Sustav za prikupljanje podataka sa senzora temeljen na platformi K3S

Perko, Marko

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:238285

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Marko Perko

Sustav za prikupljanje podataka sa senzora temeljen na platformi k3s

Diplomski rad

Pula, veljača, 2022. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Marko Perko

Sustav za prikupljanje podataka sa senzora temeljen na platformi k3s

Diplomski rad

JMBAG: Marko Perko, 0303071157

Studijski smjer: Informatika

Predmet: Izrada informatičkih projekata

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: Prof.dr.sc. Nikola Tanković

Pula, veljača, 2022. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani *Marko Perko*, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da nikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, _____, _____ godine

IZJAVA

o korištenju autorskog djela

Ja, *Marko Perko* dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava

iskorištavanja, da moj diplomski rad pod nazivom Prikupljanje podataka sa senzora putem k3s platforme koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____

Potpis _____

Sadržaj

1. Uvod

Kontejnerizacija je oblik virtualizacije operativnog sustava koji korisnicima omogućuje pokretanje aplikacija u potpuno zatvorenim, samodostatnim računalnim okruženjima. Ovi spremnici uključuju sve računalne elemente potrebne za učinkovito pokretanje aplikacija — kod, biblioteke, ovisnosti, podatke o konfiguraciji i tako dalje. podijeljena, fleksibilna i sigurna, kontejnerizacija je prirodna evolucija virtualizacije, koja omogućuje izvođenje aplikacija na gotovo svakom računalu. Nažalost, kada se spremnici organizacije počnu brojiti u stotinama ili tisućama, upravljanje tim samostalnim jedinicama može postati izuzetno skupo i dugotrajno.

Kubernetes (ponekad zvan K8s) je alat za orkestraciju spremnika otvorenog koda koji automatizira implementaciju spremnika, upravljanje, skaliranje i umrežavanje. Kubernetes omogućuje korisnicima da upravljaju spremnicima na više domaćina, automatiziraju implementaciju i ažuriranje aplikacija, kontroliraju i nadziru potrošnju resursa, učinkovito distribuiraju opterećenje aplikacije kroz infrastrukturu i još mnogo toga. A kako bi se osiguralo učinkovito upravljanje kontejnerizacijom, mnoge organizacije ovise o Kubernetes-ima.

2. Usporedba K3s-a i kubernetes-a

Kada govorimo o k3s-u moramo uzeti u obzir da je k3s lagana odnosno manje zahtjevna verzija k8s-a odnosno kubernetesa.

S obzirom da k3s nema službenog imena koristit ćemo kraticu k3s. Zbog toga što je instalacija Kubernetesa koja je upola manja u smislu memorijskog prostora. Kubernetes je riječ od 10 slova stilizirana kao K8s. Dakle, nešto u pola manje od Kubernetesa bila bi riječ od 5 slova stilizirana kao K3s.

Kubernetes, odnosno K8s, je sustav otvorenog koda za automatizaciju implementacije, skaliranja i upravljanja kontejnerskim aplikacijama. K8s Grupira spremnike koji čine aplikaciju u logičke jedinice za jednostavno upravljanje i otkrivanje.

K3s je visoko dostupna, certificirana Kubernetes distribucija dizajnirana za produkcijska okruženja s ograničenim resursima, udaljenim lokacijama ili unutar Internet stvari (IoT) uređaja.

K3s je pakiran kao jedna binarna datoteka <50MB prostora koja smanjuje ovisnosti (eng. dependency) i korake potrebne za instalaciju, pokretanje i automatsko ažuriranje produkcijskog Kubernetes klastera.

I ARM64 i ARMv7 su podržani s binarnim i multiarch slikama(eng. image) dostupnim za obje arhitekture. K3s izvrsno radi od nečega malog kao što je Raspberry Pi do AWS servera velikog 32GiB.

I to ga zapravo čini idealnim za Internet stvari te edge primjenu.

K3s poboljšanja u odnosu na k8s:

- Pakiran je kao jedna binarna datoteka.
- Lagano skladištenje koje se temelji na sqlite3 kao zadanom mehanizmu za pohranu. etcd3, MySQL, Postgres također su dostupni.
- Umotan u jednostavan pokretač koji rješava mnogo složenosti TLS-a i opcija.

- Dodane su jednostavne, ali moćne značajke, kao što su: lokalni dobavljač pohrane(eng. local storage provider), balansiranje opterećenja usluge, Helm kontroler i Traefik ulazni(eng. ingress) kontroler.
- Rad svih komponenti Kubernetes kontrolne ravnine sadržan je u jednom binarnom i procesu. To omogućuje K3 automatiziranje i upravljanje složenim operacijama klastera kao što je distribucija certifikata.
- Vanjske ovisnosti su minimizirane .
- K3s pakira potrebne ovisnosti, uključujući:
  containerd, Flanel, CoreDNS, CNI, Host utilities (iptables, socat, itd.), Ingress controller (traefik), Ugrađeni servis za balansiranje opterećenja, Ugrađeni kontroler mrežne politike

3. Priprema sustava

Za ovaj klaster koristit ćemo tri Raspberry Pi-a. Prvog ćemo nazvati master i dodijeliti statičku IP adresu 192.168.0.50. Prvi radni čvor (eng. Worker node) dodjelit ćemo ime node1 i dodijeliti IP 192.168.0.51. Posljednji radni čvor nazvat ćemo imenom node2 i dodijeliti IP 192.168.0.52.

Kako se ne bismo morali stalno pozivati svaki čvor po IP-u, dodajmo njihova imena hostova u našu /etc/hosts datoteku na našem računalu.

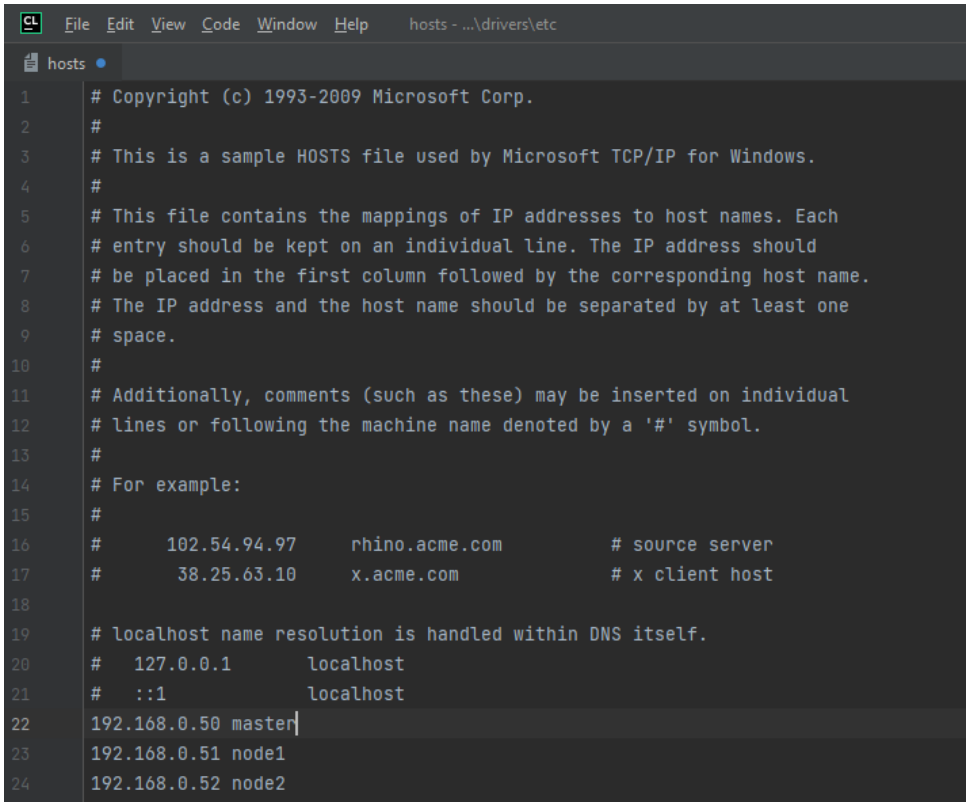
Na linux sustavima to možemo postići sljedećim naredbama:

echo -e "192.168.0.50\master" | *sudo tee -a /etc/hosts*

echo -e "192.168.0.51\node1" | *sudo tee -a /etc/hosts*

echo -e "192.168.0.52\node2" | *sudo tee -a /etc/hosts*

Na windows sustavima moramo ručno otvoriti hosts datoteku te je urediti. Datoteka bi se zadano trebala nalaziti na putu: *C:\Windows\System32\drivers\etc*

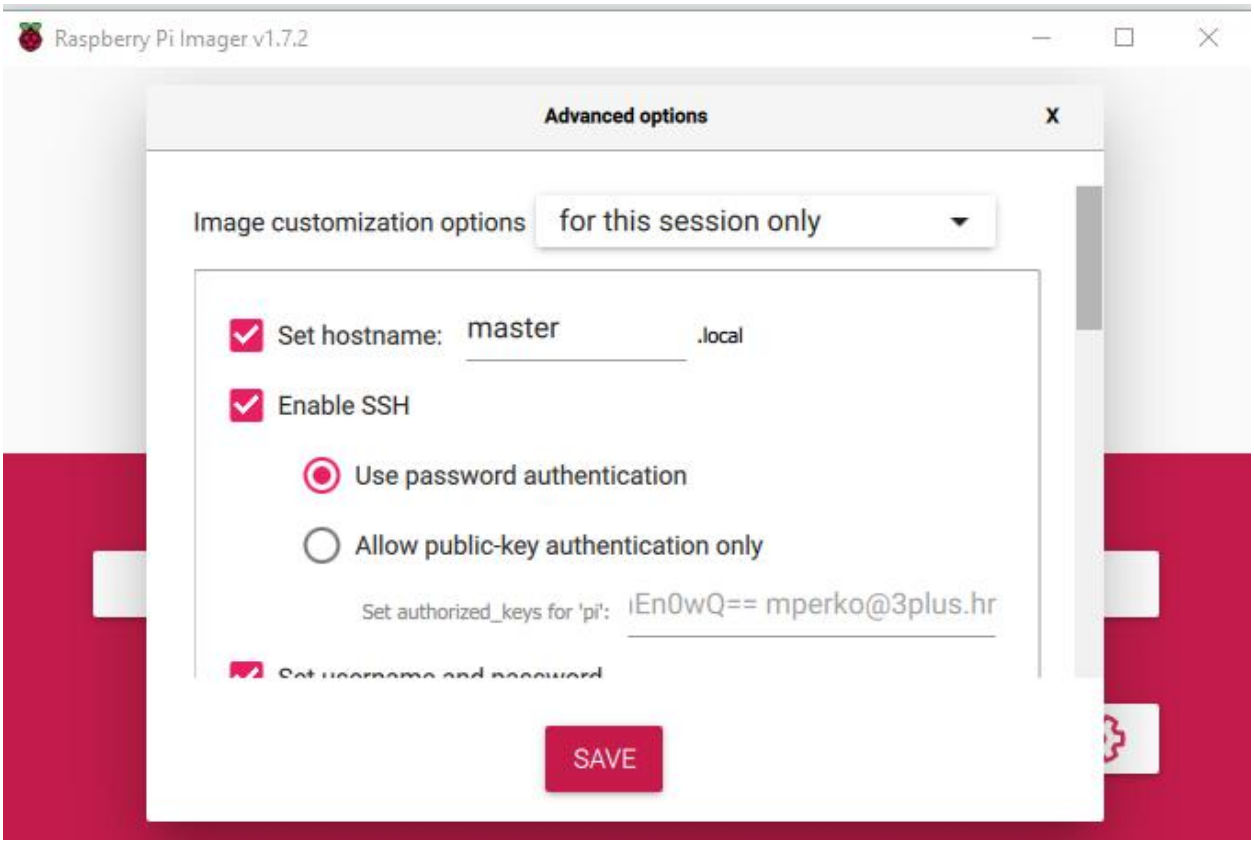


Slika1: konfiguracija domaćina

Izvor: vlastita izrada

Na raspberry pi-ovima ćemo koristit Raspberry Pi OS (bivši Raspbian) arm64. Vrlo je bitno da je arhitektura 64 bitna, iako k3s može raditi na 32 bitnim arhitekturama. Neki od alata, servisa itd. Koje ćemo koristiti nisu podržani za 32 bitne sustave.

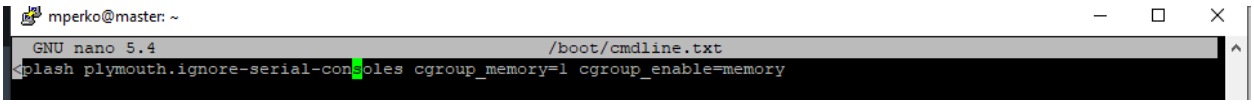
Također prije stavljanja slike diska, trebamo omogućit ssh te je poželjno da odmah postavimo ime domaćina (eng. hostname).



Slika2: prikaz postavki rpi imager-a

Izvor: osobna izrada

Standardne verzije rpi sustava zadano imaju *cgroup* isključen. Te ga trebamo uključiti K3S treba cgroups za pokretanje systemd servisa. To možemo napraviti nakon boota tako da uredimo datoteku */boot/cmdline.txt* i dodajući *cgroup_memory=1 cgroup_enable=memory* na kraj reda.



Slika3: prikaz datoteke cmdline.txt

Izvor: vlastita izrada

Raspberry Pi OS (bivši Raspbian) prema zadanim postavkama koristi nftables umjesto iptables. K3S mrežne značajke zahtijevaju iptables i ne rade s nftables. Stoga trebamo podesiti konfiguraciju da koristimo iptables.

sudo iptables -F

sudo update-alternatives --set iptables /usr/sbin/iptables-legacy

sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy

```
mperko@master:~ $ sudo iptables -F
sudo update-alternatives --set iptables /usr/sbin/iptables-legacy
sudo update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy
update-alternatives: using /usr/sbin/iptables-legacy to provide /usr/sbin/iptables (iptables) in manual mode
update-alternatives: using /usr/sbin/ip6tables-legacy to provide /usr/sbin/ip6tables (ip6tables) in manual mode
mperko@master:~ $
```

Slika4: poziv naredbe i ispis

Izvor: vlastita izrada

sudo reboot

Te je potrebno ponovno pokretanje sustava kako bi se konfiguracija postavila.

