linuxpolska

Szkolenie *Kubernetes* - Agenda

1. Podman
    a. Przegląd zagadnień związanych z Podman
        i. Podman to narzędzie do zarządzania kontenerami, które jest alternatywą dla Dockera. Główne różnice między Podmanem a Dockerem to:
            1. Bezpieczeństwo: Podman nie wymaga działania usługi daemon i może być używany bez uprawnień administratora (rootless containers).
            2. Kompatybilność: Obsługuje obrazy i polecenia zgodne z Dockerem.
            3. Podman vs Docker:
                a. Docker działa w modelu klient-serwer, a Podman działa jako samodzielne narzędzie.
                b. Podman pozwala na uruchamianie kontenerów bez potrzeby działania centralnego daemona.
    b. Tworzenie obrazów przy użyciu narzędzia Podman
        i. Podobnie jak Docker, Podman pozwala na budowanie obrazów kontenerów z użyciem Dockerfile. Proces obejmuje:
            1. Definiowanie warstw obrazu w Dockerfile.
            2. Użycie narzędzia podman build do stworzenia obrazu.
            3. Weryfikację stworzonego obrazu za pomocą podman images.
    c. Tworzenie obrazów przy użyciu podman build
        i. Składnia polecenia: podman build -t nazwa-obrazu .
            1. Opcje dodatkowe:
                a. --file – określenie alternatywnego pliku Dockerfile.
                b. --layers – wykorzystanie istniejących warstw w celu przyspieszenia procesu budowy.
    d. Zarządzanie obrazami: push, pull i tagowanie
        i. Pobieranie obrazów: podman pull obraz
        ii. Wysyłanie obrazów do rejestru: podman push obraz registry.example.com/użytkownik/obraz
        iii. Tagowanie obrazów: podman tag obraz nowa-nazwa
    e. Praktyczne zastosowanie
        i. Podman może być stosowany w różnych scenariuszach, takich jak:
            1. Uruchamianie aplikacji w izolowanych kontenerach.
            2. Tworzenie lekkich środowisk testowych.
            3. Automatyzacja procesów CI/CD.
            4. Użycie w systemach z podwyższonymi wymaganiami bezpieczeństwa.
    f. Uruchamianie kontenerów za pomocą Podman
        i. Uruchomienie prostego kontenera: podman run -d --name kontener-example obraz

   ii. Uruchomienie kontenera z interaktywnym dostępem: podman run -it --rm obraz bash
   iii. Uruchomienie kontenera z mapowaniem portów: podman run -d -p 8080:80 --name kontener-web serwer-www
   iv. Uruchomienie kontenera z woluminem do przechowywania danych: podman run -d -v /ścieżka-lokalna:/ścieżka-kontenerowa --name kontener-z-danymi obraz
   v. Uruchomienie kontenera dla konkretnego adresu IP: podman run -d --ip 192.168.1.100 -p 8080:80 --name kontener-ip obraz
   vi. Wyświetlenie działających kontenerów: podman ps
   vii. Wyświetlanie wszystkich kontenerów (w tym zatrzymanych): podman ps -a
   viii. Zatrzymanie kontenera: podman stop kontener-example
   ix. Usunięcie kontenera: podman rm kontener-example
 g. Przekształcenie workflow Docker do Podman
   i. Większość poleceń Dockera ma swoje odpowiedniki w Podmanie.
   ii. Można łatwo zastąpić docker przez podman w skryptach.
   iii. Podman posiada aliasy umożliwiające użytkownikom przyzwyczajonym do Dockera łatwe przejście: alias docker=podman
   iv. Narzędzie podman system migrate pomaga w migracji istniejących konfiguracji Dockera do Podmana.

