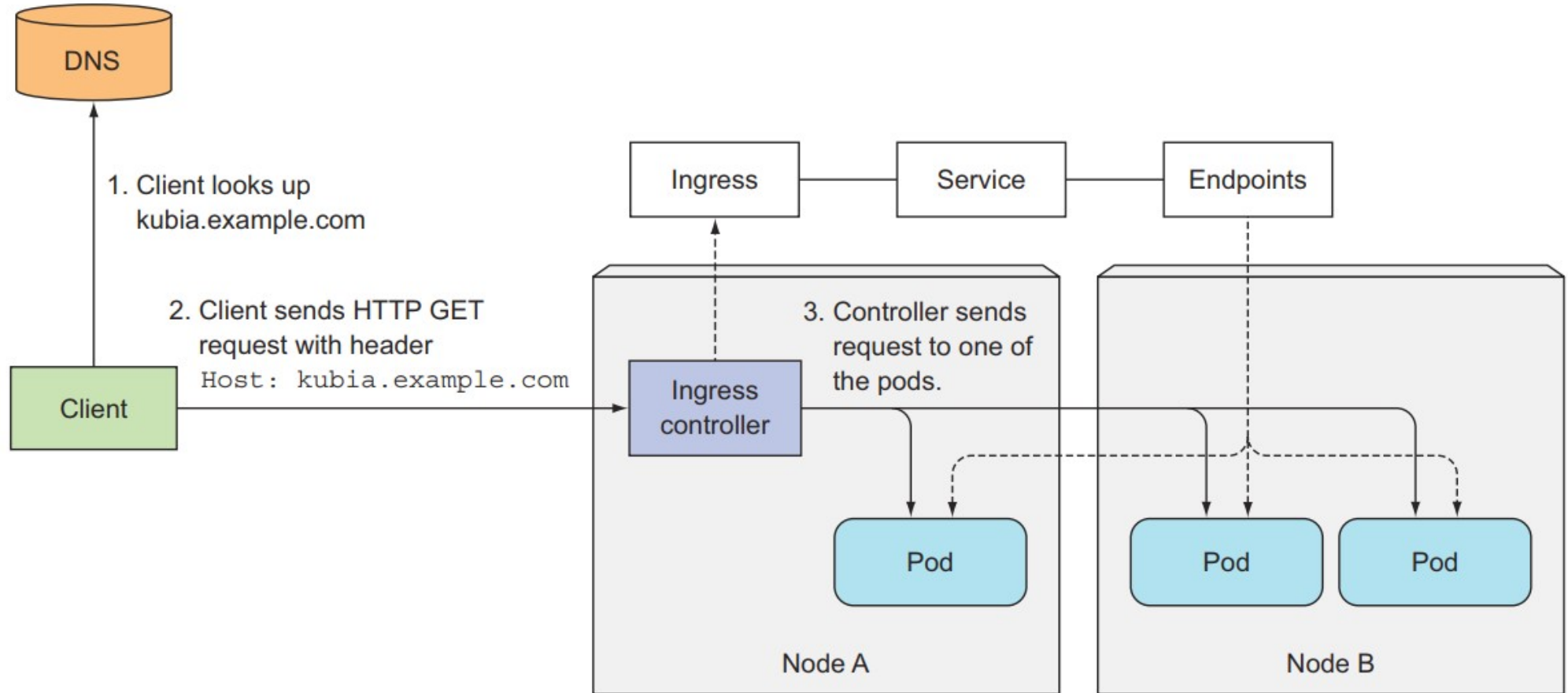
Port, na kterém běží aplikace v nginx service

Poznámka: V jednom souboru může být definováno větší množství hostů!