K3s pruža instalacijsku skriptu koja omogućuje jednostavnu instalaciju na sustavima baziranim na systemd ili openrc (uz određene preinake moguće i na init.d).

curl -sfL https://get.k3s.io | sh –

```
mperko@master: ~
mperko@master:~ $ curl -sfL https://get.k3s.io | sh -
[INFO]  Finding release for channel stable
[INFO]  Using v1.24.3+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.24.3+k3s1/sha256sum-arm64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.24.3+k3s1/k3s-arm64
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Creating /usr/local/bin/ctr symlink to k3s
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
mperko@master:~ $
```

Slika5: instalacija master čvora

Izvor: vlastita izrada

Skripta će se pobrinuti da :

- K3s bit će konfiguriran za automatsko ponovno pokretanje nakon ponovnog pokretanja čvora ili ako se proces sruši ili prekine
- Instalirat će se dodatni uslužni programi, uključujući kubectl, crictl, ctr te bash skripte k3s-killall.sh i k3s-uninstall.sh
- Datoteka kubeconfig bit će zapisana u /etc/rancher/k3s/k3s.yaml i kubectl instaliran od strane K3s-a automatski će je koristiti

Kada naredba završi, već imamo postavljen i pokrenut klaster s jednim čvorom. Provjerimo to sa

sudo kubectl get nodes

```
mperko@master:~ $ sudo kubectl get nodes
NAME     STATUS   ROLES                  AGE   VERSION
master   Ready    control-plane,master   23m   v1.24.3+k3s1
```

Slika6: prikaz master čvora

Izvor: vlastita izrada

Želimo dodati nekoliko radnih čvorova. Kada instaliramo k3s na te čvorove trebat će nam token za pridruživanje. Token spajanja postoji na datotečnom sustavu glavnog čvora. Kopirajmo to i spremimo negdje. Token si možemo ispisati pomoću

sudo cat /var/lib/rancher/k3s/server/node-token

```
mperko@master:~ $ sudo cat /var/lib/rancher/k3s/server/node-token
K109b04493501c5217c1806357eda0259f46ffc2eb5bba4058c4558422eac07aa0d::server:eef1b51221169dc143d2b7d27b969388
```

Slika7: ispis tokena

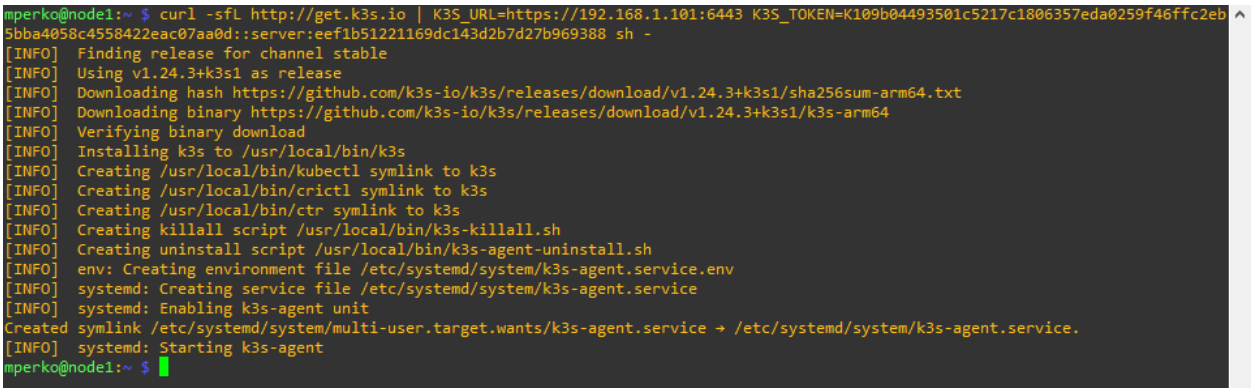
Izvor: vlastita izrada

Na Pi ćemo instalirati k3s kao i prije, ali ćemo instalaciji dati dodatne parametre kako bismo ga obavijestili da instaliramo radni čvor i da se želimo pridružiti postojećem klasteru.

*curl -sfL http://get.k3s.io | K3S_URL=https://192.168.0.50:6443 *

K3S_TOKEN=token_za_pridruzivanje sh -

Zamijenimo join_token tokenom pridruživanja.



Slika8: instalacija radnog čvora

Izvor: vlastita izrada

Ponovimo tako i za drugi radni čvor.

Postavljanje parametra K3S_URL uzrokuje pokretanje K3s-a u radnom načinu. K3s agent će se registrirati na K3s poslužitelju koji sluša na dostavljenom URL-u. Vrijednost za K3S_TOKEN pohranjena je na /var/lib/rancher/k3s/server/node-token na poslužiteljskom čvoru.

Napomena: Svaka mašina mora imati jedinstven ime domaćina. Ako mašine nemaju jedinstveno ime domaćina, proslijedi se varijabla okruženja K3S_NODE_NAME i vrijednost s važećim i jedinstvenim imenom domaćina za svaki čvor.

3.1. Pripema lokalnog kubectla

Bilo bi nepraktično morati se ssh-at do glavnog čvora za pokretanje kubectl kad god želimo pregledati ili modificirati naš klaster. Također će nam biti otežan pristup kontrolnoj ploči (eng. dashboard), no o tome više u poglavlju kontrolna ploča. Dakle, želimo staviti kubectl na naše računalo. Ali prvo, uzmimo informacije o konfiguraciji koje su nam potrebne od našeg glavnog čvora. Stoga se trebamo ssh-at na master čvor te dobit informacije o konfiguraciji, što možemo učiniti sljedećom naredbom

sudo cat /etc/rancher/k3s/k3s.yaml

```
mperko@master:~ $ sudo cat /etc/rancher/k3s/k3s.yaml
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJlRENDQVIyZ0F3SUJBZ0lCQURBS0JnZ3Foa2pPUFFRREFqQWpNU0V3SHdZRF
ZRUUREQmhyTTNNdGMyVnkKZG1WeUxXTmhRREUyTlRrMk1qQTNPVGt3SGhjTk1qSXdPREEwTVRNME5qTTVXaGNOTXpJd09EQXhNVE0wTmpNNQpXakFqTVNFd0h3WURWUVFEREJo
ck0zTXRjMlZ5ZG1WeUxXTmhRREUyTlRrMk1qQTNPVGt3V1RBVEJnY3Foa2pPClBRSUJCZ2dxaGtqT1BRTUJCd05DQUFUUVd2azNPZGhvK1N3dmRrWGdEa2RxL1JMejRpVFhzc3
FWL09tb2FtL1YKWElLZ1IvMi9QeGUrVlpXbDRZSU9GeXgzY2Y2V1VsU0ppNGNrRHp5VnY1ZWNvMEl3UURBT0JnTlZIUThCQWY4RQpCQU1DQXFRd0R3WURWUjBUQVFIL0JBVXdB
d0VCL3pBZEJnTlZIUTRFRmdRVXIzZnZBQW9aVGw4KzZyVmdSSHBXCklGNmZyaVl3Q2dZSUtvWkl6ajBFQXdJRFNRQXdSZ0loQUtTbmtPcHh0TnNJNG9iamV3dVA0eWluVUdRWV
FwRnoKMURER2JHeU1zZmdBQWlFQWxvRUc3WXFVRjFFMDZiUHB2c0J2aUZnWitKZ0pMaGR0VkdOSzByN01Dd009Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    server: https://127.0.0.1:6443
  name: default
contexts:
- context:
    cluster: default
    user: default
  name: default
current-context: default
kind: Config
preferences: {}
users:
- name: default
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJrRENDQVRlZ0F3SUJBZ0lJTms5Skl6RXZtSXd3Q2dZSUtvWkl6ajBFQXdJd0l6R
WhNQjhHQTFVRUF3d1kKYXpOekxXTnNhV1Z1ZEMxallVQXhOalU1TmpJd056azVNQjRYRFRJeU1EZ3dOREV6TkRZek9Wb1hEVEl6TURndwpOREV6TkRZek9Wb3dNREVYTUJVR0E
xVUVDaE1PYzNsemRHVnRPbTFoYzNSbGNuTXhGVEFUQmdOVkJBTVRESE41CmMzUmxiVHBoWkcxcGJqQlpNQk1HQnlxR1NNNDlBd0VHQ0NxR1NNNDlBd0VIQTBJQUJPTkQ2QSt2Q
XFLcGNlcHAKdnpmTDlHbk9yTE5GOGwzR0pMTU91WWhLaDg5VWg4amhLM2xMUVdxZlJJaW91bG5va2VUa3NqMXp0ZnF2RjRKegpRLzRKMW5talNEQkdNQTRHQTFVZER3RUIvd1F
FQXdJRm9EQVRCZ05WSFNVRUREQUtCZ2dyQmdFRkJRY0RBakFmCkJnTlZIU01FR0RBV2dCU1F2ZGo5K0ErUlp6L3liRDVUK2dNTEdDdEJCREFLQmdncWhrak9QUVFEQWdOSEFEQ
kUKQWlCWkpxeWtIZXVwRnh4ZzFJS3hKUGd3K1NRa2tHZnJtUDdPUEpqV0M0YWpvQUlnQU1Jb1B5TjVHQTN2L0t4QQp6T0twSi9FR3VRb1huL2RZQWpBSEFlcG43ekk9Ci0tLS0
tRU5EIENFUlRJRklDQVRFLS0tLS0KLS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUJkekNDQVIyZ0F3SUJBZ0lCQURBS0JnZ3Foa2pPUFFRREFqQWpNU0V3SHdZRFZRU
UREQmhyTTNNdFkyeHAKWlc1MExXTmhRREUyTlRrMk1qQTNPVGt3SGhjTk1qSXdPREEwTVRNME5qTTVXaGNOTXpJd09EQXhNVE0wTmpNNQpXakFqTVNFd0h3WURWUVFEREJock0
zTXRZMnhwWlc1MExXTmhRREUyTlRrMk1qQTNPVGt3V1RBVEJnY3Foa2pPClBRSUJCZ2dxaGtqT1BRTUJCd05DQUFRZUZ6T3l3ckxKZU9zbi9kVUY3VWh5RXR2Yk1RZHMrbkUvR
jlnWlNHZzEKbjJobUVCQmNkSEYzZzNGVXR1WUFLdTE5MlZKNDg4WEZBMGhOM0xucFJJdfNvMEl3UURBT0JnTlZIUThCQWY4RQpCQU1DQXFRd0R3WURWUjBUQVFIL0JBVXd8d0V
CL3pBZEJnTlZIUTRFRmdRVWtMM1kvZmdQa1djLzhtdytVL29ECkN4Z3JRUVF3Q2dZSUtvWkl6ajBFQXdJRFNBQXdSUUlnU2F6YWl6SVpiTTNXTUJpYzBOWFZKSzBLdFhQWU5Pe
lYKTlRzZnlya2FZcjRDSVFDMC9ibSs3NVNjc2Jhbzd1RjJDTU9UV0lJanJQVXdRdEhMeEVzUjBtUC9vZz09Ci0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    client-key-data: LS0tLS1CRUdJTiBFQyBQUklWQVRFIEtFWS0tLS0tCk1IY0NBUUVFSVBKcVVyUUpzaWZxWXVpeXJ3Mzl3Zk9xU1Z4dWM5U0ozT2lYZi85L0dQSkdvQ
W9HQ0NxR1NNNDkKQXdFSG9VUURRZ0FFNDBQb0Q2OENvcWx4Nm1tL044djBhYzZzczBYeVhjWWtzdzY1aUVxSHoxU0h5T0VyZVV0QgphcDlFaUtpNldlaVI1T1N5UFhPMStxOFh
nbk5EL2duV2VRPT0KLS0tLS1FTkQgRUMgUFJJVkFURSBLRVktLS0tLQo=
mperko@master:~ $
```

Slika9: konfiguracija kubectla

Izvor: vlastita izrada

Kopirajte informacije o konfiguraciji i vratite se na svoje računalo. Napravite direktorij za konfiguraciju s imenom .kube

Spremite kopiranu konfiguraciju kao /.kube/config. Sada uredite datoteku i promijenite redak

server: https://localhost:6443 u server: https://master:6443

Sada trebamo instalirat kubetcl na naše računalo. Prije nego što počnemo, treba naglasiti da moramo koristiti kubectl verziju koja je unutar jedne male razlike u verziji klastera. Na primjer, v1.24 klijent može komunicirati s upravljačkim ravnima v1.23, v1.24 i v1.25.

Prvo trebamo skinuti kubetcl binarnu datoteku što možemo učiniti pomoću curl komande.

curl -LO "https://dl.k8s.io/release/v1.24.0/bin/windows/amd64/kubectl.exe"

Provjerimo da li je sve u redu s datotekom, tako da preuzmemo kubectl datoteku kontrolnog zbroja (eng. checksum)

curl -LO "https://dl.k8s.io/v1.24.0/bin/windows/amd64/kubectl.exe.sha256"

Potvrdite kubectl binarnu datoteku s datotekom kontrolnog zbroja korištenjem PowerShell-a za automatizaciju provjere pomoću operatora -eq kako bi dobili rezultat True ili False, te ne bismo morali ručno provjeravati.