2. Instalacja klastra Kubernetes:
 a. W celu instalacji klastra Kubernetes wykorzystamy narzędzie kubeadm.
 b. Minimalne wymagania dotyczące node'a klastra to co najmniej dwa node'y o następujących parametrach:
   OS: Ubuntu
   RAM: 2GB lub więcej
   CPU: 2 CPU lub więcej dla node'a control-plane
   Połączenie sieciowe pomiędzy nodami
 c. Przed zainstalowanie klastra niezbędne jest przygotowania runtime kontenerów oraz narzędzie kubetools: kubeadm, kubeinit, kube-proxy oraz kubectl.
 d. Runtime dla kontenerów jest niezbędny aby móc korzystać z kontenerów
 e. Kubernetes wspiera różnorodne środowiska runtime dla kontenerów:
   containerd
   CRI-O
   Docker Engine
   Mirantis Container Runtime
 f. Poniżej znajduje się link do pobrania repozytorium, z którego pobrać można skrypt instalacyjny środowiska runtime na Wasze stacje robocze:

   https://github.com/lp-lab-gh/cri.git

setup-container.sh

g. Przed instalacją klastra skorzystamy ze skryptu instalacyjnego dla narzędzi Kubernetes, który zawiera
   kubeadm: wykorzystamy go do instalacji i zarządzania naszym klastrem Kubernetes
   kubelet: core service, który uruchamia pody
   kubectl: klient/interfejs, którym będziemy wykonywać polecenia do uruchamiania i zarządzania aplikacjami Kubernetetes
   https://github.com/lp-lab-gh/cri.git

setup-kubetools.sh

h.

- Przed skonfigurowaniem klastra warto więc dowiedzieć się nieco o wymaganiach sieciowych dla różnych węzłów.
W Kubernetes wykorzystywane są różne rodzaje komunikacji sieciowej.

Pierwszym z nich jest komunikacja między węzłami.

Jest to część obsługiwana przez sieć fizyczną, w której wszystkie węzły są ze sobą bezpośrednio połączone.

Istnieje komunikacja External-to-Service, która jest obsługiwana przez zasoby Kubernetes Service.

Istnieje komunikacja Pod-to-Service, która również jest obsługiwana przez usługi Kubernetes.

Komunikacja Pod-to-Pod jest obsługiwana przez wtyczkę sieciową.

I wreszcie, istnieje komunikacja kontener-kontener, która jest stosowana tylko wtedy, gdy wiele kontenerów działa w tym samym Pod.

Komunikacja między tymi kontenerami jest obsługiwana w ramach samego Pod.

Ważną częścią konfiguracji klastra Kubernetes jest dodatek sieciowy, ponieważ bez dodatku sieciowego nie można zdefiniować sieci Pod.

Obecnie dodatek sieciowy jest dostarczany przez ekosystem i dostępne są różne dodatki sieciowe.

Problematyczne jest to, że sam Vanilla Kubernetes nie jest dostarczany z domyślnym dodatkiem, a to dlatego, że Cloud Native Computing Foundation nie chce faworyzować jednego konkretnego rozwiązania

Oznacza to, że po uruchomieniu kubeadm należy samodzielnie skonfigurować dodatek sieciowy.

Kubernetes udostępnia Container Network Interface, czyli CNI, jako ogólny interfejs, który umożliwia łatwe korzystanie z różnych wtyczek.

Dostępność konkretnych funkcji zależy od używanej wtyczki sieciowej.

Należy na przykład szukać obsługi Networkpolicy, IPv6 lub Role Based Access Control.

Dostępnych jest wiele dodatków sieciowych. Chcę tylko wspomnieć o czterech najważniejszych.

Cilium jest tym, którego będziemy używać w tym szkoleniu.

Najpopularniejszym rozwiązaniem jest Calico.

Istnieje Flannel, który jest ogólnym dodatkiem sieciowym, który był często używany w przeszłości, ale nie obsługuje polityki sieciowej.

Istnieje Multus, który jest wtyczką, która może współpracować z wieloma wtyczkami sieciowymi i jest obecnie domyślną wtyczką w OpenShift, która jest bardzo ważną dystrybucją Kubernetes.

Jest też Weave, wspólny dodatek sieciowy, który obsługuje również typowe funkcje.

i.