# Ingress Controllers

- Ingress pro svoji funkčnost vyžaduje tzv. controller. O co se jedná? V Kubernetes jsou controllery aplikace (skripty), které na základě nějakého vstupu (a aktuálního stavu clusteru) změní stav clusteru na nějaký požadovaný stav:
  - <https://kubernetes.io/docs/concepts/architecture/controller/>
  - <https://kubernetes.io/docs/concepts/workloads/controllers/>
- Existují i další ingress controllery:
  - <https://kubernetes.io/docs/concepts/services-networking/ingress/>
  - <https://kubernetes.github.io/ingress-nginx/>
  - <https://github.com/containous/traefik>

# Ingress Architecture



# Service: příklad 2

- Stáhnout a uložit jako service.yml:
  - <https://gist.github.com/jirkapinkas/c69ead93c7fb30d4d2832c8ee1a07efd>
- `kubectl create -f service.yml`
- `kubectl get services`
- Otestuje funkčnost service:
  - `minikube service my-service`
- `kubectl delete -f service.yml`



# External Service

## service-external.yml:

```
apiVersion: v1
kind: Service
metadata:
  name: external-seico
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
```

## endpoint-external.yml:

```
apiVersion: v1
kind: Endpoints
metadata:
  name: external-seico
subsets:
  - addresses:
    - ip: 185.8.239.196
    ports:
      - name: http
        port: 80
      - name: https
        port: 443
```

```
kubectl apply -f service-external.yml
kubectl apply -f endpoint-external.yml
kubectl run curl-nginx --image=radial/busyboxplus:curl -i --tty -rm
curl http://external-seico
```

# kubectl exec

- Jak se dostat dovnitř nějakého podu:
- Vrátí název podu (například nginx-deployment-865bf46d65-7b6b8):
  - `kubectl get pods`
- Spustí “sh” uvnitř nějakého podu:
  - `kubectl exec -it nginx-deployment-865bf46d65-7b6b8 -- sh`
    - Poznámky:
      - Vše po “--” bude vykonáváno uvnitř podu
      - Pro otestování “propustnosti” mezi pody je nejlepší používat curl. Pomocí ping je možné “pingnout” konkrétní pod, ale není možné pingnout service, protože používá virtuální IP.

# Ad-hoc test Ubuntu pod

- Spustí “sh” uvnitř nového podu “ubuntu”:
  - `kubectl run -it ubuntu --image=ubuntu -- bash`
- Smaže pod “ubuntu”:
  - `kubectl delete pod/ubuntu`

# Liveness probe I.

- Jakmile hlavní proces kontejneru spadne, tak Kubelet automaticky restartuje kontejner.
- Kubernetes také může kontrolovat jestli kontejner stále žije pomocí “liveness probe” jedním z těchto mechanismů:
  - A) HTTP GET probe zavolá GET request na IP adresu kontejneru + port + path (definuje se ve specifikaci podu).
  - B) TCP Socket probe otevře TCP connection na port kontejneru
  - C) Exec probe zavolá aplikaci uvnitř kontejneru a zkontroluje její návratový kód. Status kód nula = success, jinak failure.

# Liveness probe II.

- Best practices:
  - Pokaždé nastavujte initial delay aby mohla aplikace úspěšně nastartovat. Jinak se můžete dostat do nekonečné smyčky, kdy se aplikace snaží nastartovat, ale Kubernetes ji restartuje protože nenastartovala dostatečně rychle.
    - <https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/>
  - Pomocí tohoto příkazu se dozvíte důvod restartu podu:
    - `kubectl describe pod NAZEV_PODU`
    - <https://sysdig.com/blog/debug-kubernetes-crashloopbackoff/>
  - Liveness probe je samozřejmě best practice na produkci :-)
  - <https://stackoverflow.com/questions/33484942/how-to-use-basic-authentication-in-a-http-liveness-probe-in-kubernetes>

# Liveness probe: příklad I.

- minikube start
- eval \$(minikube docker-env)
- mvn spring-boot:build-image
- kubectl apply -f ../demo.yml
- kubectl apply -f ../demo-service.yml
- V dalším okně: minikube tunnel
- kubectl get services, získat EXTERNAL-IP a zavolat GET na:  
http://EXTERNAL-IP:8080/actuator/health Výsledek by měl být status: "UP"
- kubectl get pods Počet restartů (RESTARTS) by měl být "0"
- Zavolat GET na: http://EXTERNAL-IP:8080/simulateOutage
- To změní u http://EXTERNAL-IP:8080/actuator/health status: "UP" na status: "DOWN"
- Kubernetes si toho po chvíli všimne, otočí instanci serveru a když se zavolá kubectl get pods, tak bude vidět, že počet restartů (RESTARTS) se inkrementuje o 1.

