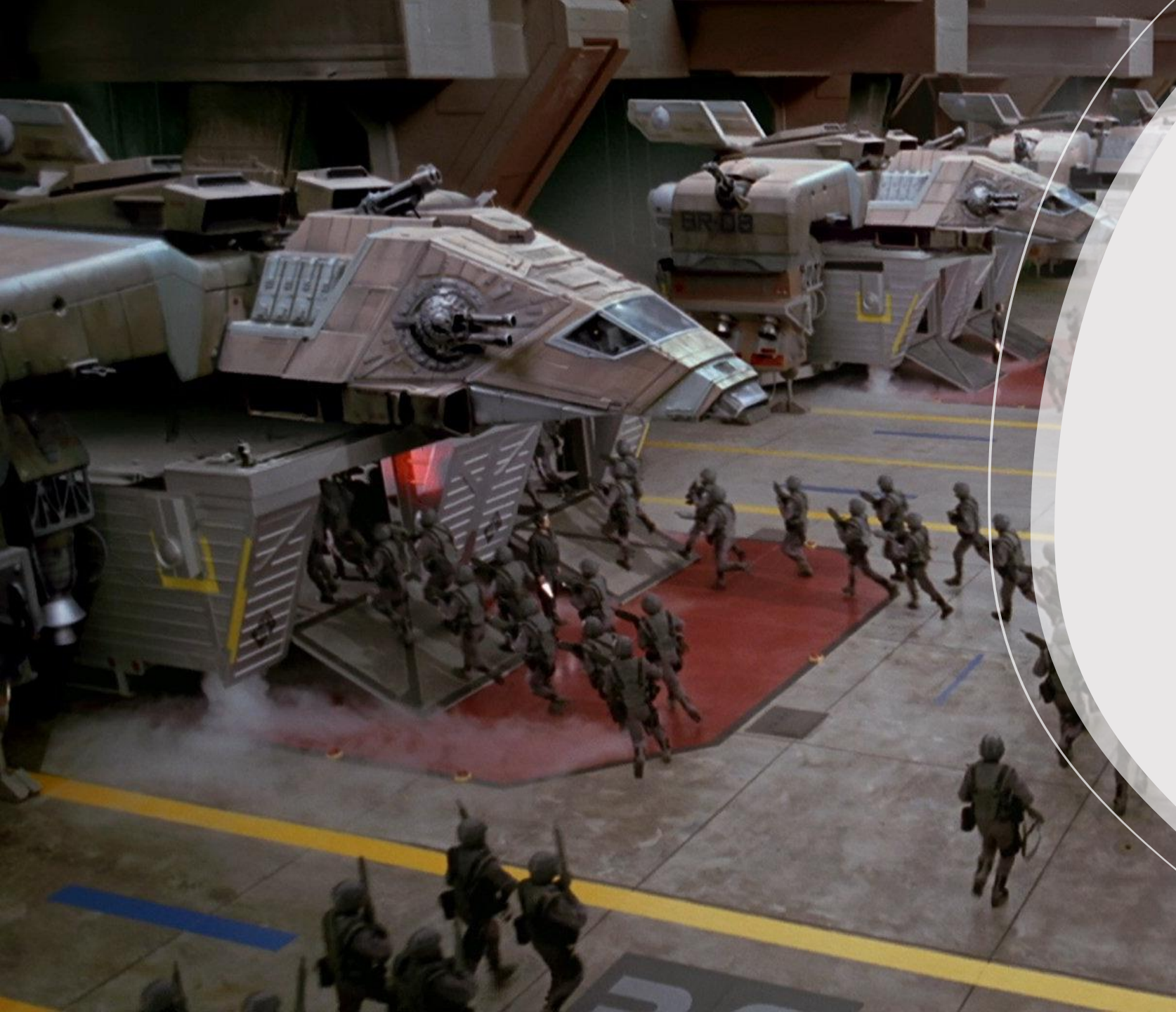
Nasadzovanie ako profík

Projekt 1

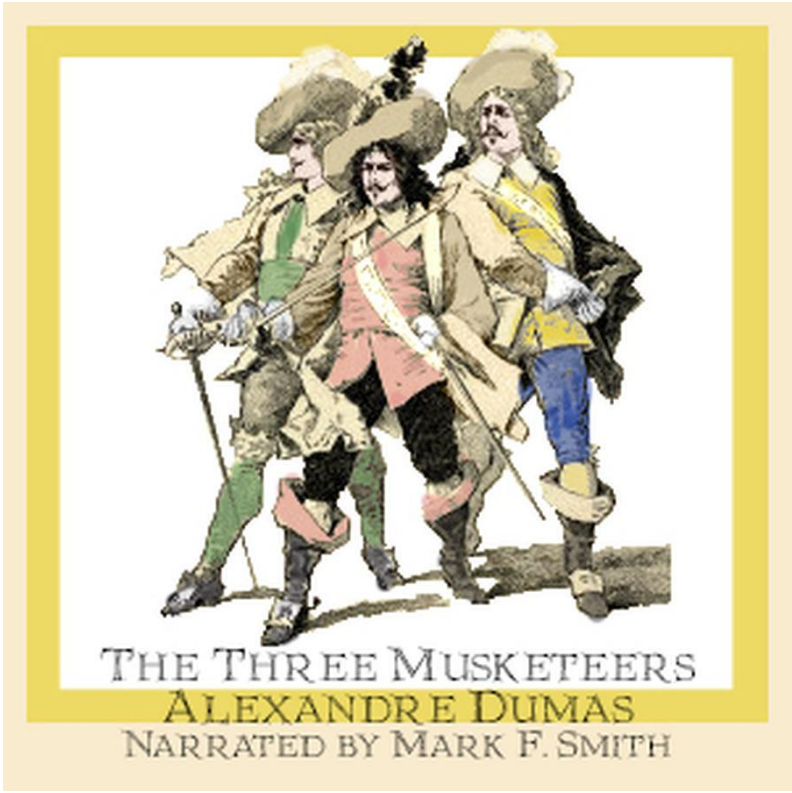


Nasadenie

- Kritéria
  - Bezpečnosť
  - Výkon servera
  - **Robustnosť a redundantnosť**
  - Cena

Orchestrátory kontajnerov

- Kontajnery
  - Komunikujú cez sieť
  - Je im jedno či bežia na jedinom OS, alebo na rôznych OS
- Čistý Docker (Compose)
  - Spúšťa kontajnery v rámci jedného OS
- Orchestrátor
  - Spúšťa kontajnery v rámci viacerých OS/serverov
  - Viac samostatných serverov sa správa ako jeden - **klaster**
  - Docker Swarm
    - Rozšírenie Docker Compose na viac strojov
    - Neujalo sa
  - Kubernetes (K8s)
    - Asi najviac používaný orchestrátor
    - Znalosti K8s sú veľmi žiadané na trhu práce
  - Nomad
    - Novší a jednoduchší



(Moderné) nasadenie na viac serverov

- High Availability Deployment
  - Aplikácia musí prežiť chybu na úrovni HW, OS, DB, aj v samotnej appke
- Potrebujeme klaster serverov
  - Ak jeden server vypadne z klastra...
  - ... orchestrátor automaticky presunie kontajnery na iný server
    - Pri DB toto musí podporovať aj ona samotná
  - Viete zlepšiť výpočtový výkon pridaním nového servera do klastra
  - Web appky vieme triviálne replikovať
  - Replikácia MySQL a DB obecne je trocha menej triviálna
    - Musí to podporovať samotná DB
    - Väčšina DB to už podporuje