$($(CertUtil -hashfile .\kubectl.exe SHA256)[1] -replace " ", "") -eq $(type .\kubectl.exe.sha256)

```
PS C:\Users\mperko> $($(CertUtil -hashfile .\kubectl.exe SHA256)[1] -replace " ", "") -eq $(type .\kubectl.exe.sha256)
True
```

Slika10: prikaz rezultata provjere putem kontrolnog zbroja

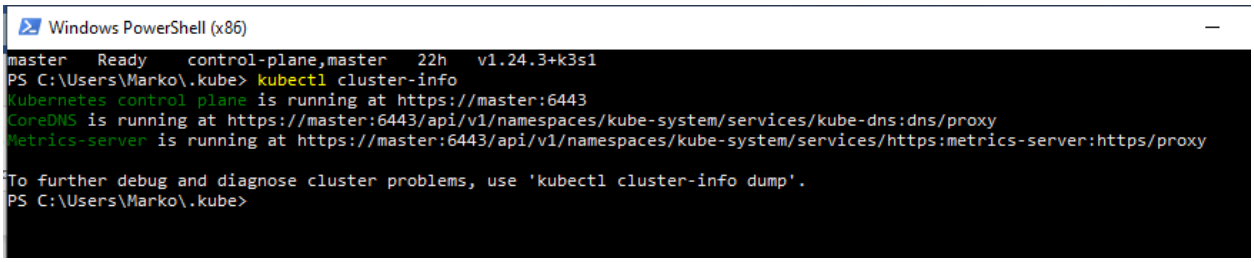
Izvor: vlastita izrada

Napomena: Docker Desktop za Windows dodaje vlastitu verziju kubectl u PATH. Ako ste već instalirali Docker Desktop, možda ćete morati postaviti svoj put ispred onoga koji je dodao instalacijski program Docker Desktop ili ukloniti kubectl Docker Desktop.

Provjerimo kubectl konfiguraciju. Kako bi kubectl pronašao i pristupio Kubernetes klasteru, potrebna mu je kubeconfig datoteka, koja se kreira automatski kada kreirate klaster pomoću kube-up.sh ili uspješno implementirate klaster Minikube. Prema zadanim postavkama, kubectl konfiguracija se nalazi na ~/.kube/config.

Provjerite je li kubectl ispravno konfiguriran dobivanjem stanja klastera:

kubectl cluster-info



Slika11: cluster informacije

Izvor: vlastita izrada

Ako *pokrenete kubectl get pods --all-namespaces*, vidjet ćete neke dodatne podove za Traefik. Traefik je obrnuti posrednik (eng. proxy)i balanser opterećenja (eng. Load balancer) koji možemo koristiti za usmjeravanje prometa u naš klaster s jedne ulazne točke. Kubernetes to dopušta, ali ne pruža takvu uslugu izravno. Imati Traefik instaliran prema zadanim postavkama je dio k3s instalacije . To čini zadanu instalaciju k3s potpuno dovršenom i odmah upotrebljivom.

3.2. Funkcionalni test

Radi provjere, pokrenimo čahuru (eng. pod) koja svake dvije sekunde ispisuje " *Hello World* " u dnevnik koristeći javnu sliku putem sljedeće naredbe:

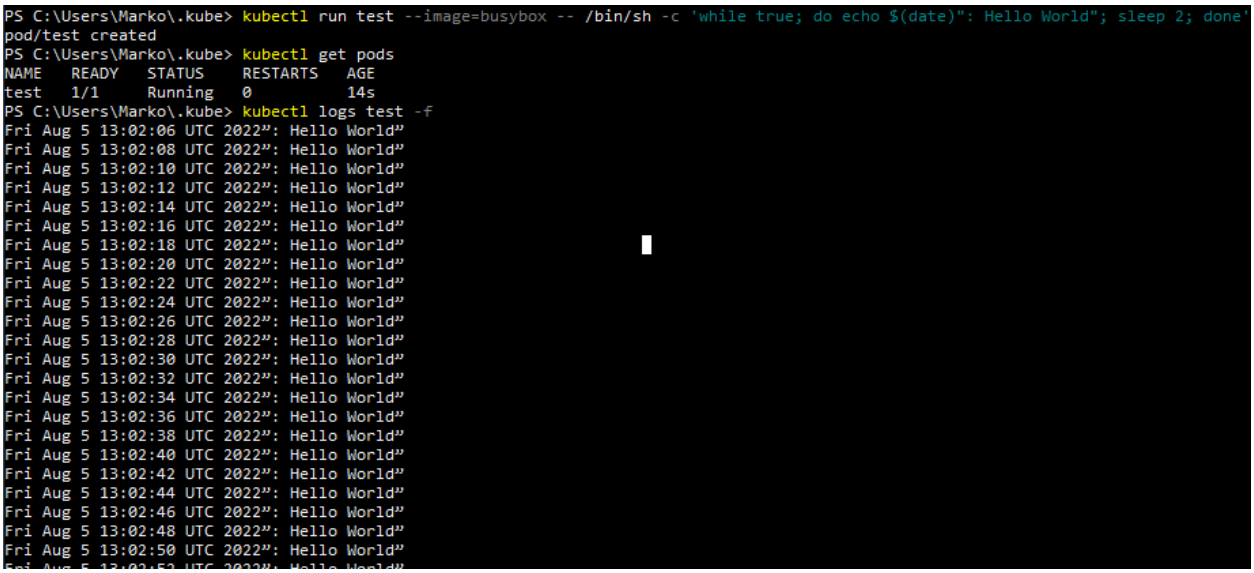
kubectl run test --image=busybox -- /bin/sh -c 'while true; do echo $(date)": Hello World"; sleep 2; done'

te provjerimo da li se sve ispravno postavilo naredbom:

kubectl get pod

logove možemo ispisati sa:

kubectl logs test -f



Slika12: ispis dnevnika test čahure

Izvor: vlastita izrada

4. K3s ulaz

Nakon postavlja lokalnog kubectla riješen je problem kako upravljati klasterom bez direktnog spajanja na klaster. Stoga ćemo započet ćemo Postavljanjem jednostavne web stranice. Možemo izravno implementirati pomoću kubectl-a. Međutim, to nije tipičan način postavljanja stvari. Općenito se koriste YAML konfiguracijske datoteke i to je ono što ćemo koristiti u ovom primjeru.

Konfiguracija implementacije prikazana je u nastavku.

Napravimo datoteku, mysite.yaml sa sljedećim sadržajem.

SimpleSSPClient.java | nestRPI.xml | nest.xml | mysite.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Slika13: mysite yaml

Izvor: vlastita izrada

Nazvali smo našu implementaciju nginx-deployment s istoimenom oznakom aplikacije. Naveli smo da želimo jednu repliku što znači da će biti kreirana samo jedna čahura. Također smo naveli jedan kontejner, koji smo nazvali nginx. Naveli smo da slika bude nginx:1.14.2. To znači da će pri implementaciji k3s preuzeti nginx sliku s Docker koncentratora (eng. hub) i od nje stvoriti čahuru. Konačno smo naveli pristup kontejneru (eng. Container port) od 80, što znači da će **unutar** spremnika čahura slušati na portu 80.

Naglašeno je "unutar kontejnera" jer je to važna razlika. Kako smo konfigurirali kontejner, dostupan je samo unutar kontejner-a i dalje je ograničen na **internu mrežu**. Ovo je neophodno kako bi se omogućilo slušanje više kontejner-a na istim portovima kontejner-a. Drugim riječima, s ovom konfiguracijom, neki drugi pod mogao bi slušati i na svom portu kontejnera 80 i nije u sukobu s ovim.

Kao što smo već naglasili pristup je konfiguriran samo unutar čahure i to ga ograničava na internu mrežu. Iako imamo lokalni kubectl i možemo upravljati klasterom izvan njega samog, nemamo način pristupiti čahurama. Klaster treba znati na koji način i kojom rutom doći do određene čahure.

Da bismo omogućili pristup ovoj čahuri, potrebna nam je konfiguracija usluge.

4.1. Konfiguracija usluge

U Kubernetesu je usluga apstrakcija. Pruža način za pristup čahuri ili skupu čahura. Korisnik se povezuje s uslugom i usluga usmjerava na jednu čahuru ili balansira opterećenje na više čahura ako je definirano više replika čahura.

Usluga se može specificirati u istoj konfiguracijskoj datoteci. Odvojite konfiguracijska područja s --- , te Dodajte sljedeće na mysite.yaml

Slika14: konfiguracija usluge

Izvor: vlastita izrada

U ovoj smo konfiguraciji našu uslugu nazvali nginx-service. Osigurali smo selektor aplikacije: nginx-deployment. Ovako usluga bira kontejnere aplikacija u koje usmjerava. Zapamtite, dali smo oznaku aplikacije za naš spremnik kao nginx-deployment. To je ono što će servis koristiti da pronađe naš kontejner. Na kraju smo naveli da je servisni protokol TCP i da usluga sluša na portu 80. S ovime smo definirali kojom rutom doći do čahure unutar klastera no i dalje trebamo način da promet izvan klastera dođe do same usluge.

4.2. Konfiguracija rute ulaza

Konfiguracija ulazne rute određuje kako promet izvan našeg klastera dovesti do usluga unutar našeg klastera. k3s dolazi unaprijed konfiguriran s Traefik-om kao kontrolerom ulaza. Stoga ćemo zapisati našu konfiguraciju ulaza specifičnu za Traefik, koristeći Traefik-ovu prilagođenu definiciju resursa (CRD) rute ulaza. Dodajte sljedeće na mysite.yaml.

```
33  ---
34  apiVersion: traefik.containo.us/v1alpha1
35  kind: IngressRoute
36  metadata:
37    name: nginx-ingress
38  spec:
39    entryPoints:
40      - web
41    routes:
42      - match: Path(`/`)
43        kind: Rule
44        services:
45          - name: mysite-nginx-service
46            port: 80
```

Slika15: konfiguracija rute ulaza

Izvor: vlastita izrada

U ovoj smo konfiguraciji zapisu ulazne rute nazvali nginx-ingress. Za ulazne točke, k3s implementira Traefik s dva, web i web secure za http i https protokole. Ovdje iako ne preporučen koristimo radi jednostavnosti web za http (npr. port klastera 80).

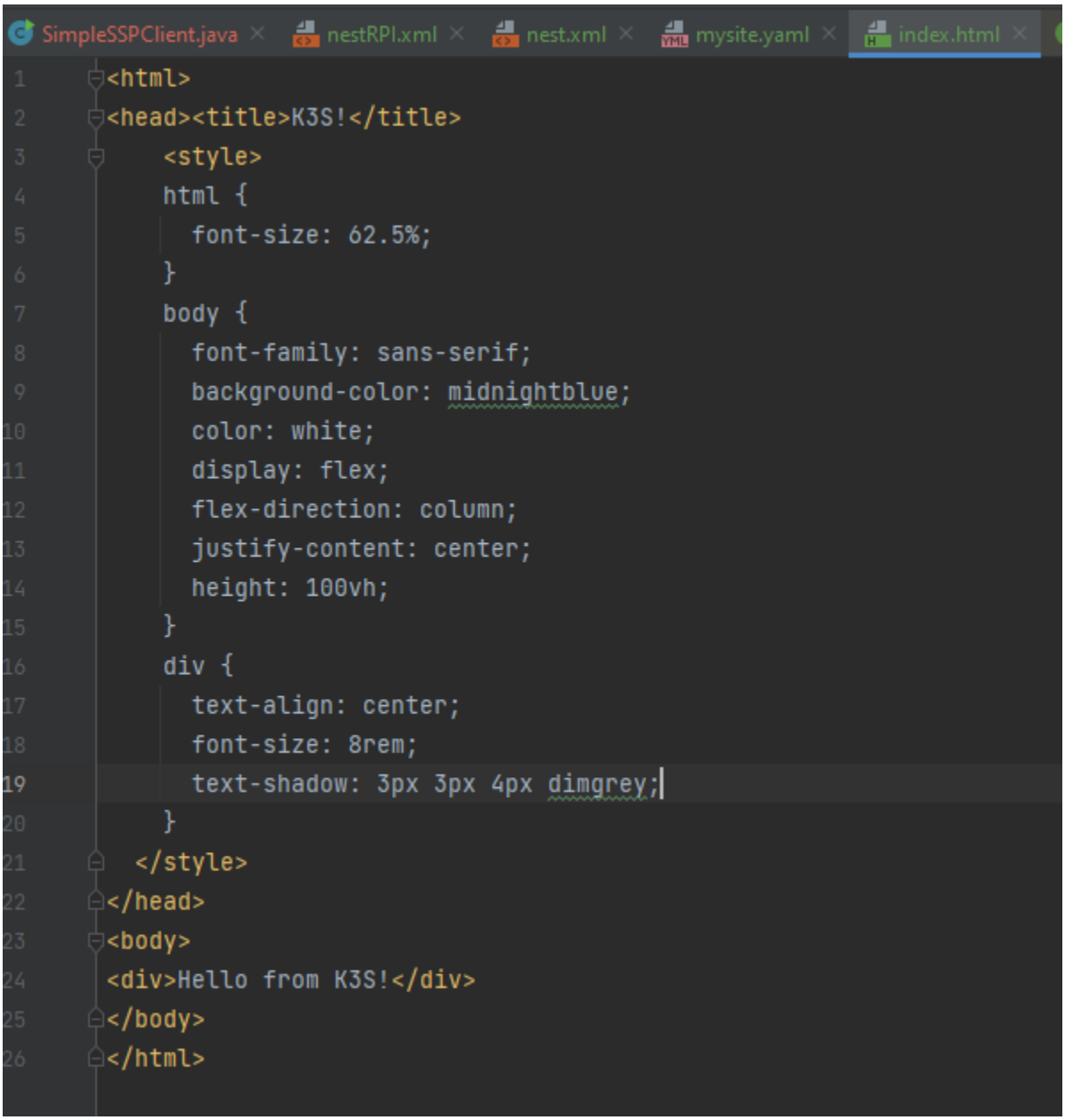
Zatim definiramo rutu koja odgovara korijenskom putu. Traefik definira dva uparivača za uparivanje puta. Put i prefiks putanje. Put se koristi za podudaranje točnih staza, a

prefiks putanje se koristi za podudaranje početka puta.. Ovdje je jednostavno i odgovara samo /. I konačno, kažemo ulaznoj ruti da usmjeri do naše mysite-nginx-service na portu 80.

Sva ova konfiguracija u osnovi govori, kada dođe HTTP promet, a put odgovara / točno, usmjerite ga na uslugu specificiranu imenom mysite-nginx-service i usmjerite ga na port 80 na usluzi. Ovo povezuje dolazni HTTP promet s uslugom koju smo ranije definirali.

4.3. Upogonjavanje

Što se konfiguracije tiče to je sve. Ako sada upogonimo, dobili bismo zadanu nginx stranicu no stvorit ćemo nešto vrlo jednostavno ali prilagođeno za upogonjavanje.