W dalszej części tej szkolenia pokażę, jak zainstalować klaster przy użyciu kubeadm.

Zanim to zrobimy, dobrze jest wiedzieć trochę o tym, co się wydarzy.

Tak więc, podczas uruchamiania kubeadm init, wykonywane są różne fazy.

Zaczyna się od preflight.

Zapewnia to spełnienie wszystkich warunków i pobranie podstawowych obrazów kontenerów.

Następnie zajmuje się certyfikatami, generowany jest samopodpisany Kubernetes Certificate Authority i tworzone są powiązane certyfikaty dla apiserver, etcd i proxy.

Kubernetes często korzysta z tych certyfikatów PKI i są one bardzo ważne. Następnie znajduje się kubeconfig, w którym konfigurowane są pliki konfiguracyjne dla podstawowych usług Kubernetes.

Następnie można uruchomić kubelet.

Gdy kubelet jest już dostępny, konfigurowany jest control-plane.

W tym celu kubelet używa statycznych manifestów pod. Tworzy je i uruchamia dla apiservera, menedżera kontrolera i schedulera.

W tym momencie używany jest etcd. Tworzone i uruchamiane są więc statyczne manifesty Pod dla etcd.

Następnie procedura jest kontynuowana wraz z przesyłaniem konfiguracji. ConfigMaps są tworzone dla ClusterConfiguration i konfiguracji komponentów kubelet.

Upload-certs jest miejscem, w którym wszystkie certyfikaty są przesyłane do /etc/kubernetes/pki na węźle control-plane.

Następnie węzeł jest oznaczany jako control plane i generowany jest token bootstrap.

Jest to token, którego można użyć do dołączenia do innych węzłów.

Następnym krokiem jest kubelet-finalized, który finalizuje ustawienia kubeleta.

Gdy to zrobimy, pozostaje już tylko jeden krok do wykonania, a jest nim dodatek. W tej części instalowane są dodatki coredns i kube-proxy.

Po wykonaniu wszystkich tych czynności mamy już działające środowisko klastra Kubernetes.

Teraz ważne jest, abyście wiedzieli trochę o tych różnych fazach, ponieważ w przypadku, gdy coś pójdzie nie tak, warto wiedzieć, gdzie coś poszło źle i jak wpłynie to na wszystko inne, co może być przydatne do rozwiązywania problemów.

j. Instalacja klastra - zestaw skryptów:

Sklonowanie repozytorium ze skryptami:

git clone https://github.com/lp-lab-gh/cri/git

Instalacja CRI:

sudo ~/cri/setup-container.sh

Instalacja kubetools:

sudo ~/cri/setup-kubetools.sh

Modyfikacja parametrów w kubeadm-config.yaml:

vim ~/cri/kubeadm-config.yaml

```
apiVersion: kubeadm.k8s.io/v1beta3
kind: ClusterConfiguration
kubernetesVersion: stable
apiServer:
  certSANs:
  - "k8s-X-master-1.k8s.lp-lab.cloud"
controlPlaneEndpoint: "k8s-X-master-1.k8s.lp-lab.cloud:6443"
```

Instalacja klastra:

sudo kubeadm init --config=~/cri/kubeadm-config.yaml

Przygotowanie klienta “kubectl”:

- **mkdir ~/.kube**
- **sudo cp -i /etc/kubernetes/admin.conf ~/.kube/config**
- **sudo chown $(id -u):$(id -g) .kube/config**

k. Instalacja wtyczki sieciowej Cilium:
   i. **sudo snap install helm --classic**
   ii. **kubectl -n kube-system delete ds kube-proxy**
   iii. **kubectl -n kube-system delete cm kube-proxy**
   iv. **helm repo add cilium https://helm.cilium.io/**
   v. **helm upgrade --install cilium cilium/cilium --version 1.16.4 --namespace kube-system --set kubeProxyReplacement=true --set k8sServiceHost=k8s-X-master-1.k8s.lp-lab.cloud --set k8sServicePort=6443 --set hubble.relay.enabled=true --set hubble.ui.enabled=true**