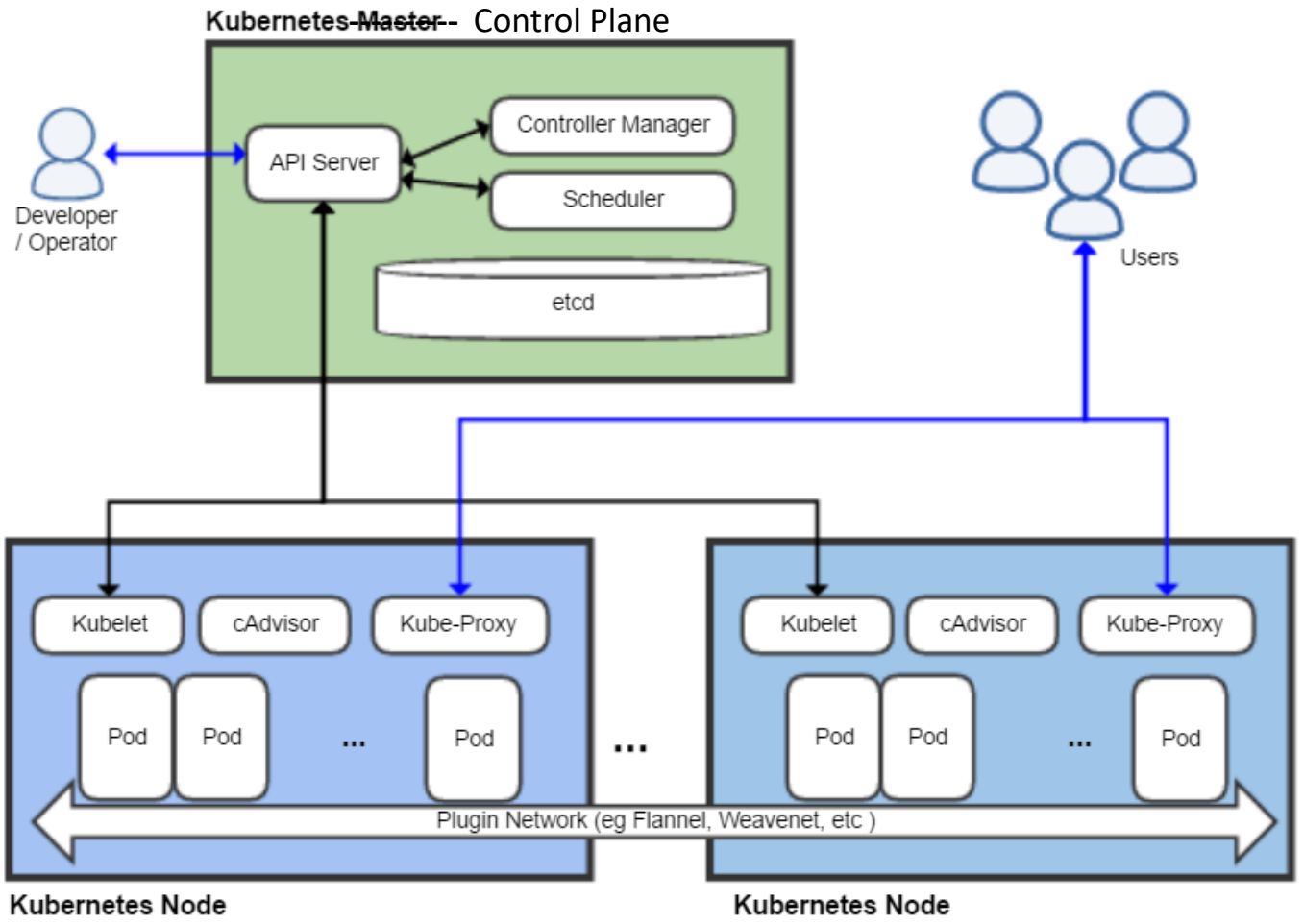
kubernetes

Čo je to?

- Názov
  - Číta sa „kjubernetis“
  - Z antickej gréčtiny – kormidelník
  - Skratka „K8s“
- Momentálne „ten“ orchestrátor
- „Cloudový“ operačný systém
- Pôvodne od Googlu
- OSS pod Apache 2.0
- Napísaný v Go
- Postavený okolo containerd (engine Docker-a)
  - Teda spúšťa OCI (Docker) kontajnery
- Okrem containerd však s Dockerom nemá takmer nič spoločné
  - Úplne odlišní klienti (CLI aj GUI) aj knižnice
- Dostupný ako self-hosted aj ako CaaS

Architektúra

- K8s vie bežať na jedinom serveri
- Alebo aj na mnohých serveroch
- Jeden OS/server = node (uzol)
- Pod – bežiaca aplikácia
- Dva základné typy nodov
  - ~~Master~~ Control Plane (CP)
  - Worker
- Control Plane
  - Drží metadáta o klastri a kontajneroch
  - Robí údržbu (rekonciliácia)
  - etcd – interná key-value DB
  - API Server – REST pre administráciu
- Worker (W)
  - Spúšťa pody
- Pri 1-nodovom klastri
  - Control Plane = Worker
- Klaster by mal mať
  - 3-6 CP a veľa W
- Podpora pre AZ (availability zones)
- K8s potrebuje port **6443**



Limity klastra

- Odporúčané limity
  - Max 110 Podov v Node
  - Max 5000 Nodov
  - Max 150 000 Podov v klastri
  - Max 300 000 kontajnerov v klastri

