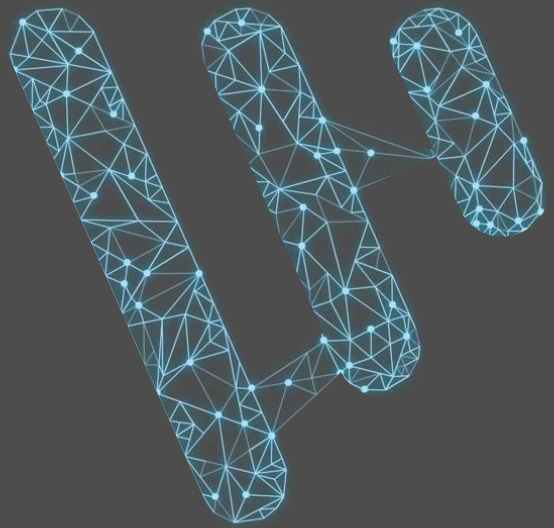


CO NÁS DNES ČEKÁ

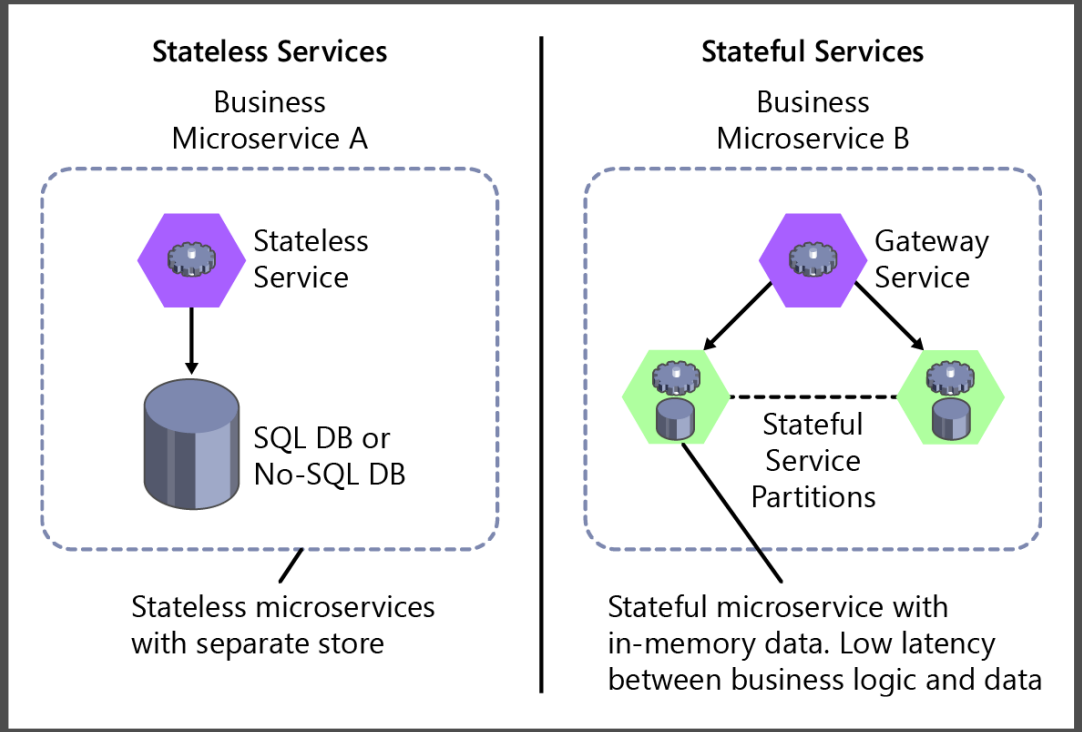
Agenda webináře

- Služby v kontejnerech
- Dockerfile
- Docker image v Kubernetech
- Bezpečnost
- ServiceMesh
- Ukládáme data, vystavujeme službu
- Typy struktur PODů
- Komponenty jako klíčová část
- Jak začít s vaší aplikací

- Q & A

STAVOVÉ VS NESTAVOVOVÉ

Stavový kontejner vyžaduje speciální zacházení



- Stavové (stateful)
  - uchovává/synchronizuje stav
  - specifické škálování
- Důležitost (obnovitelnost) stavu
  - Session Data
  - SQL Databáze
- Nestavové (stateless)
  - výkonnost
  - event processing
  - opakovatelnost
  - FaaS