l. Operacje z **kubeadm init**:
   i. --apiserver-advertise-address
   ii. --config
   iii. --dry-run
   iv. --pod-network-cidr
   v. --service-cidr

m. Dodawanie nodeów do klastra:
   i. **sudo kubeadm join**
   ii. W przypadku utraty join token’u lub jego wygaśnięcia używamy polecenia:
   sudo kubeadm token create --print-join-command
   iii. Na nodach należy uruchomić polecenie:
      1. **sudo kubeadm join k8s-X-master-1.k8s.lp-lab.cloud:6443 --token YYY -- discovery-token-ca-cert-hash sha256:ZZZ**

n. Zarządzanie kontekstami:

i. **kubectl get config**
ii. **kubectl set-context**
iii. **kubectl use-context**

3. Używanie deployment'ów:
   a. Deployment to podstawowa opcja uruchamiania kontenerów na Kubernetes
   b. Deployment jest odpowiedzialny za uruchamianie i skalowanie podów
   c. Deployment używa ReplicaSet aby zarządzać skalowalnością
   d. Deployment oferuje politykę wdrażania RollingUpdate, aby uzyskać bezprzerwową funkcjonalność aktualizacji aplikacji.
   e. Uruchomienie deploymentu za pomocą tzw. podejścia imperatywnego:
      i. **kubectl create deployment -h**
      ii. **kubectl create deployment test-nginx --image nginx --replicas=3**
      iii. **kubectl get po -w**
      iv. **kubectl get all**
4. Używanie DaemonSet:
   a. DaemonSets są powszechnie używane do uruchamiania agentów.
   Jest to zasób, który uruchamia jedną instancję aplikacji na każdym węźle klastra. Z tego powodu nie ma liczby replik zdefiniowanych w DaemonSet. Po prostu uruchamia jednego agenta na każdym węźle klastra.
   b. Sprawdźmy czy mamy jakieś obiekty tego typu w klastrze:
      i. **kubectl get ds -A**
   c. Powoływanie obiektu DaemonSet dla obrazu nginx:
      i. **kubectl create deploy mydaemon --image=nginx --dry-run=client -o yaml > mydaemon.yaml**
      ii. Podnosimy do edycji:
      vi mydaemon.yaml
      zmieniamy:
      Deployment > DaemonSet
      i usuwamy:
      - replicas:1
      - strategy: {}
      iii. **kubectl apply -f mydaemon.yaml**
      iv. **kubectl get ds -n default**
5. Używanie StatefulSet:
   a. StatefulSet oferuje kilka funkcji, które są potrzebne w aplikacjach stanowych.
   - Zapewnia gwarancje dotyczące kolejności i unikalności podów
   - Utrzymuje stały identyfikator dla każdego z podów, które tworzy, a pody w StatefulSet nie są wymienne.
   - Każdy Pod ma trwały identyfikator, który utrzymuje podczas zmiany harmonogramu.
   - Unikalne identyfikatory podów ułatwiają dopasowanie istniejących woluminów do zastąpionych podów.
   b. Spójrzmy na przykładowy plik **statefuldemo.yaml**:
   cat statefuldemo.yaml
   kubectl apply -f statefuldemo.yaml