Distribúcie

- K8s má mnoho „variánt“
  - Podobne ako GNU/Linux distrá (Debian, Arch, Ubuntu, ...)
- Cloudové
  - Amazon EKS (Elastic Kubernetes Service)
  - Microsoft AKS (Azure Kubernetes Service)
  - Google GKE (Google Kubernetes Engine)
- Self-hosted
  - Red Hat OpenShift
  - SUSE Rancher
  - „Čistý“ Kubernetes
  - **Kubes (K3s)**
    - „Lightweight“ K8s
    - Takmer čistý K8s, ale o dosť jednoduchší na rozbehanie
    - Najčastejšie používaný

Špecializované „implementácie“ K8s

- KIND (Kubernetes in Docker)
  - Na lokálne testovanie nasadenia a vývoj/testovanie operátorov
- Minikube
  - Špecializovovaný na malé 1-node nasadenia
- Docker Desktop
  - Docker Desktop má v sebe aj 1-node K8s

Inštalácia K3s

- Najjednoduchšie je si prenajať klaster od cloudového poskytovateľa
- Vieme si ho nainštalovať aj sami
  - Návod pre začiatočníkov https://k3s.rocks
- Každý klaster má **kubeconfig**
  - Súbor v /etc/rancher/k3s/k3s.yaml
  - Slúži na prihlásenie sa do klastra ako admin
  - Môžeme mať viacero kubeconfig-ov s rôznymi oprávneniami
    - Napr. read-only kubeconfig, alebo iba s prístupom do určitých častí klastra
  - Pozor na default self-signed certifikát pri prihlásení z iného PC
- kubectl
  - CLI klient
  - Ak v Docker Desktop zapnete K8s, tak ho už máte u seba
  - Na pripojenie na vzdialený klaster potrebuje kubeconfig
    - Cez flag --kubeconfig, alebo premennú prostredia KUBECONFIG
    - kubectl get pods -A # vypise vsetky pody
    - kubectl apply –f entrance.yaml # nasadi nieco na cluster
- LENS
  - GUI klient
  - https://k8slens.dev/, alebo lepšie https://freelensapp.github.io/

K3s na VPS

- Majme 3 VPS
  - VPS_1
  - VPS_2
  - VPS_3

- Na VPS_1 nahráme K3s podľa návodu https://k3s.rocks/install-setup/
  - Na ostatných VPS urobíme to isté, ale naviac s premennými K3S_TOKEN a MASTER_IP (je to v návode)
- Čo s doménou?
  - Možnosť 1.: Upravíme A/AAAA DNS záznam v doméme
    - ***.entrance.sk** -> všetky 3 IP adresy VPS
    - Máme tak implicitný DNS load-balancing
  - Možnosť 2.: Alebo si urobíme ďalší VPS a tam nahodíme NGINX/Traefik/HAProxy na load-balancovanie
    - To dáva viacero výhod oproti DNS load-balancovaniu, ale musíme sa o to starať
    - Vieme detekovať padnutý Node a teda na neho nepresmerovávať requesty
    - Vieme terminovať TLS mimo klastra

Správanie appky na klasrti

- Správanie appky na klastri
  - Majme web appku, ktorá beží na VPS_2 (to si klaster rozhoduje viac-menej sám)
  - Nech má adresu app.entrance.sk (SPOILER: to vieme nastaviť v klastri cez Ingress)
  - U seba v browseri zadáme app.entrance.sk
  - DNS vráti jednu z troch IP adries a pošle požiadavku na príslušný VPS
    - Browser pošle požiadavku na VPS_1
  - K8s na tej VPS sa pozrie, či práve na ňom beží appka
  - Ak nie, tak prepošle požiadavku na správnu VPS

Pod

- Základná jednotka na nasadenie
- Každý pod má
  - IP adresu v rámci klastra
  - **1**-N kontajnerov
    - Kontajnery v pode zdieľajú sieť
      - Narozdiel od Dockera, kde je každý kontajner plne izolovaný
    - Kontajnery v pode sú na tom istom node
    - Zvykne byť **iba 1 kontajner** na pod
    - Alebo jeden „hlavný“ kontajner a niekoľko „obslužných“ kontajnerov
      - Obslužný kontajner – napr. zálohuje dáta z „hlavného“ kontajnera atď.
  - Priradené HW zdroje
    - CPU (v milijadrách)
    - RAM (v MiB)
    - GPU (celé GPU, alebo časť GPU*)
    - *Requests* – zdroje pri spustení kontajnera
      - Súčet requestov všetkých kontajnerov v klastri MUSÍ byť < súčet zdrojov nodov
    - *Limits* – maximálne možné zdroje kontajnera
    - Requests a limits nemusíme nastavovať, ale zvykne sa

```
apiVersion: v1
kind: Pod
metadata:
  name: keycloak
spec:
  containers:
  - name: keycloak
    image: jboss/keycloak:latest
    resources:
      requests:
        cpu: "300m"
        memory: "1000Mi"
      limits:
        cpu: "2200m "
        memory: "4000Mi"
```