MONOLITICKÝ KONTEJNER

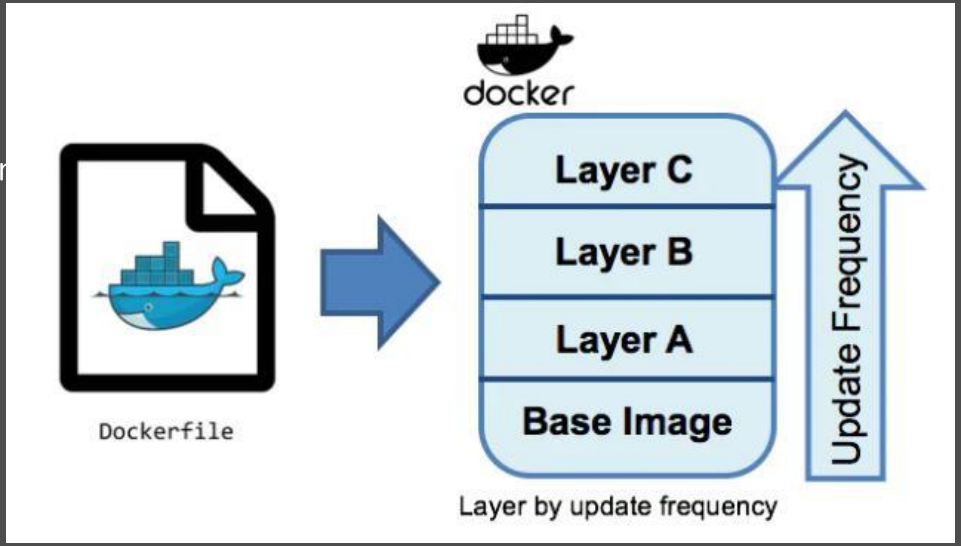
Existující téma vypadající jako cesta správným směrem

- Typicky stavový
- Omezený výkonem na HW
  - zásadně vstupuje do plánování zdrojů
- Neschopný efektivně škálovat
  - bez distribuce
  - plná redundace
- Složité opustit daný cyklus
  - I využívanou platformu
- Běžný problém
  - Legacy IT => CloudNative Apps

VYTVOŘENÍ JEDNODUCHÉ APLIKACE

Existují prototypy kontejnerů již v poodstatě pro všechno

- Buduje vrstvy (stage)
- Monolitický typicky:
  - velká IMAGE „all-in-one“
  - služby skrze init/systemd (privilegovaný kor
  - VOLUME /data, /var/lib, apod.
  - CMD cokoliv-persistentní
  - EXPOSE port(ů)
- Microservisní typicky:
  - Alpine/Micro IMAGE
  - vyjmenované balíčky + verze
  - COPY “služby“
  - VOLUME (?)
  - CMD jen-daná-služba/process
  - EXPOSE port(ů)



DOCKERFILE – POPIS KONTEJNERU

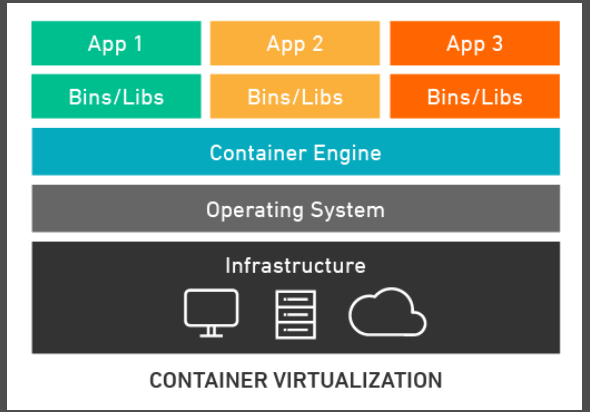
Dockerfile je zajetý a velmi dobře funkční model automatizace

- Tvoří předpis jak vyrobit kontejner

- Klíčová slova:
  - FROM
  - RUN
  - COPY/ADD
  - EXPOSE
  - VOLUME
  - CMD/ENTRYPOINT

- Kompatibilní např. s podman
- Řada nadstaveb – docker-compose, apod.

```
FROM python:3-alpine
WORKDIR /usr/src/app
EXPOSE 8000
COPY requirements.txt .
RUN pip install -qr requirements.txt
COPY server.py .
CMD ["python3", "./server.py"]
```