kubectl get statefulset lub **kubectl get sts**

c. Rozwiązanie problemu ze storage

6. Uruchamianie pojedynczych podów:
   a. Główne ryzyka:
      i. Brak ochrony zrównoważonego obciążenia
      ii. Brak LoadBalancing'u
      iii. Brak bezprzerwowej aktualizacji aplikacji
   b. Stosujemy pojedyncze pody wyłącznie w celu testowania, rozwiązywania problemów lub analizy aplikacji/klastra.
   c. W każdym innym razie stosujemy obiekty tj. Deployment, DaemonSet lub Statefulset
   d. Uruchomienie przykładowego pojedynczego pod'a:
      i. kubectl run pod -h | less
      ii. kubectl run pod-with-sleep --image=busybox -- sleep 3600
      iii. kubectl get po
7. Zarządzanie inicjalizacją podów:
   a. Jeśli więc wymagane jest przygotowanie przed uruchomieniem głównego kontenera, odpowiedzią jest init container. Init container'y są częścią pod'a i są uruchamiane razem z głównym kontenerem. Init container działa do końca, a po jego zakończeniu można uruchomić główny kontener. Init container'y są używane w każdym przypadku, w którym wymagana jest wstępna konfiguracja aplikacji.
   b. Przykładowy plik:
      i. kubectl apply -f initme.yaml
      ii. Modyfikacja parametrów na sleep 30

```
apiVersion: v1
kind: Pod
metadata:
  name: init-demo-sleep
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
  initContainers:
  - name: install
    image: busybox
    command:
    - "sleep"
    - "30"
```

      iii. kubectl apply -f initmesleep.yaml
8. Skalowanie aplikacji:
   a. Używamy polecenia **kubectl scale** w stosunku do obiektów tj. Deployment, ReplicaSet lub StatefulSet:

   i. **kubectl scale deployment myapp --replicas=3**
  b. Użycie komendy **kubectl get all** z parametrem **--selector**
   i. **kubectl get all --selector app=test-nginx**
  c. Alternatywą jest użycie obiektu: HorizontalPodAutoscaler.
9. Używanie Sidecar Containers:
  a. W niektórych przypadkach potrzebny jest jednak dodatkowy kontener do modyfikacji lub prezentacji danych wygenerowanych przez główny kontener. Zdefiniowano też pewne szczególne przypadki użycia, którymi są:
   i. sidecar, który zapewnia dodatkową funkcjonalność głównemu kontenerowi
   ii. ambassador, który jest używany jako proxy do zewnętrznego łączenia kontenerów
   iii. adapter, który służy do standaryzacji lub normalizacji danych wyjściowych głównego kontenera.
  b. Przykład w oparciu o plik **sidecarlog.yaml**
   i. **kubectl apply -f sidecarlog.yaml**
   ii. **kubectl exec -it two-containers -c nginx-container -- cat /usr/share/nginx/html/index.html**
10. Zarządzanie storage:
  a. Trwałe woluminy są specyficzne dla każdego z namespace'ów, więc to administrator namespace'a musi upewnić się, że trwałe woluminy są dostępne.
  b. W jaki sposób zamierzasz połączyć się z poda do trwałego woluminu? Zamierzamy to zrobić za pomocą PVC, co jest skrótem od persistent volume claim.To żądanie trwałego wolumenu jest niezależnym zasobem API, który zostanie skonfigurowany jako wolumen pod'a. Wystarczy więc ustawić typ woluminu pod na persistence volume claim.
  c. Żądanie trwałego wolumenu to po prostu żądanie pamięci masowej. Żądanie to jest wyrażone jako rozmiar i to, czy pamięć masowa powinna być zapisywalna (tzw. pamięć RWX) czy tylko do odczytu (tzw. pamięć RWO). Jak to działa? Działa to w ten sposób, że uruchamiasz swój pod i PVC w określonym namespace, a następnie PVC sprawdzi w tym namespace, czy dostępna jest określona pamięć masowa. Jeśli istnieje pasujący trwały wolumen, to jest to proste, a PVC zostanie powiązane z pasującym PV. Istnieje jednak opcja elastyczności, która jest realizowana za pomocą StorageClass. StorageClass jest używany do automatycznego dostarczania pamięci masowej, jeśli nie jest ona dostępna w momencie nadejścia żądania PVC. Aby móc pracować z obiektami StorageClass, potrzebny jest również odpowiedni provider dla trwałego woluminu (PV).
  d. Czym jest PV Provisione? Dostawca pamięci masowej komunikuje się z pamięcią masową dostępną w określonym namespace.
  e. StorageClass to obiekt API reprezentujący Provisioner'a pamkięci. Tak więc bez względu na to, jaką masz pamięć masową, tak długo, jak masz provisioner'a i StorageClass która z nim współpracuje, StorageClass może przekazywać pamięć masową na żądanie PVC.