Workloads

- ReplicaSet
  - Nadstavba nad Pod
  - Množina identických Podov
  - Cez ReplicaSet nastavujeme škálovanie
- **Deployment**
  - Nadstavba nad ReplicaSet
  - Najčastejšie používaný na webové aplikácie
  - Správa verzovania
    - Aktualizácia aplikácie na novú verziu
    - Rollback na staršiu verziu
- **StatefulSet**
  - Nadstavba nad ReplicaSet
  - Používaný na databázy
- DaemonSet
  - Nadstavba nad Pod
  - Zabezpečí, že Pod beží na každom Node

```
apiVersion: v1
kind: Deployment
metadata:
  name: keycloak
spec:
  replicas: 3
  selector:
    matchLabels:
      app: keycloak
  template:
    metadata:
      name: keycloak
    spec:
      containers:
      - name: keycloak
        image: jboss/keycloak:latest
        resources:
          requests:
            cpu: "300m"
            memory: "1000Mi"
          limits:
            cpu: "2200m "
            memory: "4000Mi"
```

Service

- Load balancer pre workload
- Zabezpečuje, že jednotlivé pody vo workloade sa „správajú ako jeden“
- Typy
  - **ClusterIP**
    - IP a interný DNS záznam v rámci klastra
      - Napr. keycloak.default.svc.cluster.local
    - Load balancer v rámci clastra
  - LoadBalancer (verejný)
    - Zverejní workload mimo klastra
      - Napr. keycloak.entrance.sk
    - Load balancer v rámci internetu
    - Potrebujeme mať kúpenú doménu pre náš klaster
    - Drahé v prípade cloudových K8s
      - Za každú sa platí extra
  - NodePort
  - ExternalName
  - Headless

```
apiVersion: v1
kind: Service
metadata:
  name: keycloak
spec:
  selector:
    app: keycloak
  type: ClusterIP
  ports:
    - port: 8080
      targetPort: 8080
```

Ingress

- Ingress je interné reverzné proxy pre K8s
- K8s iba deklaruje „rozhranie“
- Potrebujeme dodať implementáciu
  - NGINX Ingress (default v čistom K8s)
  - Traefik Ingress (default v K3s)
  - Istio Ingress Gateway (má brutálne veľa fíčur pre mikroslužby – service mesh)
- Je to novší spôsob ako zverejniť appku na internet
  - Vytvoríme si Service typu ClusterIP
    - Teda iba interná služba
  - A k nej vytvoríme Ingress objekt
    - Napr. https://auth.entrance.sk:443 -> Service keycloak, port 8080
  - Ak sme na cloude, tak ušetríme $ oproti LB Service
  - Vieme aj robiť TLS pre appky na úrovni klastra
    - Napr. Let’s Encrypt

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak
  annotations:
    spec.ingressClassName: traefik
    cert-manager.io/cluster-issuer: letsencrypt-prod
traefik.ingress.kubernetes.io/router.middlewares:
default-redirect-https@kubernetescrd
spec:
  rules:
    - host: auth.entrance.sk
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: keycloak
                port:
                  number: 8080
  tls:
    - secretName: keycloak-tls
      hosts:
        - auth.entrance.sk
```

Cert Manager

- Je to „plugin“ do K8s
- Inštaluje sa ako každý iný Deployment
  - kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.14.5/cert-manager.yaml
- Automatizované vydávanie a obnova TLS certifikátov
- Self-signed cert
- ACME (Let’s Encrypt, ZeroSSL, …)

Registrácia Let’s Encrypt

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    email: janko.hrasko@gmail.com
    preferredChain: ''
    privateKeySecretRef:
      name: letsencrypt-prod
    server: https://acme-
v02.api.letsencrypt.org/directory
    solvers:
      - http01:
          ingress:
            class: traefik
```