# Liveness probe: příklad II.

- Zavolat GET na: `http://EXTERNAL-IP:8080/simulateCrash`
- To natvrdo ukončí běh hlavního procesu v kontejneru
- Kubernetes si toho po chvíli všimne, otočí instanci serveru a když se zavolá `kubectl get pods`, tak bude vidět, že počet restartů (RESTARTS) se inkrementuje o 1.
- Také je zajímavé zavolat: `kubectl describe pod NAZEV_PODU`

# Readiness Probe

- Obdobně jako existuje liveness probe, tak také existuje readiness probe. Používá se úplně stejným způsobem a slouží k tomu, aby se neposílaly požadavky klienta na server, který teprve startuje a není “ready” pro přijímání požadavků.
- Více informací o liveness a readiness probe ve Spring Boot:
  - <https://spring.io/blog/2020/03/25/liveness-and-readiness-probes-with-spring-boot>



# Volumes

- Volume v Kubernetes slouží ke stejnému účelu jako v Dockeru. Jenom jich je větší množství ...
  - emptyDir: Prázdný adresář pro temporary data
  - hostPath: Pro mount adresářů z filesystemu nodu dovnitř podu
  - gitRepo: Volume, který provádí checkout git repozitáře Od 1.16 DEPRECATED!
  - nfs
  - GcePersistentDisk (Google Compute Engine Persistent Disk), awsElasticBlockStore, azureDisk
  - configMap, secret: Speciální typy volumů pro konfigurační data
  - a další ...
    - <https://kubernetes.io/docs/concepts/storage/volumes/>

# hostPath příklad I.

Do spec.template se přidá:

```
spec:
  containers:
  - name: nginx
    image: nginx:1.15.11
    ports:
    - containerPort: 80
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html/
      readOnly: true
  volumes:
  - name: html
    hostPath:
      path: /data/html
      type: Directory
```

Celý soubor:

<https://gist.github.com/jirkapinkas/634e7a0062d84e79b28afa2827f47ff4>

# hostPath příklad II.

- Spustit minikube:
    - minikube start
  - Pak spustit v samostatném okně:
    - minikube mount /home/jirka/Desktop/html:/data/html
  - A nakonec nasadit aplikaci:
    - kubectl apply -f nginx-mount.yml
  - Minikube mount provede “mount” adresáře /home/jirka/Desktop/html dovnitř minikube do adresáře: /data/html
    - To se dá zkontrolovat tím, že se zavolá:
      - minikube ssh
      - ls /data/html
- Poznámka: Data se mountují, nekopírují se!!!

# TIP: ContainerCreating

- Když je pod stále ve stavu “ContainerCreating”, pak pomůže:
  - `kubectl describe pods`

# ConfigMaps & Secrets

# Proxy

- Spustí proxy:
  - `kubectl proxy`
    - Spouští proxy: <http://localhost:8001/> ... zde je k dispozici Kubernetes API
    - <https://kubernetes.io/docs/tasks/extend-kubernetes/http-proxy-access-api/>

# Job

- S Kubernetes je také možné spouštět jednorázové joby:
  - <https://kubernetes.io/docs/concepts/workloads/controllers/job/>
- Nebo opakované joby (cron):
  - <https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>
- Sekvence jobů:
  - <https://stackoverflow.com/questions/40713573/how-to-run-containers-sequentially-as-a-kubernetes-job>

# ReplicaSet

- ReplicaSet (dřív ReplicationController) slouží k nastavení replikování podů. Stejného výsledku ale docílíme pomocí Deploymentu a tudíž se ReplicaSet napřímo nepoužívá.
  - <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>



# DaemonSet

- DaemonSet zajišťuje, že na všech uzlech clusteru (na každém node) běží kopie podu. To se hodí například pro kontejnery, které zajišťují sběr logů z ostatních kontejnerů (filebeat, logstash), nebo pro kontejnery pro sběr metrik (různé prometheus exportery).
  - <https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/>

# Helm

- Helm = package manager pro Kubernetes
  - <https://helm.sh/>
  - <https://hub.helm.sh/>

# Cheatsheet

- <https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

# Strimzi (Kafka Operator)

- <https://strimzi.io/quickstarts/>

# MongoDB Community Operator

- <https://www.mongodb.com/blog/post/run-secure-containerized-mongodb-deployments-using-the-mongo-db-community-kubernetes-oper>