f. Tak więc w przypadku, gdy PVC zgłasza żądanie pamięci masowej i nie ma dostępnej pamięci masowej, klasa pamięci masowej przechwyci to żądanie i utworzy trwały wolumin (PV) na żądanie zgodnie z wymaganiami określonymi w PVC.Dzięki temu Kubernetes może dynamicznie przydzielać przestrzeń dyskową, gdy jest ona potrzebna.
g. Konfiguracja PersistenceVolumes:
    i. Przykład:
        1. **vi pv.yaml**
        2. **kubectl apply -f pv.yaml**
h. Konfiguracja PersistenceVolumeClaim:
    i. Przykład:
        1. **vi pvc.yaml**
        2. **kubectl apply -f pvc.yaml**
        3. Troubleshooting
i. Konfiguracja StorageClass:
    i. Instalacja serwera NFS na node'dzie - **"k8s-X-master-1"**:
        1. na nodzie control-plane: **sudo apt install nfs-server -y**
        2. **sudo mkdir /nfsexport**
        3. **sudo sh -c 'echo "/nfsexport *(rw,no_root_squash)" > /etc/exports'**
        4. **sudo systemctl restart nfs-server**
    ii. Instalacja klienta NFS na node'ach - **"k8s-X-worker-1"** i **"k8s-X-worker-2"**:
        1. **sudo apt install nfs-client**
        2. **showmount -e k8s-X-master-1**
    iii. Instalacja dodatku nfs-subdir-external-provisioner na węźle "**k8s-X-master-1**":
        1. **helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner**
        2. **helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner --set nfs.server=192.168.0.X --set nfs.path=/nfsexport**
    iv. Zarządzanie PVC:
        1. **kubectl get pv**
        2. **kubectl apply -f nfs-provisioner-pvc-test.yaml**
        3. **kubectl get pv,pvc**
    v. Ćwiczenia z PVC i SC:
        1. **kubectl apply -f another-pvc-test.ymal**
        2. **kubectl get pvc**
        3. **kubectl patch storageclass nfs-client -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'**
        4. **kubectl get pvc**
j. Konfiguracja ConfigMap i Secret'ów

i. ConfigMap jest zasobem API używanym do przechowywania danych specyficznych dla witryny. Secret to ConfigMap zakodowany w base64.
ii. Nie ma żadnej fundamentalnej różnicy między Secret i ConfigMap, z jedynym wyjątkiem, że Secret są trudniejsze do odczytania, ponieważ najpierw trzeba je zdekodować. ConfigMaps są używane do przechowywania zmiennych środowiskowych, parametrów startowych lub plików konfiguracyjnych. Gdy plik konfiguracyjny jest używany w ConfigMap lub Secret, jest montowany jako wolumin, aby zapewnić dostęp do jego zawartości.
    1. Przykład w oparciu o **webserver.yaml** z repozytorium.

11. Zarządzanie dostęp do aplikacji (CNI, Services):
    a. Omówienie CNI:
        i. Calico
        ii. Flannel
        iii. Cilium
    b. Omówienie usług (services):
        i. NodePort
        ii. ClusterIP
        iii. LoadBalancer
    c. Konfiguracja Ingress:
        i. Instalacja:
            1. **helm upgrade --install ingress-nginx ingress-nginx --repo https://kubernetes.github.io/ingress-nginx --namespace ingress-nginx --create-namespace**
            2. **kubecti get pods -n ingress-nginx**
            3. **kubectl create deploy nginxsvc --image=nginx --port=80**
            4. **kubecti expose deploy nginxsvc**
        ii. Wsparcie dla ruchu HTTP i HTTPS
            1. **kubectl create ingress nginxsvc --class=nginx --rule=nginxsvc.info/*=nginxsvc:80**
            2. **kubectl port-forward -n ingress-nginx svc/ingress-nginx-controller 8080:80**
            3. **echo "127.0.0.1 nginxsvc.info" >> /etc/hosts**
            4. **curl nginxsvc.info:8080**
        iii. Konfiguracja i omówienie IngressClass
            1. **kubectl get ingressclass -o yaml**
        iv. Przykłady:
            1. Przykład z webshop-apps:
                a. **kubectl get deployment kubectl get svc webshop**
                b. **kubectl create ingress webshop-ingress --rule="/=webshop:80" --rule="/hello=newdep:8080"**
                c. **sudo vim /etc/hosts**
                    i. *127.0.0.1 webshop.info*
                d. **kubectl get ingress**
                e. **kubectl describe ingress webshop-ingress**