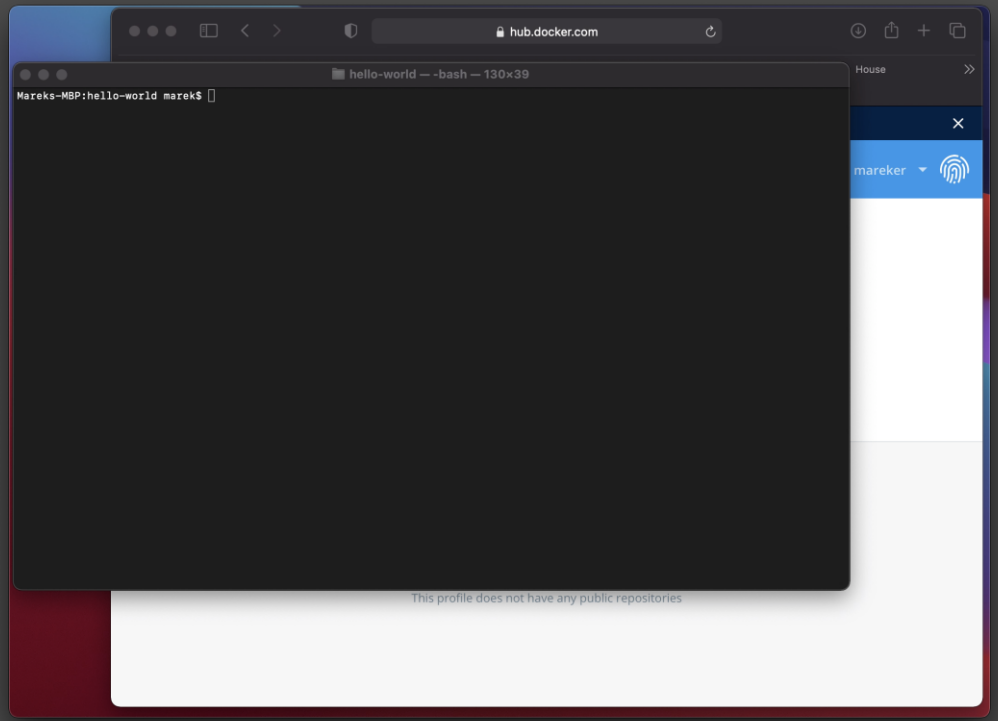


DOCKERFILE – JEDNODUCHÁ UKÁZKA

Dockerfile je jednoduchý, s registry/repository získává na přenositelnosti

- Připravená Hello-world aplikace v python3

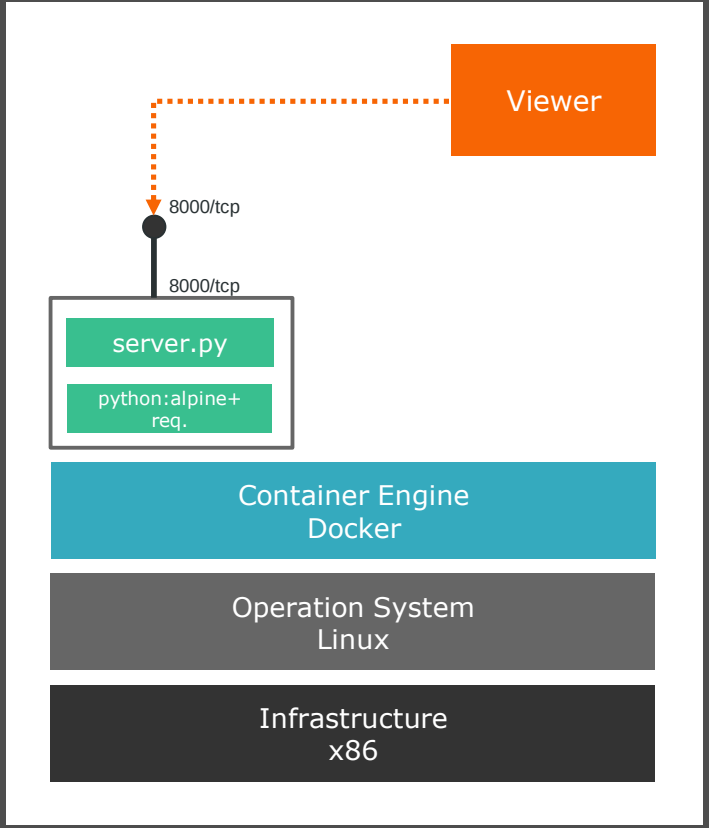
- Build, spuštění a push ven
- Upravení výstupu aplikace
- Nahradíme verze



STRUKTURA UKÁZKY

Zajistili jsme prostředí procesu a vypropagovali ven jeho port

- Single-host Docker
- Výstup jde přímo „ven“
- Nepoužili jsme žádné Volume/Bind
- Výstupem je Docker Image s requirements Linux/x86
- Uživatel může být kdekoliv (použili jsme localhost, není to však nutností)