Slika16: indeks.html

Izvor: vlastita izrada

Još nismo pokrili mehanizme pohrane u Kubernetesu, pa ćemo malo varati i samo pohraniti ovu datoteku u Kubernetes konfiguracijsku kartu. Ovo nije preporučeni način postavljanja web stranice, ali će funkcionirati za svrhu gdje želimo pokazati primjer upogonjavanja.

kubectl create configmap mysite-html --from-file index.html

```
C:\Users\Marko\.kube>kubectl create configmap mysite-html --from-file index.html
configmap/mysite-html created
```

Slika17: kreiranje konfiguracijske mape

Izvor: vlastita izrada

Ova naredba stvara resurs configmap pod nazivom mysite-html iz lokalne datoteke index.html. Ovo u biti pohranjuje datoteku (ili skup datoteka) unutar Kubernetes resursa koji možemo pozvati u konfiguraciji. Obično se koristi za pohranjivanje konfiguracijskih datoteka (odatle i naziv) pa ga ovdje malo zlorabimo. Kasnije ćemo raspravljati o pravilnim rješenjima za pohranu u Kubernetesu.

S kreiranom konfiguracijskom mapom, montirajmo je u naš nginx spremnik. To radimo u dva koraka. Prvo, moramo odrediti volumen, pozivajući mapu konfiguracije. Zatim moramo montirati volumen u nginx spremnik. Dovršite prvi korak dodavanjem sljedećeg ispod oznake specifikacije unutar datoteke mysite.yaml.

```
25      volumes:
26        - name: html-volume
27          configMap:
28            name: mysite-html
```

Slika18: konfiguracija nxinx volumena

Izvor: vlastita izrada

To govori Kubernetesu da želimo definirati volumen s imenom html-volume i taj volumen bi trebao sadržavati sadržaj konfiguracijske mape pod nazivom html-volume koji smo stvorili u prethodnom koraku.

Zatim, u specifikaciji spremnika nginx dodajte sljedeće.

```
22        volumeMounts:
23          - name: html-volume
24            mountPath: /usr/share/nginx/html
```

Slika19: konfiguracija spremnika

Izvor: vlastita izrada

To govori Kubernetesu da za nginx spremnik želimo montirati volumen pod nazivom html-volume na stazi **unutar** čahure /usr/share/nginx/html. Odatle nginx slika služi HTML-u. Montiranjem našeg volumena na taj put, zamijenili smo zadani sadržaj našim sadržajem volumena.

Sada smo spremni za upogoniti čahuru. To možemo učiniti sa

kubectl apply -f mysite.yaml

Trebali biste vidjeti nešto slično sljedećem.

```
C:\Users\Marko\.kube>kubectl apply -f mysite.yaml
deployment.apps/mysite-nginx created
service/mysite-nginx-service created
ingressroute.traefik.containo.us/mysite-nginx-ingress created
```

Slika20: upogonjavanje mysite.yaml

Izvor: vlastita izrada

To znači da je Kubernetes kreirao resurse za svaku od tri konfiguracije koje smo naveli. Provjerite status čahure sa

kubectl get pods

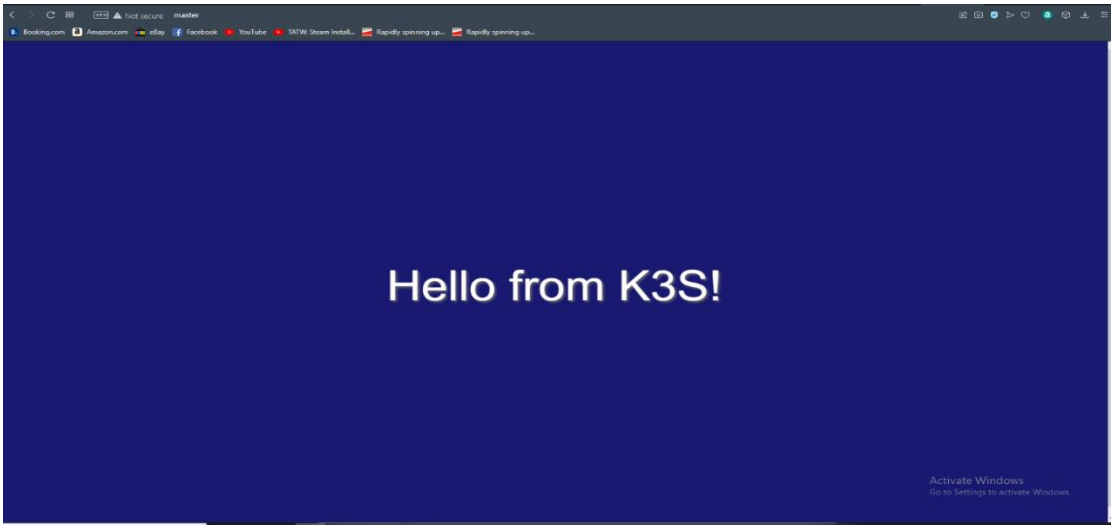
```
C:\Users\Marko\.kube>kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
mysite-nginx-5559ffd776-5cqhm   1/1     Running   0          78s
```

Slika21: ispis čahura

Izvor: vlastita izrada

Ako vidite status ContainerCreating, dajte mu malo vremena i ponovno pokrenite kubectl get pods. Obično će prvi put potrajati neko vrijeme jer k3s mora preuzeti nginx sliku da bi napravio pod. Nakon nekog vremena trebali biste dobiti status Running.

Nakon što se pod pokrene, vrijeme je da ga isprobamo. Otvorite preglednik i upišite master u adresnu traku.



Slika22: prikaz stranice

Izvor: vlastita izrada

5. Upogonjavanje

Kubernetes upogonjavanje objekt je resursa koji pruža deklarativna ažuriranja za aplikacije temeljene na spremniku. Razumijevanje načina na koji Kubernetes-ov objekt za implementaciju funkcionira i prednosti koje pruža znači pobliže promotriti arhitekturu Kubernetes sustava.

Kubernetes radi u klasterima koji se sastoje od glavnog čvora i nekoliko radnih čvorova koji ugošćuju podove u koje su smješteni pojedinačni spremnici. Implementacija Kubernetesa izravno se odnosi na same module, pružajući opis željenog stanja modula. Zamislite Kubernetes upogonjavanje kao neku vrstu nacrta koji detaljno opisuje kako bi moduli trebali funkcionirati i kako bi trebalo izgledati radno opterećenje klastera. Kubernetes kontroleri zatim uskaču kako bi osigurali da moduli odgovaraju unaprijed određenim karakteristikama koje je uspostavilo Kubernetes upogonjavanje.

Često je problem izabrati pravi način za upogonjavanje pogotovo s obzirom na to da je izbor iznimno velik. Generalno ne postoji dobri i loši načini već pogodni odnosno ne pogodni za određene čahure

5.1. Helm upogonjavanje

Kako je k3s je potpuno certificirana CNCF (Cloud Native Computing Foundation) Kubernetes verzija. To znači da se Helm, alat za automatiziranje stvaranja, pakiranja, konfiguracije i postavljanja aplikacija na Kubernetes, može izravno koristiti bez daljnje konfiguracije.

Za instalaciju helm-a na windows sustavu dovoljno je da skinemo verziju za windows sustav s navedenog linka: https://github.com/helm/helm/releases

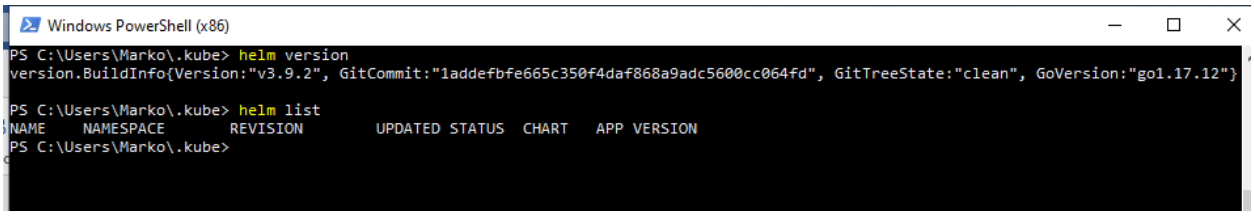
Te dodavanjem puta u env varijable.

Instalaciju možemo provjeriti sa naredbom:

Helm version

Te prostor imena sa

helm list



slika23: ispis verzije helma i prostora imena helma

Izvor: vlastita izrada

Službeni Helm grafikon za MQTT Mosquitto Broker dostupan je na https://artifacthub.io/packages/helm/k8s-at-home/mosquitto. Možete instalirati Mosquitto pokretanjem sljedećih naredbi

helm repo add k8s-at-home https://k8s-at-home.com/charts/

helm repo update

kreiranje dodijeljenog prostora imena

kubectl create namespace mqtt

Instalacija mosquitto-a

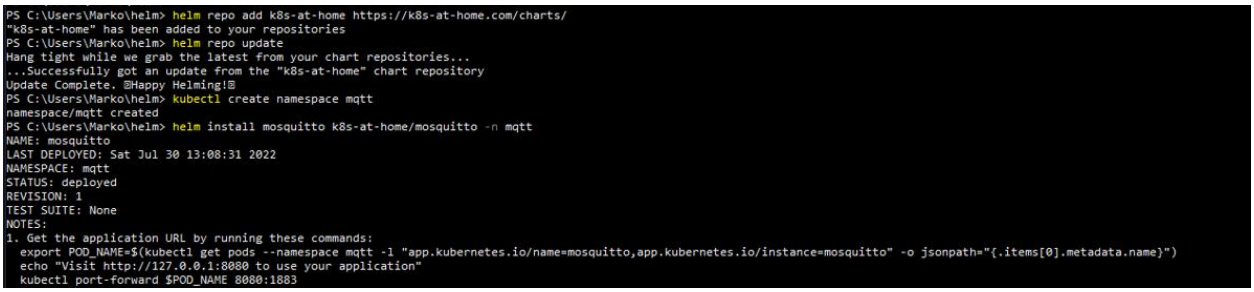
helm install mosquitto k8s-at-home/mosquitto -n mqtt

provjera instalacije

helm list -n mqtt

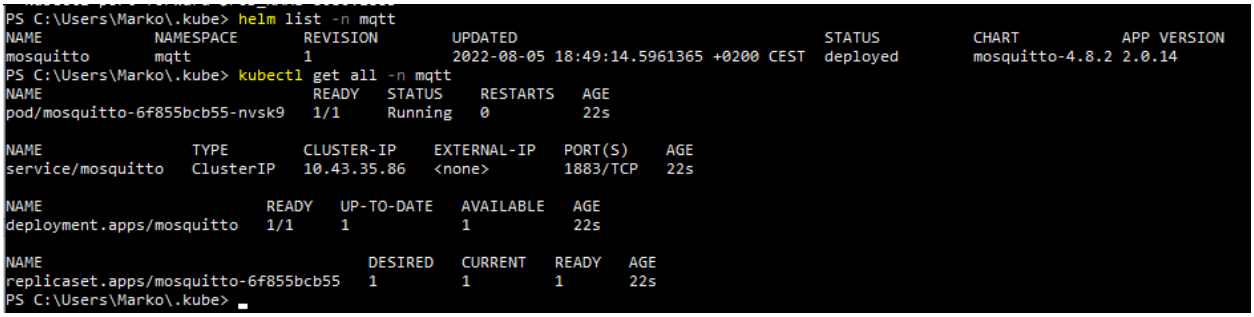
Provjera resursa

kubectl get all -n mqtt



Slika24: kreiranje repozitorija te imenskog prostora mqtt-a

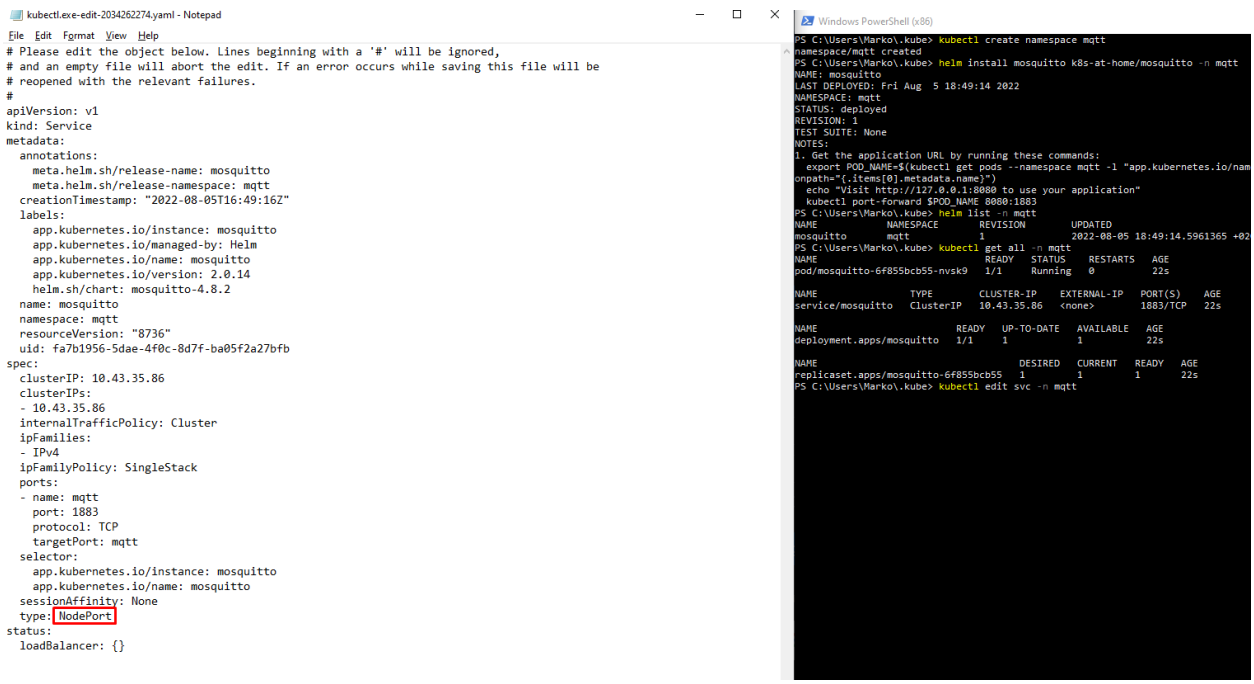
Izvor: vlastita izrada



Slika25: ispis podataka o mqtt-u i prikat usluge i čahure mqtt

Izvor: vlastita izrada

Helm Grafikon instalira implementaciju i uslugu tipa ClusterIP. Nažalost, vrsta usluge nije parametrizirana i ne može se promijeniti pomoću datoteke values.yaml. Dakle, da biste pristupili MQTT brokeru izvan Kubernetes klastera, trebamo ručno promijeniti vrstu u NodePort uređivanjem usluge.