2. Ingress Rule dla różnych ścieżek:
    a. **kubectl create ingress mygress --rule="/mygress=mygress:80" --rule="/yourgress=yourgress:80"**
3. Różne virtual host'y:
    a. **kubectl create ingress nginxsvc--class=nginx --rule=nginxsvc.info/*=nginxsvc:80 --rule=otherserver.org/*=otherserver:80**

d. Używanie port forwarding'u do bezpośredniego dostępu do aplikacji

12. Zarządzanie klastrem:
    a. Analiza node'ów klastra
    b. Używanie **crictl** do zarządzania i analizowania kontenerów na węźle
    c. Uruchomienie tzw. static pod
        i. **ssh k8svc@k8s-X-worker1**
        ii. **sudo vi /etc/kubernetes/manifests/staticpod.yaml**
    d. Zarządzanie stanem node'ów
        i. Drain
            1. **kubectl drain --ignore-daemonsets --delete-emptydir-data**
        ii. Cordon
            1. **kubectl cordon <node>**
            2. **kubectl describe node <node>**
            3. **kubectl get nodes**
        iii. Uncordon
            1. **kubectl uncordon <node>**
    e. Zarządzanie usługami node'a
        i. **ps aux | grep kubelet**
        ii. **ps aux | grep containerd**
        iii. **systemctl status kubelet**
        iv. **sudo systemctl stop kubelet**
        v. **sudo systemctl start kubelet**
13. Przeprowadzanie zadań konserwacyjnych na node'ach
    a. Instalacja i przegląd funkcji metrics-server
        i. dokumentacja: **https://github.com/kubernetes-sigs/metrics-server.git**
        ii. **kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml**
        iii. **kubectl -n kube-system get pods # look for metrics-server**
        iv. **kubectl -n kube-system edit deployment metrics-server**
            1. W spec.template.spec.containers.args, zmień na następujące wartości:
                a. **--kubelet-insecure-tls**
                b. **--kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname**
        v. **kubectl -n kube-system logs metrics-server<TAB>** powinna pokazać "Generating self-signed cert" i "Serving securely on [::]443

vi. **kubectl top pods --all-namespaces** zwróci najbardziej aktywne pody

b. Backup bazy danych etcd
   i. **sudo apt install etcd-client**
   ii. **sudo etcdctl --help; sudo ETCDCTL_API=3 etcdctl --help**
   iii. **ps aux | grep etcd**
   iv. **sudo ETCDCTL_API=3 etcdctl –endpoints=localhost:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key get / --prefix --keys-only**
   v. **sudo ETCDCTL_API=3 etcdctl –endpoints=localhost:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key snapshot save /tmp/etcdbackup.db**

c. Przywracanie kopii bazy danych etcd
   i. **sudo ETCDCTL_API=3 etcdctl snapshot restore /tmp/etcdbackup.db data-dir /var/lib/etcd-backup**
   ii. Usługi core’owe Kubernetes podczas operacji restore’u etcd muszą zostać zatrzymane
   iii. Aby to wykonać należy przesunąć tymczasowo zawartość katalogu /etc/kubernetes/manifests/*.yaml to /etc/kubernetes/
   iv. Proces kubelet co jakiś czas sprawdza zawartość katalogu
   v. Poleceniem **sudo crictl ps** zweryfikować można czy usługi się zatrzymały

d. Przeprowadzanie aktualizacji węzłów
e. Klaster o wysokiej dostępności - opis
f. Klaster o wysokiej dostępności - wdrożenie