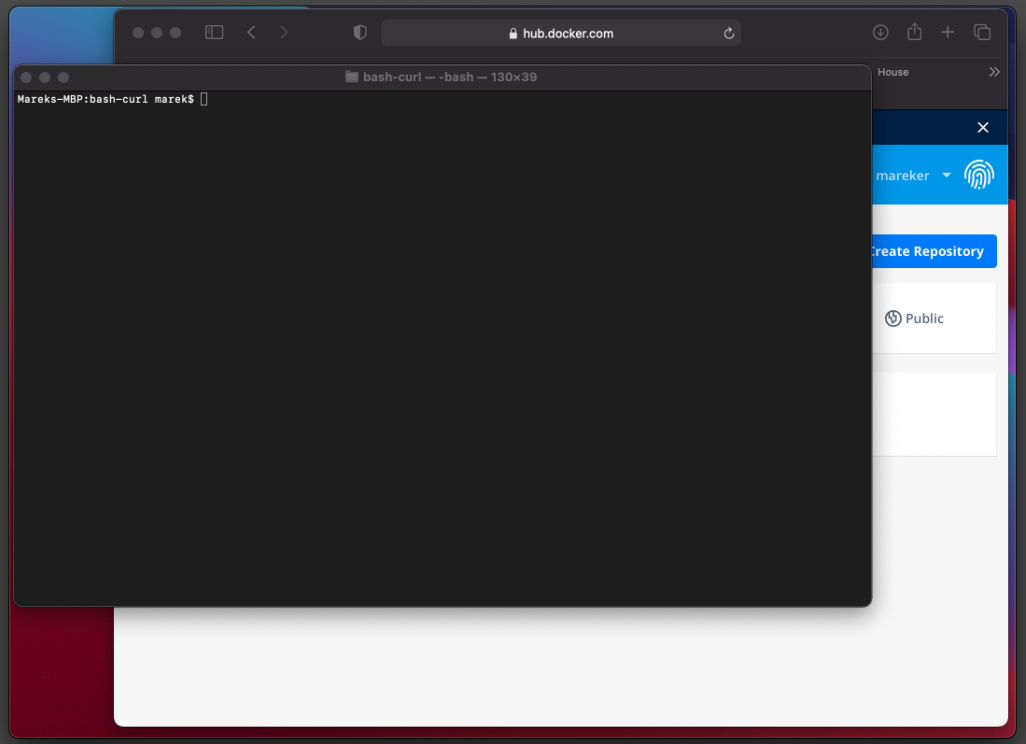


TOTÉŽ NAD KUBERNETY

Docker je skvělý a jednoduchý – Kubernety definují svoji strukturu přesněji

- Využijeme Tanzu Kubernety

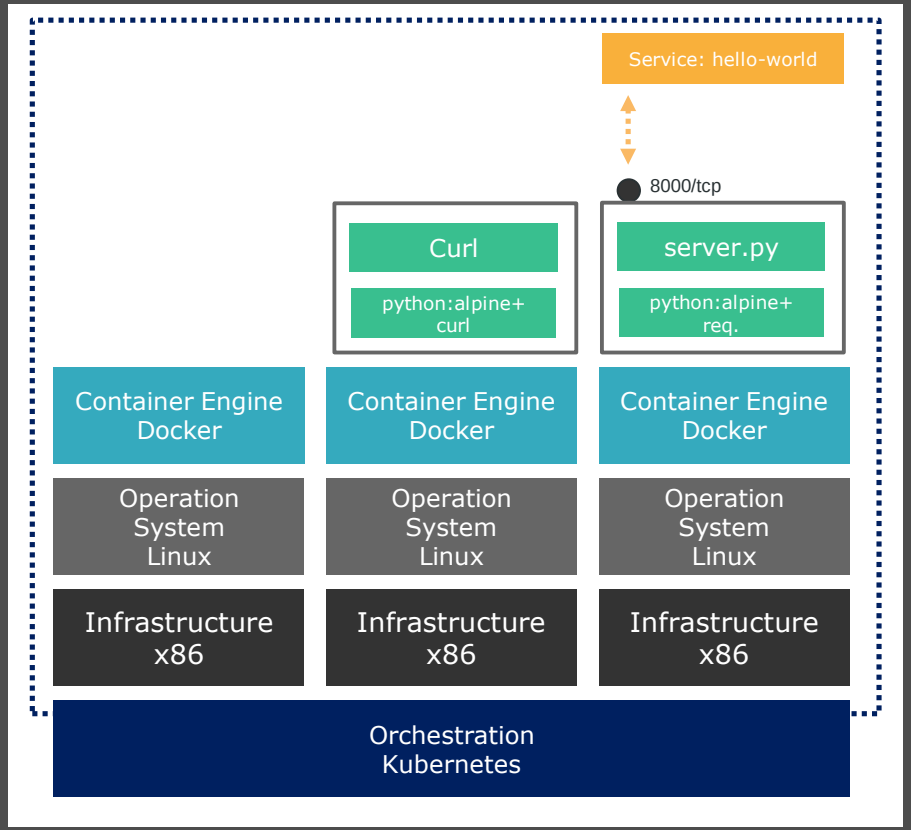
- Vytvoříme POD hello-world
- Vytvoříme Service hello-world
- Vytvoříme Docker Image bash-test, jeho push a z něj POD
- Využijeme ServiceMesh/Inner DNS Kubernet



STRUKTURA UKÁZKY

Ukázali jsme si, že PODy spolu komunikují

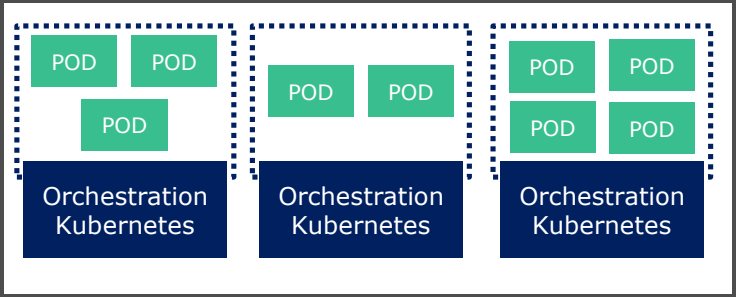
- Výstup nejde „ven“, zůstává v K8s
- PODy spolu uvnitř komunikují skrze ServiceMesh
  - Klíčová komponenta
- Nepoužili jsme žádné Volume
- Ani Deployments apod. – nadstavbu PODů
- Už nevíme, kde „co běží“
- Zatím jsme použili v podstatě jen Docker strukturu
- Bezpečnost
  - Skenování obrazů