Slika26: promjena tipa usluge

Izvor: vlastita izrada

Možemo provjeriti promjene s naredbom:

Kubectl svc -n mqtt

```
PS C:\Users\Marko> kubectl get svc -n mqtt
NAME        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
mosquitto   NodePort   10.43.249.125   <none>        1883:32640/TCP   2d7h
```

Slika27: ispis usluge mqtt

Izvor: vlastita izrada

Ako imate Mosquitto klijent instaliran na vašem lokalnom prijenosnom računalu, možete se pretplatiti na posrednika koji radi na klasteru i objavljivati poruke.

Mosquitto je moguće preuzet na linku : https://mosquitto.org/download/

Na temu se možemo spojiti sljedećom naredbom:

mosquitto_sub -h master.local -p 32640 -t test

```
PS C:\Users\Marko> kubectl get svc -n mqtt
NAME        TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
mosquitto   NodePort   10.43.249.125   <none>        1883:32640/TCP   2d7h
PS C:\Users\Marko> mosquitto_sub -h master.local -p 32640 -t test
Hello from master
Hello from master
Hello from master
Hello from master
```

Slika28: ispis testa

Izvor: vlastita izrada

5.2. Docker upogonjavanje

Docker je otvorena platforma za razvoj, isporuku i pokretanje aplikacija. Docker nam omogućuje da odvojimo svoje aplikacije od naše infrastrukture kako bismo mogli brzo isporučiti softver. S Dockerom možemo upravljati svojom infrastrukturom na isti način na koji upravljamo svojim aplikacijama. Činjenica da možemo upravljati infrastrukturom na tako jednostavan način nam je iznimno bitna kada govorimo o k3s-u. Zamislimo da želimo upogoniti pod koji će objavljivati aplikaciju koja zahtijeva nekakvu ovisnost prema nekoj biblioteci. To ne bi bilo moguće ako prethodno na svakom čvoru u klasteru instaliramo određenu biblioteku.

Na većim klasterima to bi iziskivalo poteškoće u menadžmentu klastera a i pružilo bi velik manualni posao s velikim mogućnostima za grešku. Tu nam izniman značaj ima docker, jer na omogućuje da aplikaciju podignemo u virtualnom okruženju unutar docker kontejnera. Te na kraju k3s jednostavno može objaviti kontejner unutar poda, bez potrebe za svim silnim konfiguriranjem.

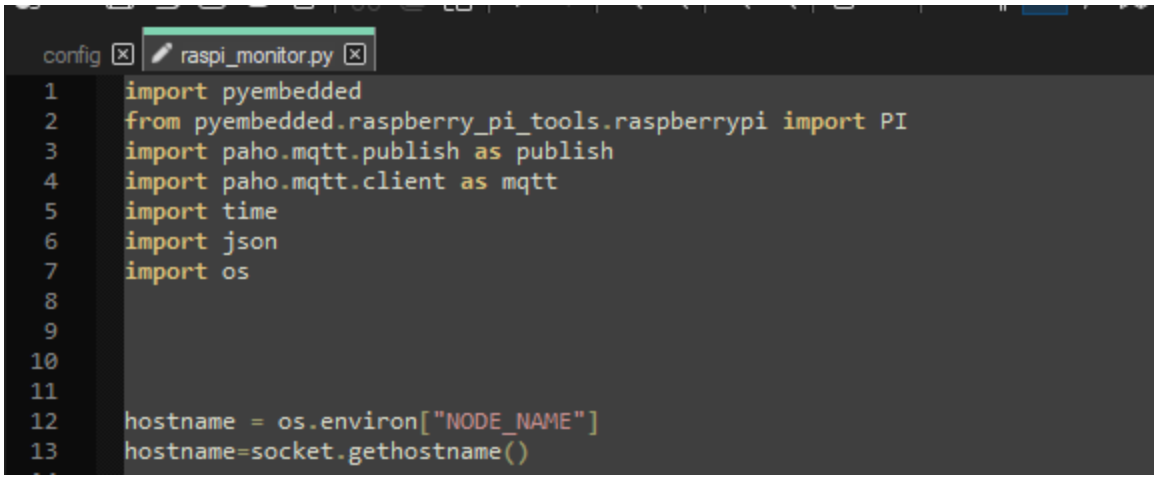
Iako bi preporučen način bio da se sve odradi bez ssh konekcije na klaster, radi jednostavnosti ćemo docker dio pokazati na klasteru s obzirom na to da da zbog tehničkih nemogućnosti na lokalnoj mašini instalacije wsl2 systema kojeg zahtijeva docker. Stoga ćemo dio vezan za docker prikazati na samom klasteru s obzirom na to da su čvorovi u klasteru koriste linux sustav.

Postavka se sastoji od tri datoteke:

1. Python kod
2. Docker datoteke (Dockerfile)
3. Yaml skupa poslužiteljskih procesa (eng. Daemonset)

5.2.1 Python kod

Koristimo Python knjižnicu pyembedded za čitanje podataka iz sustava i paho-mqtt za objavljivanje u MQTT. Koristimo također os biblioteku kako bi dinamički dohvatili prostor imena sa svakog čvora.



Slika29: python kod

Izvor: vlastita izrada

Metoda get_data će dohvatiti podatke pomoću metoda iz pyembedded biblioteke te će ih vratiti u json formatu.

```python
pi = PI()

def get_data():

    ram_raw = pi.get_ram_info()
    disk_raw = pi.get_disk_space()
    cpu_raw = pi.get_cpu_usage()

    temperature_raw=pi.get_cpu_temp()


    temperature = {
        "name": "temperature",
        "value": temperature_raw,
        "unit": "°C",
        "node": hostname
    }

    ram = {
        "name": "ram",
        "value": {
            "total": round(int(ram_raw[0])/1000000,1),
            "used": round(int(ram_raw[1])/1000000,1),
            "free": round(int(ram_raw[2])/1000000,1),
            },
        "unit": "GB",
        "node": hostname
        }


    disk = {
        "name": "disk",
        "value": {
            "total": disk_raw[0],
            "used": disk_raw[1],
            "free": disk_raw[2],
            },
        "unit": "GB",
        "node": hostname
        }


    cpu = {
        "name": "cpu",
        "value": cpu_raw,
        "unit": "Percent",
        "node": hostname
        }

    return temperature, ram, disk, cpu

```

Slika30: python kod

Izvor: vlastita izrada

Na kraju će se podatci iterativno objavljivati putem MQTT klijenta.

```
67  mqttc = mqtt.Client("k3s")
68
69  mqttc.connect("mosquitto.default.svc.cluster.local", 1883, 60)
70  mqttc.loop_start()
71  while True:
72      mqttc.publish("test","Hello")
73
74
75      temperature, ram, disk, cpu = get_data()
76      mqttc.publish("temperature",str(temperature))
77      mqttc.publish("ram",json.dumps(ram))
78      mqttc.publish("disk",json.dumps(disk))
79      mqttc.publish("cpu",json.dumps(cpu))
80
81      time.sleep(2)
```

Slika31: python kod

Izvor: vlastita izrada

5.2.2 Dockerfile

test-mqttt.py | Dockerfile.test | Dockerfile.monitor

```
1  FROM python
2
3  RUN pip install psutil pyembedded paho-mqtt
4
5  COPY raspi_monitor.py ./
6
7  CMD ["python","./raspi_monitor.py"]
8
```

Slika32: python kod

Izvor: vlastita izrada

Docker datoteka je vrlo jednostavna. Datoteka je niz naredbi kako izgraditi docker sliku.

Iz python biblioteke pomoću pip programa se instaliraju potrebne zavisnosti. Kopira se python kod te putem naredbe python i koja kao parametar prima python kod.

Sliku možemo izgraditi putem sljedeće naredbe:

docker build -t mperkok3s/rpi-k3s:1.0 -f Dockerfile .

```
mperko@master:~/kubernetes-on-raspberry-pi/apps/4_raspiMonitor $ sudo docker build -t mperkok3s/rpi-k3s:1.0 -f Dockerfile.monitor .
Sending build context to Docker daemon   12.8kB
Step 1/4 : FROM python
latest: Pulling from library/python
cfc947b533a3: Pull complete
9ca36aa4204d: Pull complete
1fdcd2014de7: Pull complete
e288b20a6167: Pull complete
9c814ed08943: Pull complete
5f6c125e963d: Pull complete
1cea02ceb178: Pull complete
c3e02b5c8de7: Pull complete
5fbada9afe05: Pull complete
Digest: sha256:b1a4d9bcebe696cf805803f98747d3ccd885504bd1473dc45db2014d08622de5
Status: Downloaded newer image for python:latest
 ---> 5010a1a4e02e
Step 2/4 : RUN pip install psutil pyembedded paho-mqtt
 ---> Running in 91dec0fcb0bd
Collecting psutil
  Downloading psutil-5.9.1.tar.gz (479 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 479.1/479.1 KB 1.8 MB/s eta 0:00:00
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Collecting pyembedded
  Downloading pyembedded-3.5.tar.gz (8.2 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting paho-mqtt
  Downloading paho-mqtt-1.6.1.tar.gz (99 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.4/99.4 KB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting pyserial
  Downloading pyserial-3.5-py2.py3-none-any.whl (90 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.6/90.6 KB 2.8 MB/s eta 0:00:00
Building wheels for collected packages: psutil, pyembedded, paho-mqtt
  Building wheel for psutil (pyproject.toml): started
  Building wheel for psutil (pyproject.toml): finished with status 'done'
  Created wheel for psutil: filename=psutil-5.9.1-cp310-cp310-linux_aarch64.whl size=282720 sha256=a4d0768afe8fa7193a4e3498100871eecdd
f39fb410c905c4c2477b87745f394
  Stored in directory: /root/.cache/pip/wheels/4f/4f/0f/1cb5a8ab74b6f9128b9cb2025eb1d7ea3c2d5854a55af27e5b
  Building wheel for pyembedded (setup.py): started
  Building wheel for pyembedded (setup.py): finished with status 'done'
  Created wheel for pyembedded: filename=pyembedded-3.5-py3-none-any.whl size=11547 sha256=fbb32e075b04e8ee3084bff6eac08de785a05c155b8
66b490bc9ab544379af40
  Stored in directory: /root/.cache/pip/wheels/2f/37/4a/871f1b4580772161bc5facf305a9a4e62421a392b4e03a34e2
  Building wheel for paho-mqtt (setup.py): started
  Building wheel for paho-mqtt (setup.py): finished with status 'done'
  Created wheel for paho-mqtt: filename=paho_mqtt-1.6.1-py3-none-any.whl size=62133 sha256=674763a75b408d2a010a24a9430d54413f6d273b69a
385585b693b6faa5c03b5
  Stored in directory: /root/.cache/pip/wheels/8b/bb/0c/79444d1dee20324d442856979b5b519b48828b0bd3d05df84a
Successfully built psutil pyembedded paho-mqtt
Installing collected packages: pyserial, paho-mqtt, pyembedded, psutil
Successfully installed paho-mqtt-1.6.1 psutil-5.9.1 pyembedded-3.5 pyserial-3.5
```

Slika33: docker pull

Izvor: vlastita izrada

Sliku možemo gurnuti u repozitorij putem sljedeće naredbe:

docker push mperkok3s/rpi-k3s:1.0

Slika34: docker push

Izvor: vlastita izrada

5.3. Skup poslužiteljskih procesa

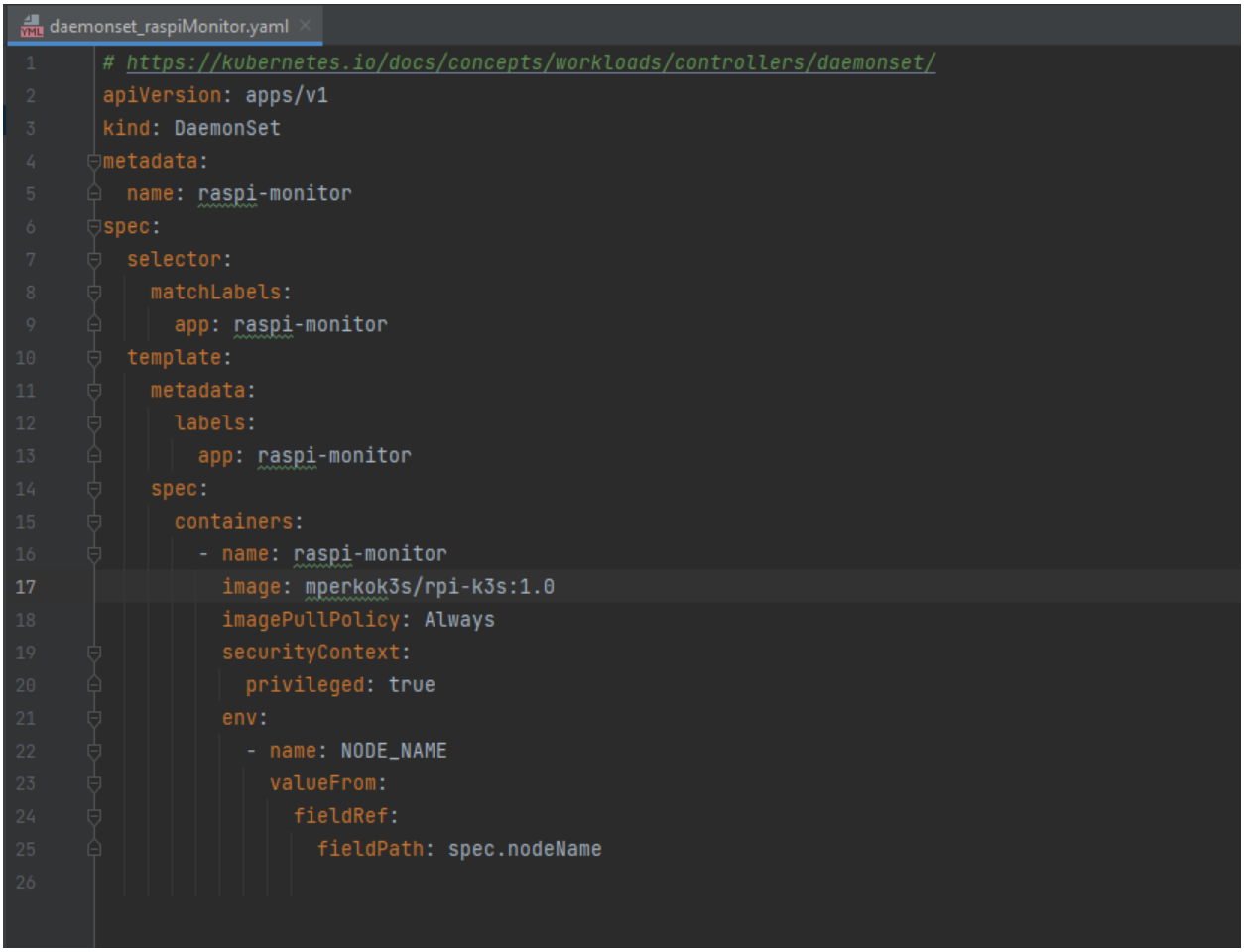
Skup (eng. set) poslužiteljskih procesa (eng. daemon) je posebna vrsta upogonjavanja.