14. Zarządzanie scheduler’em
   a. Kube-scheduler bierze na siebie zadania związane z znalezieniem node’a w celu zaschedulowania nowych podów
   b. Node’y mogą być filtrowane wg specyficznych wymagań tj.:
      i. Wymagania co do zasobów sprzętowych
      ii. Reguł affinity i anti-affinity
      iii. Taints i Tolerations
      iv. Scheduler na podstawie wprowadzonych danych wybiera node’y nadając im punktację a następnie wybiera najwyżej ocenionego node’a w celu zaschedulowania pod’a zgodnego z wymaganiami
   c. Kubelet zajmuje się pobranie obrazu i instalacją pod’a na wskazanych przez kube-scheduler’a node’dzie
   d. Ustawianie preferencji node’a
      i. kubectl label node
      ii. nodeSelector
      iii. nodeName
   e. Ustawienie affinity i anti-affinity
      i. **pod-with-node-affinity.yaml**
      ii. **pod-with-node-anti-affinity.yaml**

iii. **pod-with-pod-affinity.yaml**

f. Zarządzanie taints i tolerations
   i. Przykłady:
      1. **taint-tolerations.yaml**
      2. **taint-tolerations2.yaml**

g. Omówienie LimitRange i Quota.
h. Konfiguracja LimitRange:
   i. Przykład:
      1. **limitrange.yaml**

15. Sieć w Kubernetes:
    a. Omówienie sieci CNI i plugin'ów sieciowych
    b. Omówienie service auto registration
       i. Przykłady:
    c. Polityki networkPolicy do zarządzania ruchem pomiędzy podami
    d. Konfiguracja networkPolicy w klastrze
16. Zarządzanie bezpieczeństwem klastra
    a. Omówienie dostępu do API klastra
    b. Zarządzanie security context
    c. Używanie kont serwisowych do dostępu do API klastra
    d. Tworzenie polity RBAC
    e. Konfiguracja ról klastra oraz RoleBindings
    f. Tworzenie konto użytkowników Kubernetes
17. Logowanie, Monitorowanie oraz rozwiązywanie problemów z klastrem
    a. Monitorowanie zasobów Kubernetes
       i. **kubectl top pod**
       ii. **kubectl top nodes**
    b. Zapoznanie się z narzędziami pomocnymi w rozwiązywaniu problemów z klastrem
       i. **kubectl describe**
       ii. **kubectl events**
    c. Rozwiązywanie problemów z aplikacjami
       i. **kubectl get**
       ii. **kubectl logs**
    d. Rozwiązywanie problemów z węzłami klastra Kubernetes:
       i. **sudo systemctl status kubelet**
       ii. **sudo systemctl restart kubelet**
       iii. **sudo openssl x509 -n /var/lib/kubelet/pki/kubelet.crt -text**
       iv. **kubectl get pods -n kube-system**
    e. Rozwiązywanie problemów z dostępem do aplikacji pracujących w ramach klastra:
       i. **kubectl get endpoints**
18. Harbor jako repozytorium obrazów
    a. Dlaczego warto używać Harbor do bezpiecznego przechowywania obrazów?
    b. Instalacja Harbor
    c. Konfiguracja i przesyłanie obrazów do Harbor
19. Monitorowanie i diagnostyka

a. Podstawy monitorowania
b. Prometheus, Grafana i Kubernetes Dashboard
    i. Instalacja i przegląd zdarzeń

20. Integracja wdrażania z GitOps
    a. Przegląd strategii GitOps
    b. Zasady i korzyści GitOps dla CI/CD
    c. Repozytoria
    d. Struktura repozytoriów gotowych na GitOps
    e. Konfigurowanie przepływu pracy GitOps
    f. Praktyczny przykład z wykorzystaniem ArgoCD