BEZPEČNOST

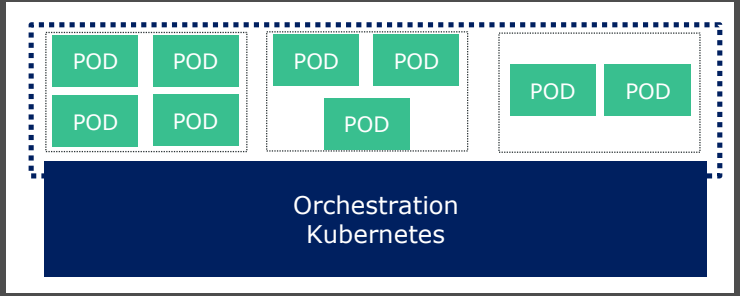
Je v kontejnerech velké téma – řeší se někdy až „násilně“

Oddělením Kubernetes clusterů



- Běžné pro DEV, TEST & PROD
- Odděluje do maximální úrovně – vidí se pouze vnější služby
- Oddělení týmů? – větší množství zdrojů
  - Zdroje se zase snáz škálují

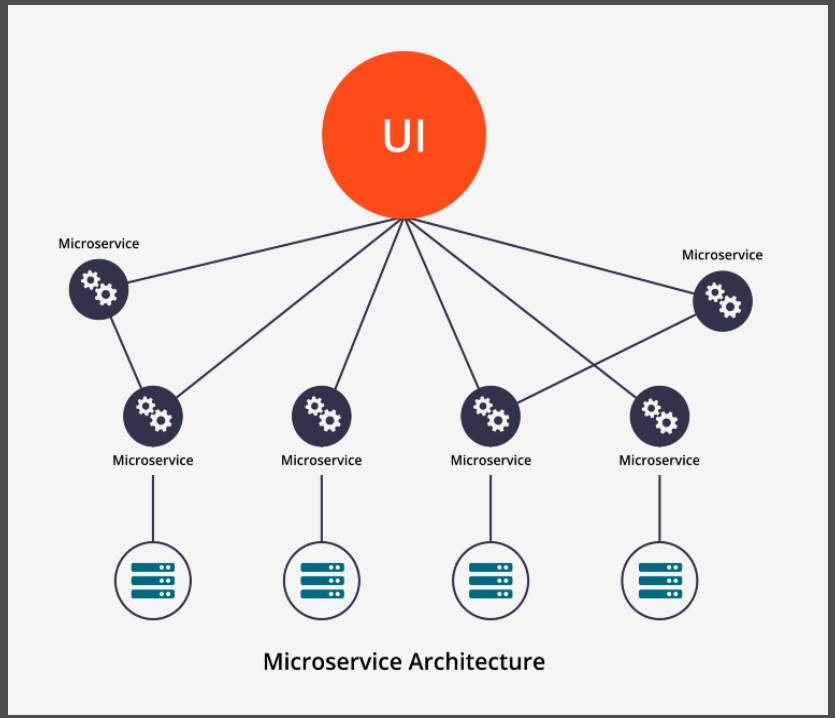
Využitím Namespace Kubernetů



- Namespace je základní dělení vždy
- Efektivně odděluje „struktury“, nikoliv data a logiku
- Existuje CNI s izolací (Canal)
- Projekt = Soubor Namespace – podporuje nyní Rancher