Skup poslužiteljskih procesa osigurava da svi ili samo neki čvorovi pokreću kopiju čahura. Kako se čvorovi dodaju u klaster, dodaju im se i čahure. Kako se čvorovi uklanjaju iz klastera, te se jedinice skupljaju u smeće. Brisanjem skupa poslužiteljskih procesa očistit ćete čahure koje je stvorio. Imajte na umu da nije moguće obrisati čahuru ako ne obrišete skup poslužiteljskih procesa.

U jednostavnom slučaju, jedan skup poslužiteljskih procesa, koji pokriva sve čvorove, koristio bi se za svaku vrstu poslužiteljskih procesa. Složenija postavka može koristiti više skupova poslužiteljskih procesa za jednu vrstu poslužiteljskog procesa, ali s različitim oznakama i/ili različitim zahtjevima za memoriju i procesor za različite vrste hardvera.

Zbog toga nam je iznimno korisnika kod monitoring aplikacija, prenosivosti aplikacija , pokretanja opcija pohrane na svakom čvoru i slično.

Skup poslužiteljskih procesa iako drugačiji način upogonjavanja sama yaml datoteka nije drugačija u puno stvari od ostalih načina upogonjavanja osim što vuče docker sliku. Koristit ćemo docker sliku iz prethodnoga podpoglavlja. Te env dijelom po kojem znamo na kojem čvoru je što upogonjeno.



Slika35: prikaz konfiguracije skupa posluziteljskih procesa

Izvor: vlastita izrada

Sljedećom naredbom možemo upogoniti Skup poslužiteljskih procesa. Koristit ćemo docker sliku iz prethodnoga podpoglavlja.

kubectl create -f daemonset_raspiMonitor.yaml

```
PS C:\Users\Marko\kubernetes-on-raspberry-pi\apps\4_raspiMonitor> kubectl create -f daemonset_raspiMonitor.yaml
daemonset.apps/raspi-monitor created
```

Slika36: upogonjavanje skupa posluziteljskih procesa

Izvor: vlastita izrada

a pretplatom na teme putem mosquitto-a mogu se dobiti parametri sustava.

mosquitto_sub -h master.local -p 32640 -t raspi/+/temperature

```
PS C:\Users\Marko> mosquitto_sub -h master.local -p 32640 -t raspi/+/temperature
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.043, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.043, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.043, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.53, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.069, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.53, 'unit': 'T C', 'node': 'master'}
{'name': 'temperature', 'value': 53.556, 'unit': 'T C', 'node': 'node1'}
{'name': 'temperature', 'value': 54.53, 'unit': 'T C', 'node': 'master'}
```

Slika37: mosquito pretplata na tempreaturu procesora

Izvor: vlastita izrada

6. Kontrolna ploča

Kontrolne ploče nisu potrebne, ali su jednostavne za dobivanje informacija o Kubernetes resursima. No kada imamo velike klastere gdje moramo voditi računa u resursima klastera tada nam je kontrolna ploče iznimno koristan alat. Pomoću kontrolne ploče možemo pratiti sve resurse klastera čak i kada nemamo nikakav način da pristupimo klasteru.

Također postoji nekoliko različitih dashboarda s različitim svrhama kao što su:

- Kubernetes dashboard
- Rancher dashboard
- Traefik dashboard
- Longhorn dashboard
- Itd.

6.1. Kubernetes dashboard

Prvo trebamo instalirati kontrolnu ploču. Prvo postavimo varijable okruženja

```
GITHUB_URL=https://github.com/kubernetes/dashboard/releases

VERSION_KUBE_DASHBOARD=$(curl -w '%{url_effective}' -I -L -s -S
${GITHUB_URL}/latest -o /dev/null | sed -e 's|.*/||')
```

Slika38: prikaz naredbi

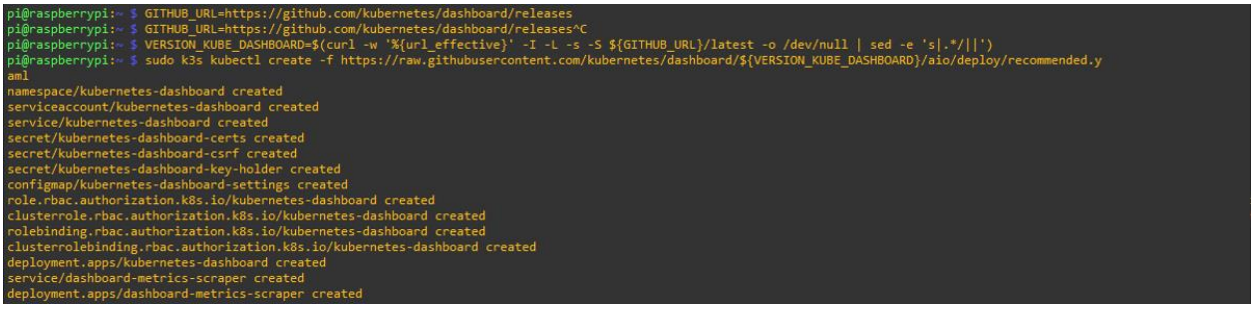
Izvor: vlastita izrada

ako smo postavili varijable možemo pozvati sljedeću naredbu koja će se pobrinuti o instalaciji.

```
kubectl create -f
https://raw.githubusercontent.com/kubernetes/dashboard/${VERSION_KUBE_
DASHBOARD}/aio/deploy/alternative.yaml
```

Slika39: docker push

Izvor: vlastita izrada



Slika40: ispis instalacije

Izvor: vlastita izrada

Poželjno je postaviti prava pristupa kontrolnoj ploči Iz sigurnosnih razloga, preporučena konfiguracija daje kontrolnoj ploči ServiceAccount ograničen pristup Kubernetes resursima. To može spriječiti slučajno otkrivanje osjetljivih podataka klastera kao što su tajne ili certifikati.

Uz to, da biste iskoristili svu funkcionalnost web korisničkog sučelja, trebat će vam račun usluge s administrativnim (super korisničkim) povlasticama, odnosno račun usluge s ulogom klastera administratora klastera.

Stoga ćemo urediti datoteku admin-user.yml:

admin-user.yml

```yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Slika41: admin-user.yml

Izvor: vlastita izrada

Izradite novi račun usluge putem

kubectl apply -f admin-user.yml

Izlaz terminala trebao bi potvrditi stvaranje novog računa usluge, kao i obvezivanje njegove uloge.