CSI - Container Storage Interface

- Rozhranie pre úložiská dát
- Implicitne „local“ CSI
- Ak Workload potrebuje prístup k úložisku
  - Treba vytvoriť PersistentVolume a PersistentVolumeClaim pre Workload
  - Na každom Node bude mať Pod z Workloadu prístup k úložisku toho Node
  - Avšak úložisko medzi Nodami **nie je** zdieľané
    - Ak máme tri repliky na troch Nodoch, tak každá replika má svoje vlastné dáta
      - Repliky nevidia dáta iných replík
    - Pri DB nám to (až tak) nevadí, tie si zdieľanie vedia riešiť samé
- Distribuované implementácie CSI
  - Umožňujú zdieľať disky medzi nodami
  - Nevhodné však pre DB - výkon
  - Longhorn, Ceph

Konfigurácia objektov

ConfigMap

- Objekt, ktorý drží textové dáta vo forme kľúč-hodnota
- V prog. jazykoch sa reprezentuje ako (Hash)Map<String, String>

Secret

- Objekt, ktorý drží base64 dáta vo forme kľúč-hodnota
- V prog. jazykoch sa reprezentuje ako (Hash)Map<String, byte[]>
- Určený pre heslá a prihlasovacie údaje.
- K8s nemá skutočné šifrovanie pre Secrets
  - Dá sa dodať plugin
    - Ale väčšinou to je over-kill

Nasadzujeme do K8s

Entrance App

- Máme appku, ktorá sa skladá z
  - React FE (**Entrance Web**)
  - Spring Boot BE (**Entrance API**)
  - **Keycloak**
  - **MySQL**
  - **PostgreSQL**
- Pozri https://gitlab.science.upjs.sk/pro1a-2024/entrance/-/tree/master/prod_deployment/kubernetes

Pokročilé nasadzovanie

Helm

- Balíčkovací manažér pre K8s
- *Chart* formát - nadstavba nad YAML
  - Jedná sa o TAR balík rôznych YAML súborov – objektov, ktoré chceme nasadiť
  - Podpora pre šablónovanie YAML
  - Podpora pre podmienky a cykly v YAML
- Ak nasadíme HelmChart YAML objekt
  - Helm si stiahne príslušný TAR
  - Rozbalí a aplikuje zadané hodnoty z HelmChart do YAML šablón
  - Evaluuje podmienky a cykly v YAML šablónach
  - Rozšablónované YAML súbory nasadí do K8s
- Čeknite bitmami/mysql a bitnami/postgresql pre HA nasadenie DB

```
helm install my-release oci://registry-1.docker.io/bitnamicharts/postgresql
```

Operátor

```
func (c *EntranceController) Reconcile(
    ctx context.Context,
    req ctrl.Request
) (ctrl.Result, error) {
    entrance := &entrancev1.Entrance{}
    err := c.client.Get(ctx, req.NamespacedName, entrance)
    if err != nil {
        c.Log.Error(err, "unable to fetch entrance")
        return ctrl.Result{}, client.IgnoreNotFound(err)
    }

    ...

    c.Log.Info("successfully reconciled entrance")
    return ctrl.Result{}, nil
}
```

- Rozšírenie („plugin“) do K8s
- Štyri časti
  - **CRD** – Custom Resource Definition
    - Schéma pre vlastné JSON/YAML objekty
  - REST API Server
    - To si porieši K8s
  - REST API Client
    - Vieme si vygenerovať, alebo použiť dynamický klient
  - **Kontrolér**
    - Sleduje život nejakých objektov
      - Zvyčajne podľa CRD
    - Implementuje metódu Reconcile
    - Reconcile definuje „čo sa má stať“
      - Napr. bude pri vytvorení objektu Entrance nasadzovať našu appku Entrance
    - Rekonciliácia sledovaného objektu sa spustí
      - pri spustení kontroléra
      - pri zápise nejakého objektu
      - periodicky každých N minút
- Čeknite mysql/mysql-operator a zalando/postgres-operator

Čo sme vynechali počas semestra

- Zďaleka sme nepokryli všetko, čo treba na vývoj väčších aplikácií

- CI/CD pipelines
  - Lebo niekto zaspal…

- Telemetria (OpenTelemetry – OTEL)
  - Realtime sledovanie stavu aplikácií a klastrov

  - Metriky (Prometheus)
    - Využitie CPU, RAM, ..., počty a trvanie požiadavok, ...
  - Tracing (Jaeger)
    - Trasovanie požiadaviek v mikroslužbách
  - Vizualizácia, Alerting, Dashboardy (Grafana/Plutono)
  - Logy (Loki/Vali, ElasticSearch/OpenSearch)
    - Centralizovaný zber logov


Live
Long
and
Prosper