DŮLEŽITOST SERVICE MESH

ServiceMesh je nedílnou součástí MicroServisní architektury

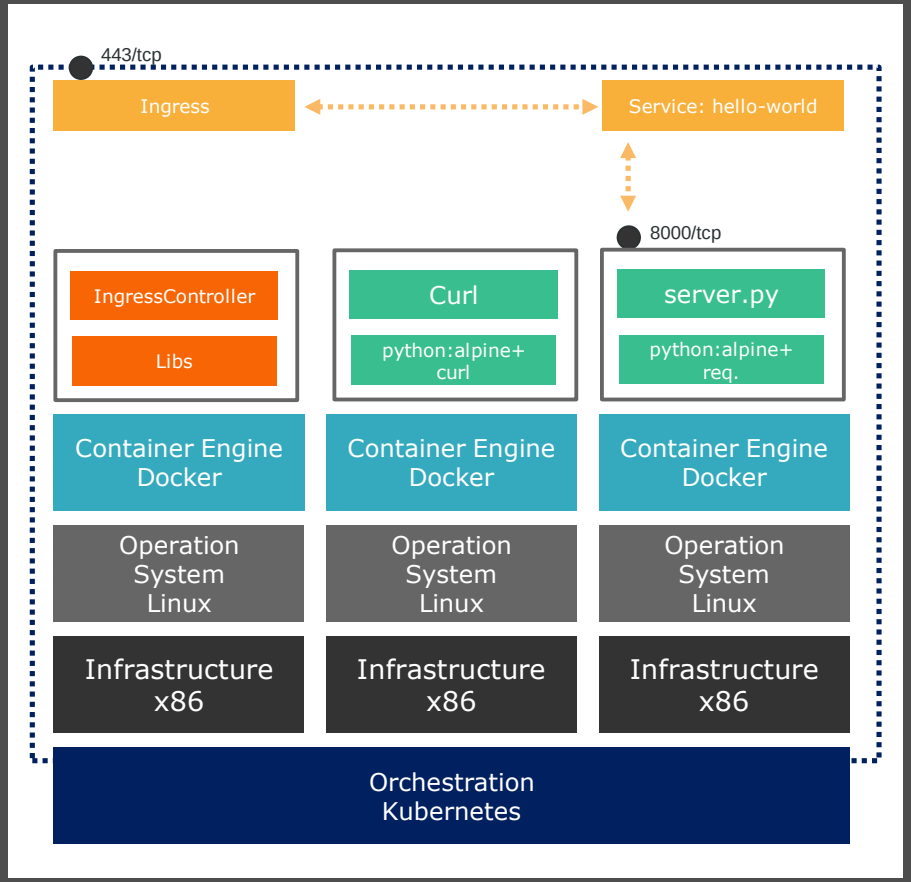


- Service se automaticky registrují
  - Dnes se aktivně používají i externě orientované Service
  - Minimalizuje služby propagované ven
- Dovoluje dynamiku
- V BareMetal zrealizovatelné např. skrze Avahi či Consul
  - Vyžaduje často hodně ruční práce
- Typicky
  - <service>.<namespace>.[svc].cluster.local
  - existují SRV záznamy pro Service (+ port + protokol)
  - <pod-ip>.<namespace>.pod.cluster.local

KDYŽ CHCEME DOSTAT SLUŽBU VEN

Možností je víc. Klasická: Ingress

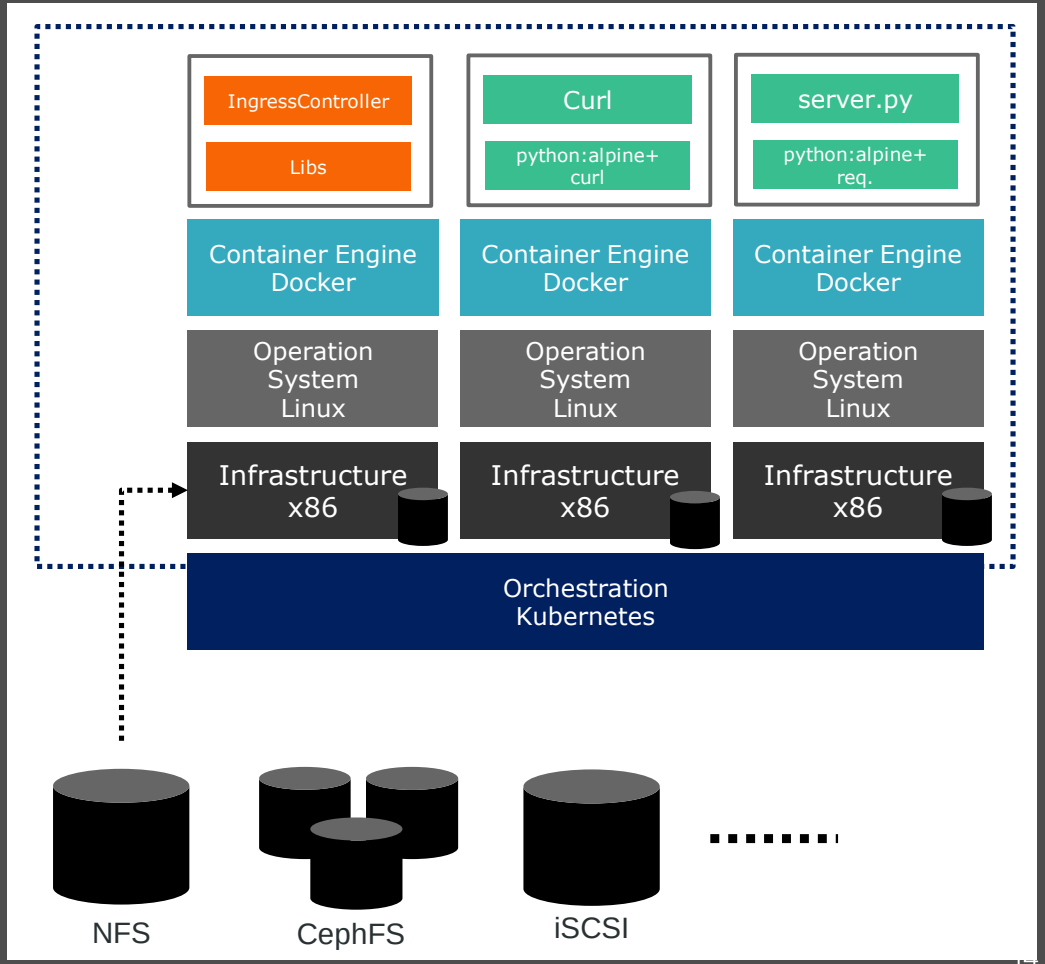
- Ingress Controller + Ingress
  - L7 balancing
  - typicky ve spojení s CertManager – správa SSL
  - Ingress je Balancing pravidlo
- NodePort
  - Vystaví na každý NODE na stejný port
  - Vhodné mít externí LB (i pro jednu AZ)
  - Rozumí na úrovni L4
- LoadBalancer
  - Typicky služba pro IngressController
  - L4 Balancing
  - Skrze MetalLB zajistí ExternalIP v ARP či BGP