```
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
```

Slika42: kreiranje administratora

Izvor: vlastita izrada

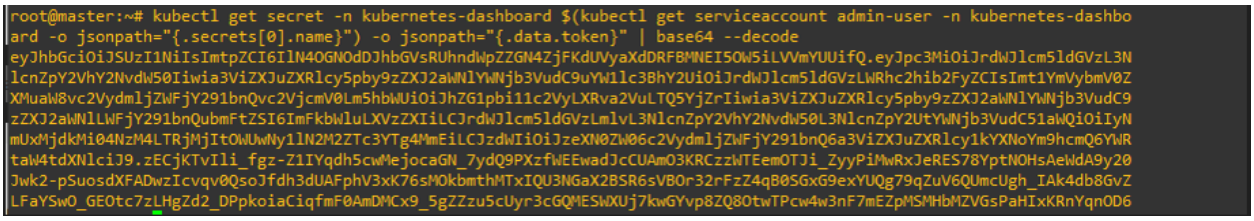
6.2. Pristup kontrolnoj ploči Kubernetes

Preporučeni način za pristup Kubernetes nadzornoj ploči je putem tokena nositelja. To omogućuje veliku fleksibilnost u upravljanju dopuštenjima.

Da biste dobili token nositelja administratorskog korisnika ServiceAccount, pokrenite sljedeću naredbu.

kubectl get secret -n kubernetes-dashboard $(kubectl get serviceaccount admin-user -n kubernetes-dashboard -o jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}" | base64 –decode

Čim izvršite instrukciju, token nositelja računa usluge bit će ispisan na konzoli. Dugačak je izlaz sličan onom prikazanom u nastavku.



Slika43: dekodirani ispis tokena

Izvor: vlastita izrada

Kopirajmo token nositelja na sigurno mjesto jer će uskoro biti potreban. Sada, preko lokalnog kubectla, otkrijte Kubernetes kontrolnu ploču sljedećom naredbom:

kubectl proxy

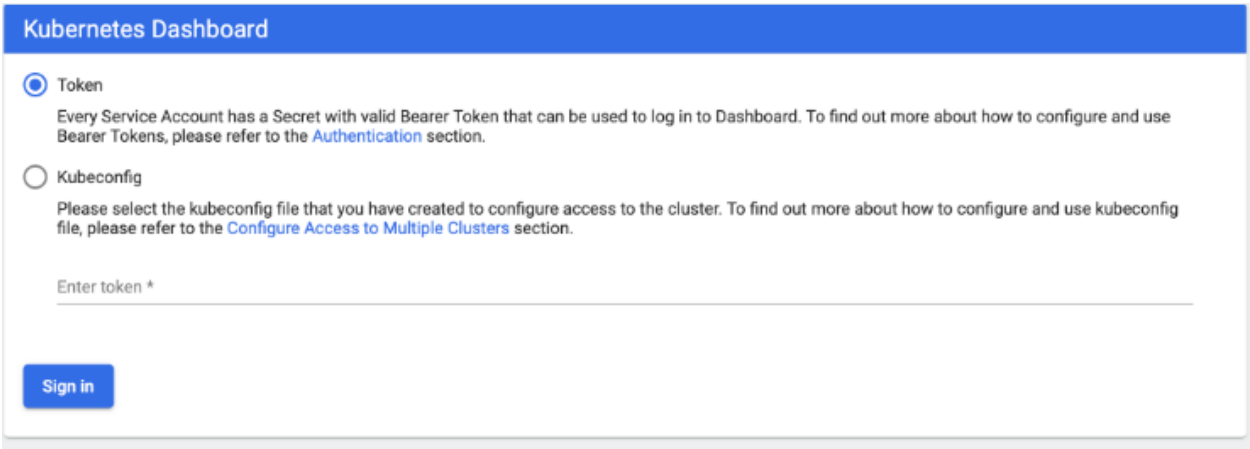
na terminalu će se pojaviti poruka koja pokazuje da je WebUI dostupan na 127.0.0.1:8001. Iz sigurnosnih razloga, Nadzorna ploča bit će dostupna samo sa stroja

koji izvodi naredbu kubectl proxy. Imajte to na umu, pogotovo ako je Kubernetes instaliran na udaljenom poslužitelju.

Nakon što je proxy aktivan, možete pristupiti web korisničkom sučelju usmjeravajući preglednik na sljedeću adresu:

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy

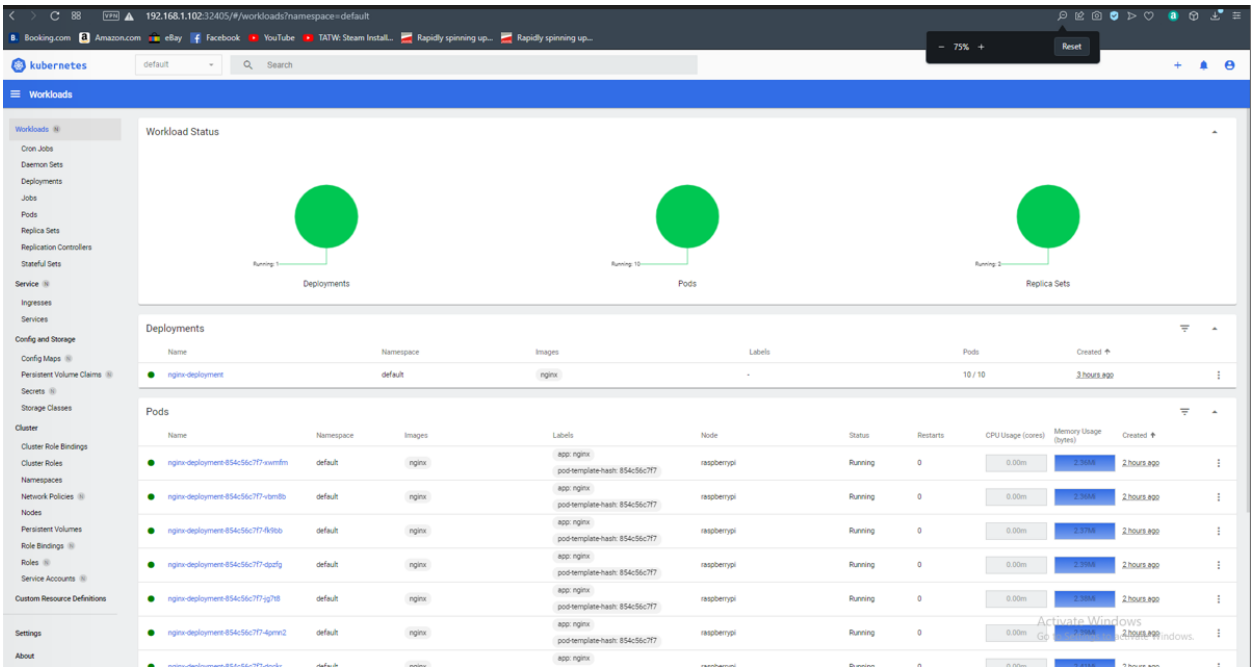
Trebali biste vidjeti zaslon sličan onom prikazanom u nastavku



Slika44: prikaz dijela za unos tokena nositelja

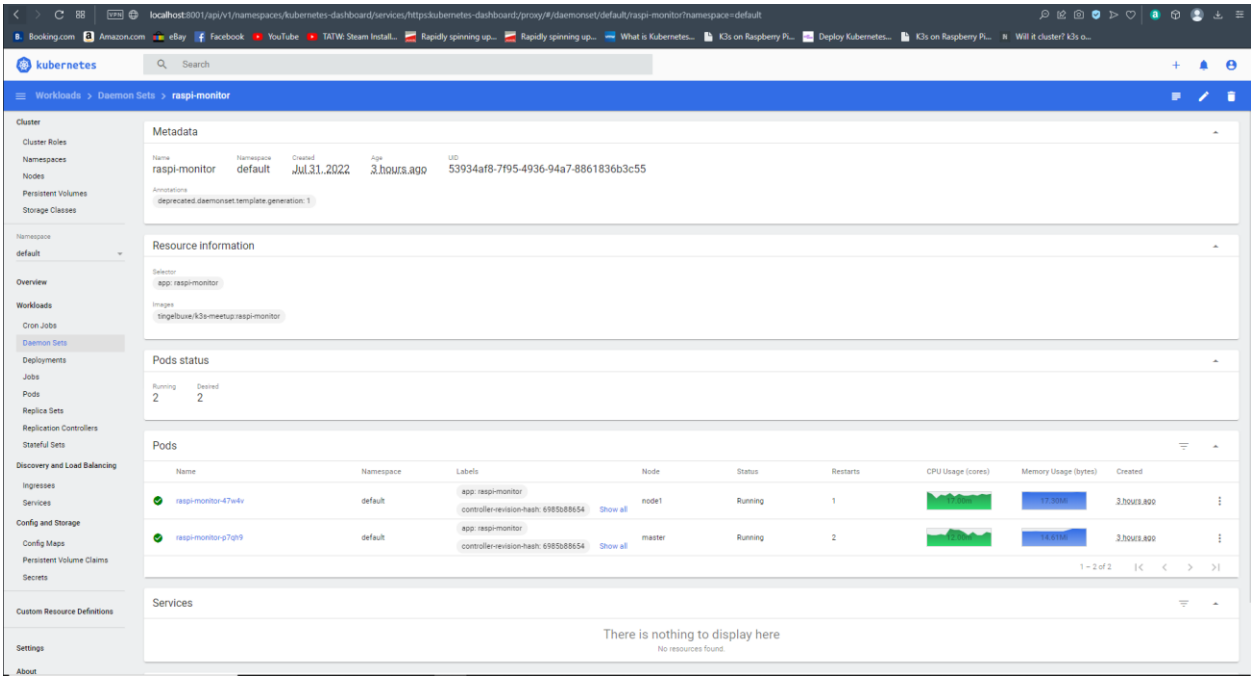
Izvor: vlastita izrada

Unesemo admin-korisnički bearer Token i kliknemo gumb Prijava za nastavak. Prikazat će nam se zaslon "Pregled" Kubernetes nadzorne ploče.



Slika45: prikaz dashboarda

Izvor: vlastita izrada



Slika46: prikaz dashboarda

Izvor: vlastita izrada

7. Pohrana

Datoteke na disku u spremniku su prolazne, što predstavlja neke probleme za netrivijalne aplikacije kada se izvode u spremnicima. Jedan od problema je gubitak datoteka kada se kontejner ruši. Kubelet ponovno pokreće spremnik, ali s čistim stanjem. Drugi problem se javlja kod dijeljenja datoteka između spremnika koji rade zajedno u čahuri. Kubernetesova apstrakcija volumena rješava oba ova problema.

Docker ima koncept volumena, iako je nešto labaviji i manje se njime upravlja. Docker volumen je direktorij na disku ili u drugom spremniku. Docker nudi drivere za volumen, ali je funkcionalnost donekle ograničena.

Kubernetes podržava mnoge vrste svezaka. Čahura može istovremeno koristiti bilo koji broj tipova volumena. Tipovi kratkotrajnog volumena imaju vijek trajanja modula, ali postojani volumeni postoje nakon vijeka trajanja modula. Kada pod prestane postojati,

Kubernetes uništava efemerne volumene; međutim, Kubernetes ne uništava trajne volumene. Za bilo koju vrstu volumena u danoj čahuri, podaci se čuvaju tijekom ponovnih pokretanja spremnika.

7.1. Volumeni

U svojoj srži, volumen je direktorij, moguće s nekim podacima u njemu, koji je dostupan spremnicima u čahuri. Način na koji taj imenik nastaje, medij koji ga podržava i njegov sadržaj određuju se određenim tipom volumena koji se koristi.

Da bismo koristili volumen, trebamo navesti volumene koji će se osigurati za Pod u i navesti gdje da se montiraju ti volumeni u spremnike. Proces u spremniku vidi prikaz datotečnog sustava sastavljen od početnog sadržaja slike spremnika, plus volumena (ako je definirano) montiranih unutar spremnika. Proces vidi korijenski datotečni sustav koji se u početku podudara sa sadržajem slike spremnika. Bilo koji zapis unutar hijerarhije datotečnog sustava, ako je dopušten, utječe na to što taj proces gleda kada izvrši naknadni pristup datotečnom sustavu. Volumeni se montiraju na navedenim putovima unutar slike. Za svaki spremnik definiran unutar Poda, morate neovisno odrediti gdje ćete montirati svaki volumen koji spremnik koristi.

Volumeni se ne mogu montirati unutar drugih volumena iako postoje obilazni putovi koji nude tu funkcionalnost to tada nije volumen. Također, svezak ne može sadržavati čvrstu vezu na bilo što u drugom volumenu.

Kubernetes podržava nekoliko vrsta tipova. S obzirom na to da ih ima iznimno puno a i mnogo ih je zastarjelo. Stoga ćemo spomenuti neke od najčešćih i nama najzanimljivijih

- azureDisk CSI migration
- configMap
- downwardAPI
- Regional persistent disks
- hostPath
- iscsi

- local
- nfs
- persistentVolumeClaim
- secret

7.2. Održivi volumeni

Upravljanje pohranom je poseban problem od upravljanja računalnim instancama. Podsustav PersistentVolume pruža API za korisnike i administratore koji apstrahira detalje o tome kako se pohrana pruža od načina na koji se troši. Da bismo to učinili, uvodimo dva nova API resursa: PersistentVolume i PersistentVolumeClaim.

PersistentVolume (PV) je dio pohrane u klasteru koji je dodijelio administrator ili ga je dinamički dodijelio pomoću Klasa pohrane. To je resurs u klasteru baš kao što je čvor resurs klastera. PV-ovi su dodaci za volumen poput Volumesa, ali imaju životni ciklus neovisan o bilo kojem pojedinačnoj čahuri koji koristi PV. Ovaj API objekt bilježi pojedinosti implementacije pohrane, bilo da je to NFS, iSCSI ili sustav za pohranu specifičan za pružatelja oblaka.

PersistentVolumeClaim (PVC) zahtjev je za pohranom od strane korisnika. Sličan je čahuri. Čahure troše resurse čvorova, a PVC-ovi troše PV resurse. Podovi mogu zahtijevati određene razine resursa (CPU i memorija). Zahtjevi mogu zahtijevati određenu veličinu i načine pristupa npr. mogu se montirati ReadWriteOnce, ReadOnlyMany ili ReadWriteMany.

Dok PersistentVolumeClaims dopuštaju korisniku da konzumira apstraktne resurse za pohranu, uobičajeno je da korisnici trebaju PersistentVolume s različitim svojstvima, kao što je izvedba, za različite probleme. Administratori klastera moraju biti u mogućnosti ponuditi razne PersistentVolume koji se razlikuju na više načina od veličine i načina pristupa, bez izlaganja korisnika pojedinostima o tome kako se ti volumeni implementiraju. Za te potrebe postoji resurs StorageClass.

7.3. Životni ciklus svezaka i zahtjeva

PV su resursi u klasteru. PVC-ovi su zahtjevi za tim resursima i također djeluju kao provjere potraživanja za resurs. Interakcija između PV-a i PVC-a slijedi ovaj životni ciklus

7.3.1. Opskrba

Postoje dva načina na koji se PV-ovi mogu osigurati: statički ili dinamički.

7.3.2. Statični

Administrator klastera stvara brojne PV-ove. Oni nose detalje o stvarnoj pohrani koja je dostupna korisnicima klastera. Oni postoje u Kubernetes API-ju i dostupni su za upotrebu

7.3.3. Dinamički

Kada nijedan od statičkih PV-ova koje je administrator stvorio ne odgovara korisnikovom PersistentVolumeClaimu, klaster može pokušati dinamički osigurati volumen posebno za PVC. Ova se opskrba temelji na StorageClasses: PVC mora zahtijevati klasu pohrane, a administrator mora stvoriti i konfigurirati tu klasu da bi se dogodila dinamička dodjela. Zahtjevi koji zahtijevaju klasu učinkovito onemogućuju dinamičku opskrbu za sebe.

Kako bi omogućio dinamičku pohranu na temelju klase pohrane, administrator klastera mora omogućiti kontroler pristupa DefaultStorageClass na API poslužitelju. To se može učiniti, na primjer, osiguravanjem da je DefaultStorageClass među zarezima razdvojenim, poredanim popisom vrijednosti za oznaku --enable-admission-plugins

komponente API poslužitelja. Za više informacija o zastavama naredbenog retka API poslužitelja, provjerite dokumentaciju kube-apiservera

7.3.4. Uvezivanje

Korisnik kreira, ili je u slučaju dinamičke opskrbe već kreirao, PersistentVolumeClaim s određenom traženom količinom pohrane i određenim načinima pristupa. Kontrolna petlja u glavnom uređaju prati nove PVC-ove, pronalazi odgovarajući PV (ako je moguće) i povezuje ih zajedno. Ako je PV dinamički osiguran za novi PVC, petlja će uvijek taj PV vezati za PVC. Inače će korisnik uvijek dobiti barem ono što je tražio, ali količina može biti veća od traženog. Jednom vezane, PersistentVolumeClaim veze su isključive, bez obzira na to kako su bile vezane. Vezanje PVC-a na PV je mapiranje jedan-na-jedan, koristeći ClaimRef koji je dvosmjerno povezivanje između PersistentVolume i PersistentVolumeClaim.

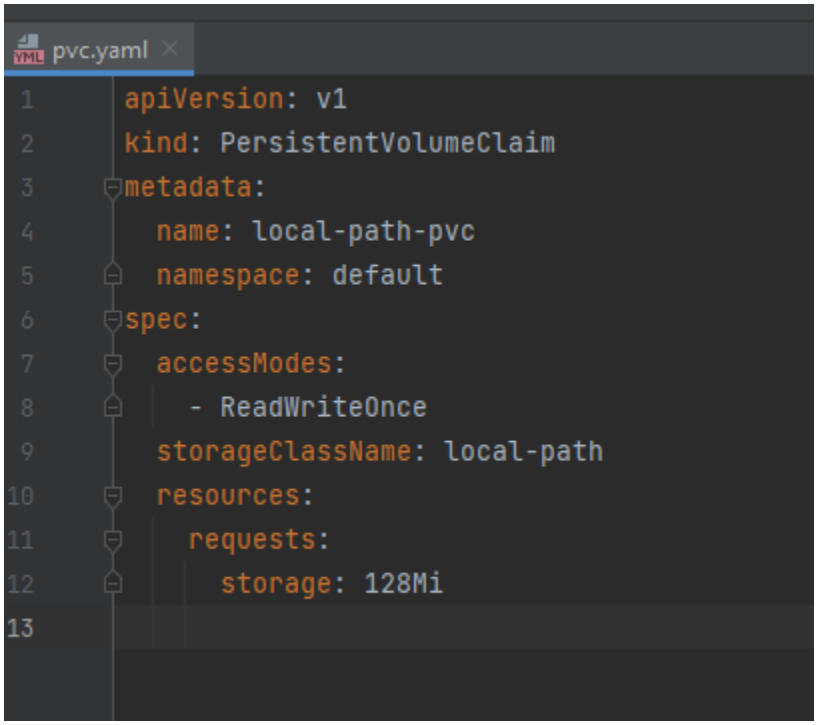
Potraživanja će ostati nevezana na neodređeno vrijeme ako ne postoji odgovarajući volumen. Zahtjevi će biti vezani čim odgovarajući volumeni postanu dostupni. Na primjer, klaster opremljen s mnogo 50Gi PV-a ne bi odgovarao PVC-u koji zahtijeva 100Gi. PVC se može vezati kada se 100Gi PV doda u klaster.

Čahure koriste zahtjeve kao volumene. Klaster provjerava zahtjev da pronađe vezani volumen i montira taj volumen za čahuru. Za volumene koji podržavaju višestruke načine pristupa, korisnik određuje koji je način željeni kada koristi svoje zahtjeve kao volumen u čahuri.

Nakon što korisnik ima zahtjev i taj zahtjev je vezan, vezani PV pripada korisniku sve dok mu je potreban. Korisnici planiraju čahure i pristupaju svojim PV-ovima za koje je položeno pravo uključivanjem odjeljka persistentVolumeClaim u blok volumena čahure.

7.4. Postavljanje dobavljača lokalne pohrane

K3s dolazi s Rancher's Local Path Provisioner i to omogućuje mogućnost stvaranja trajnih zahtjeva za volumen iz kutije koristeći lokalnu pohranu na odgovarajućem čvoru. U nastavku ćemo pokriti jednostavan primjer. Napravimo pvc.yaml



Slika47: pvc.yml

Izvor: vlastita izrada

Kreirajmo pvc naredbom:

kubectl create -f pvc.yaml