KDYŽ CHCEME UKLÁDAT DATA

Persistent Volume a jeho přístupy

- PersistentVolume
  - liší se AccessMode
  - mnoho různých typů
- Sdílené FS
  - Typicky NFS
  - AccessMode: ReadWriteMany
- Lokální pro NOD
  - Lokální FS (LocalPath) či FC
  - ReadWriteOnce
- Standard CSI (Container Storage Interface)
  - Širší možnosti interface
  - Např. Snapshoty (zálohování)



DYNAMICKÁ KONFIGURACE

Kubernetes má velké množství různých specifických objektů

- Standardně konfigurace
  - jako součást ENV
    - může být velmi „nákladné“
  - jako soubor v rámci APP
    - nepřehledné
    - úprava => rebuild
- Kubernetes
  - nabízí ConfigMaps
    - key => value
    - nezabezpečené
  - Secrets
    - různé typy
    - zabezpečené
  - úprava => redeploy

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: test-application
      image: registry.comp.com/new-app
      command: [ "python3", "calculate.py" ]
      envFrom:
      - configMapRef:
          name: my-db-config
      - secretRef:
          name: my-db-access
  restartPolicy: Never
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-db-config
  namespace: default
data:
  DB_HOSTNAME: my-db
  DB_USERNAME: root
  DB_PORT: 3306
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-db-access
type: Opaque
data:
  DB_PASSWORD: MWYyZDFlMmU2N2Rm
```

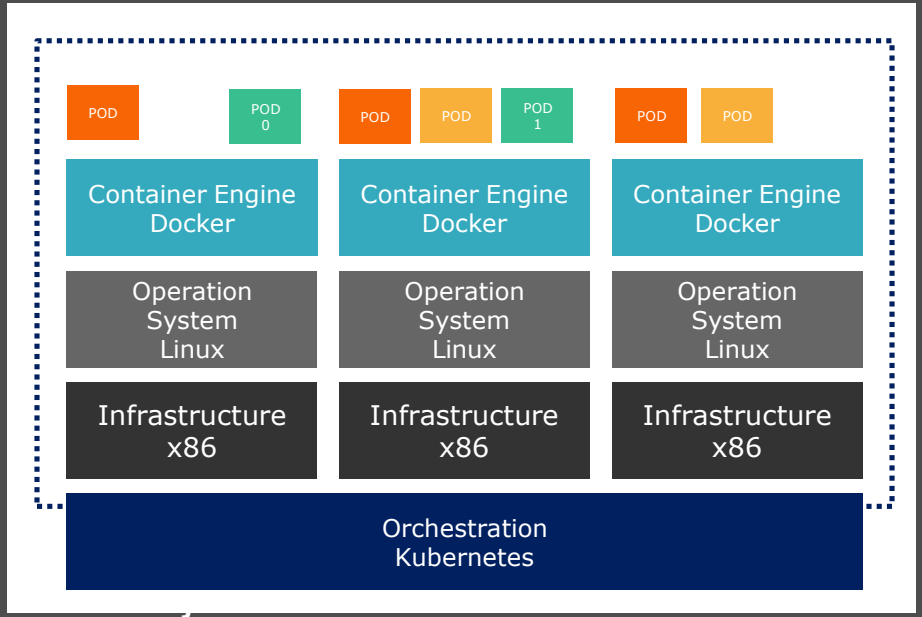
TYPY APLIKACÍ

Kubernetes je připravený na různé potřeby

- DaemonSet
  - POD na každém NODu, nestavový
  - Collector logů, monitoring

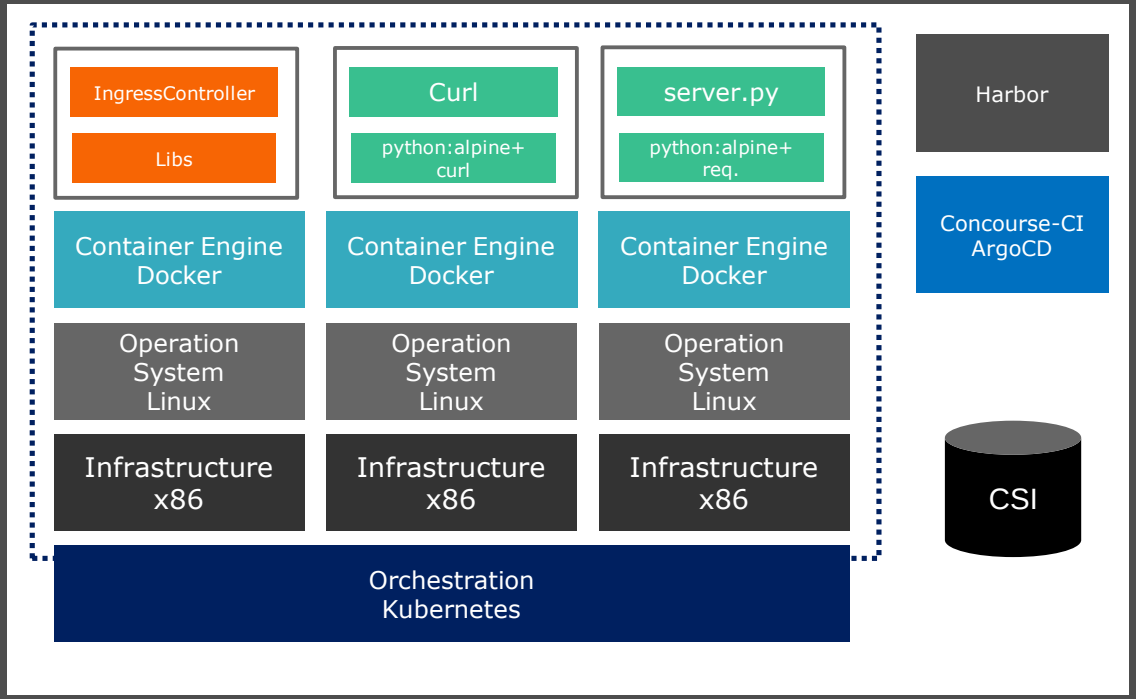
- StatefulSet
  - Shodné PODy v počtu dle Rep
  - Závislost PODů – 1 neexistuje bez 0, 2 bez 1 apod.
  - Primárně pro stavově orientované skupiny
  - Ne nutně na různých NODech

- Deployment
  - Shodné PODy v počtu dle Replica
  - Vznikají (a zanikají) nezávisle na sobě
  - Nejběžnější mechanismus pro App
  - Ne nutně na různých NODech



KOMPONENTY JSOU KLÍČOVÉ

Použité komponenty určí, jak náročný bude Day Two Ops

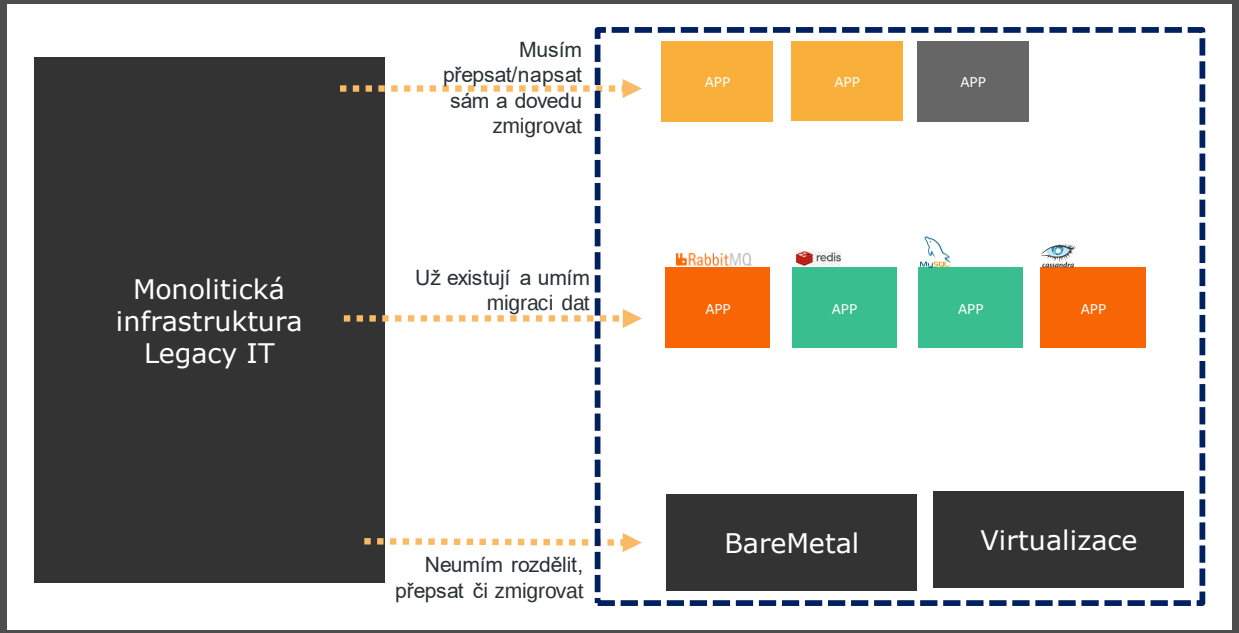


- Registry
  - Bezpečí vašich dat
  - Pravidelné skenování a synchronizace
  - Harbor
- CI/CD
  - Concourse-CI
  - Integrace GitLab
  - ArgoCD – vizualizace komponent
- Volume
  - CSI plugin
  - Rozšiřuje interface o podstatné funkce
- Doménová jména + certifikáty

JAK PŘEJÍT S VAŠÍ APLIKACÍ

Začněte dělením, pokračujte dělením

- Využijte komunitních zdrojů
  - Většina běžných App existuje připravená pro K8s
- Definujte si pravidla bezpečnosti
  - Z komunity přichází i hlouposti, nemusíte mít všechno
- Dívejte se na proces
  - Jste připraveni z 1 aplikace vyrobit 30 micro služeb?
  - Podporujete práci DevOps?
- Nemusíte začít vším
  - Začněte tím, že začnete



Q & A

DĚKUJEME ZA POZORNOST

MEDIA, TELCO & CLOUD services