```
PS C:\Users\Marko\.kube> kubectl create -f pvc.yaml
persistentvolumeclaim/local-path-pvc created
PS C:\Users\Marko\.kube>
```

Slika48: kreiranje održivog volumena

Izvor: vlastita izrada

Kreirat ćemo testni pod.yaml

pvc.yaml pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
    - name: volume-test
      image: nginx:stable-alpine
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: vol
          mountPath: /data
      ports:
        - containerPort: 80
  volumes:
    - name: vol
      persistentVolumeClaim:
        claimName: local-path-pvc
```

Slika49: pod.yml

Izvor: vlastita izrada

Kreirajmo pvc naredbom:

kubectl create -f pod.yaml

```
PS C:\Users\Marko\.kube> kubectl create -f pod.yaml
pod/volume-test created
```

Slika50: upogonjavanj čahure

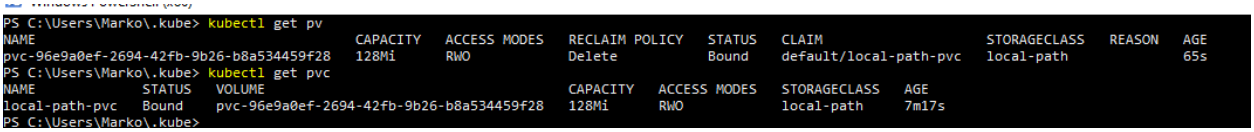
Izvor: vlastita izrada

Provjerimo PV s naredbom:

Kubectl get pv

A PVC s naredbom:

Kubectl get pvc

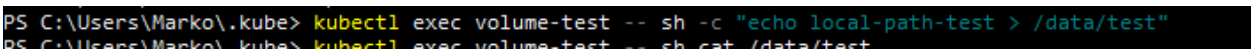


Slika51: dohvaćanje pvc-a

Izvor: vlastita izrada

Testirajmo sljedećom naredbom:

kubectl exec volume-test -- sh -c "echo local-path-test > /data/test"



Slika52: pokretanje testa

Izvor: vlastita izrada

Provjerimo da li se test datoteka nalazi na node1 sljedećim naredbama:

sudo su –

ls /var/lib/rancher/k3s/storage

cat /var/lib/rancher/k3s/storage/pvc-96e9a0ef-2694-42fb-9b26-b8a534459f28_default_local-path-pvc/test

Slika53: provjera testa

Izvor: vlastita izrada

Način na koji Local Path Provisioner radi zapravo je vrlo jednostavan. Općenito, Local Path Provisioner implementira modul kubernete-incubator/external-provisioner i dodaje implementaciju povrh njega

U našem klasteru raspoređena je jedinica lokalnog pružatelja putanje.

Pod pokreće poslužiteljski proces za pružanje usluga lokalnog puta.

Kada se prvi put pokrene, učitat će konfiguracijsku datoteku iz Kubernetes ConfigMap-a kako bi odredio put gdje se montirati u čvoru. Prema zadanim postavkama, vrijednost je /var/lib/rancher/k3s/storage.

Poslužiteljski proces će proaktivno nadzirati konfiguraciju ako dođe do novih promjena.

Poslužiteljski proces će slušati događaje Kubernetes klastera koristeći kubernetes-go klijent

7.5. Postavljanje Longhorn-a

K3s podržava Longhorn. Longhorn je open-source distribuirani sustav za pohranu blokova za Kubernetes. Postavljanje longhorn volumena ne razlikuje se puno od postavljanja lokalne pohrane. No snjim dolaze sve prednosti koje nudi longorn dashboard te ostale funkcionalnosti Longhorna

Primijenite longhorn.yaml da instalirate Longhorn:

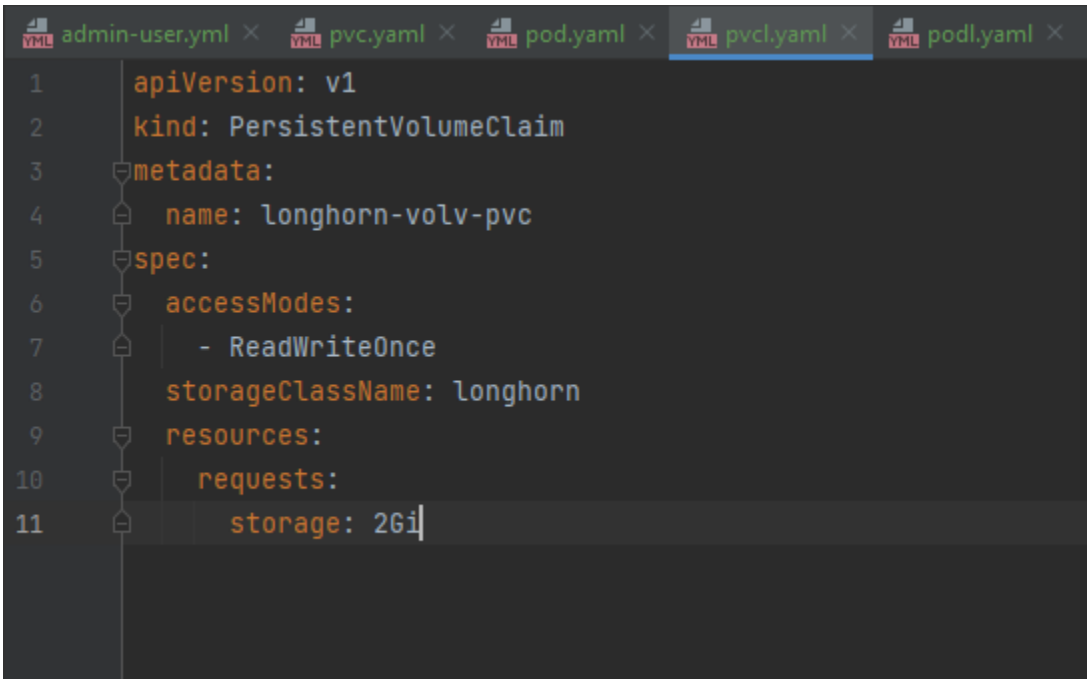
kubectl apply -f
https://raw.githubusercontent.com/longhorn/longhorn/master/deploy/longhorn.yaml

Longhorn će biti instaliran u imenskom prostoru longhorn-system.

Primijenite yaml da biste stvorili PVC i pod:

kubectl create -f pvc.yaml

kubectl create -f pod.yaml



Slika54: pvcl.yaml

Izvor: vlastita izrada

admin-user.yml | pvc.yaml | pod.yaml | pvcl.yaml | podl.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
  namespace: default
spec:
  containers:
    - name: volume-test
      image: nginx:stable-alpine
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: volv
          mountPath: /data
      ports:
        - containerPort: 80
  volumes:
    - name: volv
      persistentVolumeClaim:
        claimName: longhorn-volv-pvc
```

Slika55: podl.yaml

Izvor: vlastita izrada

kubectl get pv

kubectl get pvc

8. Prenosivost klastera

S obzirom na to da je klasteru potreban internet . Ukoliko bi htjeli prenijeti klaster na drugu lokaciju na kojoj nemamo ethernet kabla ili druge poteškoće. rpi4 dolazi s bluetooth i wifi modulom. Stoga možemo iskoristiti bluetooth u tu svrhu. Tu dolazimo do novog problema a to je DHCP. U klasteru gotovo uvijek želimo statičke adrese no s obzirom na to da je vec prikazano u poglavlju postavljanje sustava u ovom dijelu to nećemo spominjati.

Koristit ćemo bluetooth server koji će se vrtiti na rpi-ijevima, bluetooth klijent preko kojeg ćemo se spajati.

Određene bash skripte koje će nam pomoći u vezi ovisnosti koje bluetooth zahtijeva.

Prije nego krenemo trebamo staviti bluetooth poslužiteljski proces u compact mode. To možemo napraviti da dodamo zastavu -C na kraju ExecStart-a.

```
  GNU nano 5.4                                  /etc/systemd/system/bluetooth.target.wants/bluetooth.service *
[Unit]
Description=Bluetooth service
Documentation=man:bluetoothd(8)
ConditionPathIsDirectory=/sys/class/bluetooth

[Service]
Type=dbus
BusName=org.bluez
ExecStart=/usr/libexec/bluetooth/bluetoothd -C
NotifyAccess=main
#WatchdogSec=10
#Restart=on-failure
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
LimitNPROC=1
ProtectHome=true
ProtectSystem=full

[Install]
WantedBy=bluetooth.target
Alias=dbus-org.bluez.service
```

Slika56: konfiguracija compact moda

Izvor: vlastita izrada

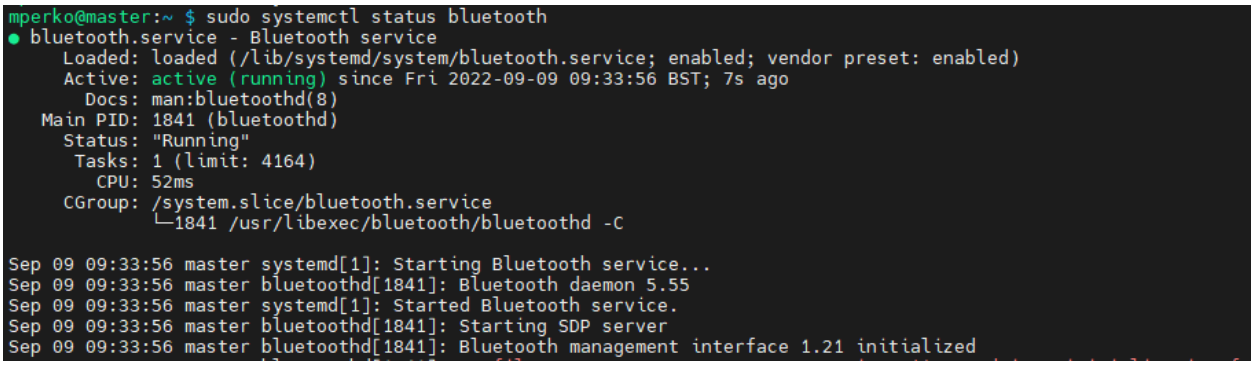
Te nakon izmjene trebamo ponovo pokrenuti poslužiteljski proces te samu bluetooth uslugu.

To možemo učiniti sljedečim naredbama:

Sudo systemctl daemon-reload

Te

Sudo systemctl restart bluetooth



Slika57: provjera bluetooth usluge

Izvor: vlastita izrada

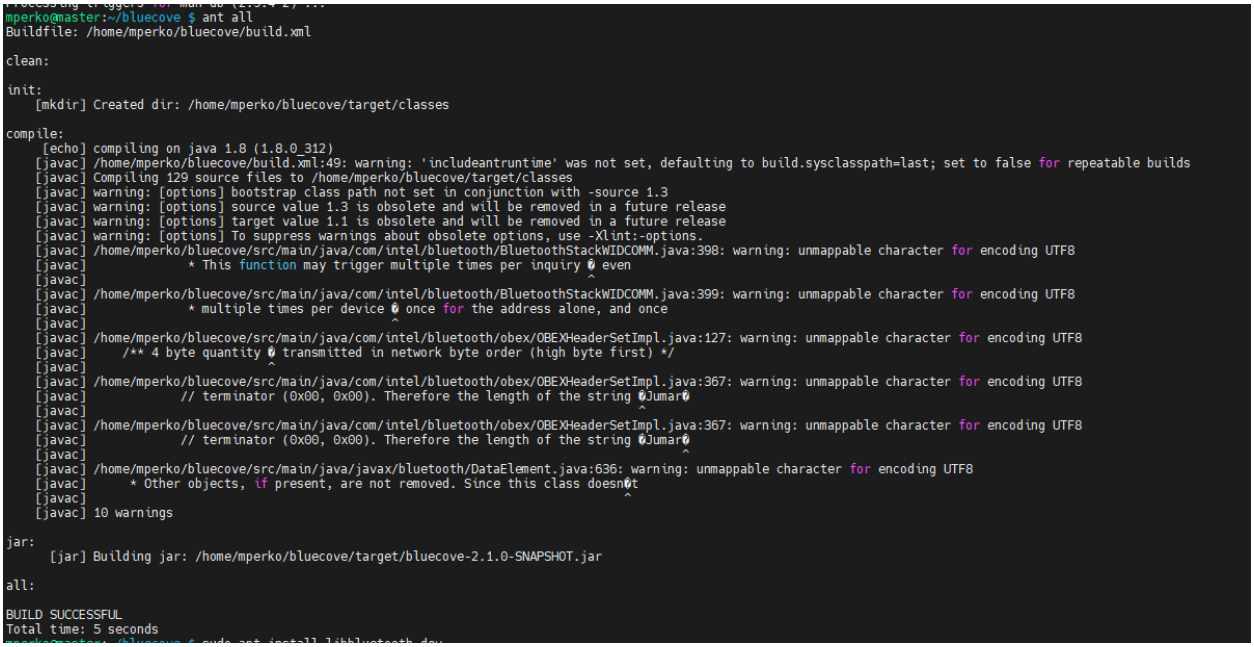
Također koristit ćemo blucove i blucove-gpl biblietoke i libbluetooth_aarch64.so zajednički objekt. Najbolje je da navedene ovisnosti izgradimo na sustavu na kojem ćemo ih koristiti kako bi izbjegli moguće poteškoće s arhitekturom. Također potreban nam je ant i libbluetooth-dev biblioteka. Koje možemo instalirati putem sljedeće naredbe:

Sudo apt install ant libbluetooth-dev

Za izgradnju je dovoljno da se pozicioniramo u bluecove datoteku i pokrenemo naredbu:

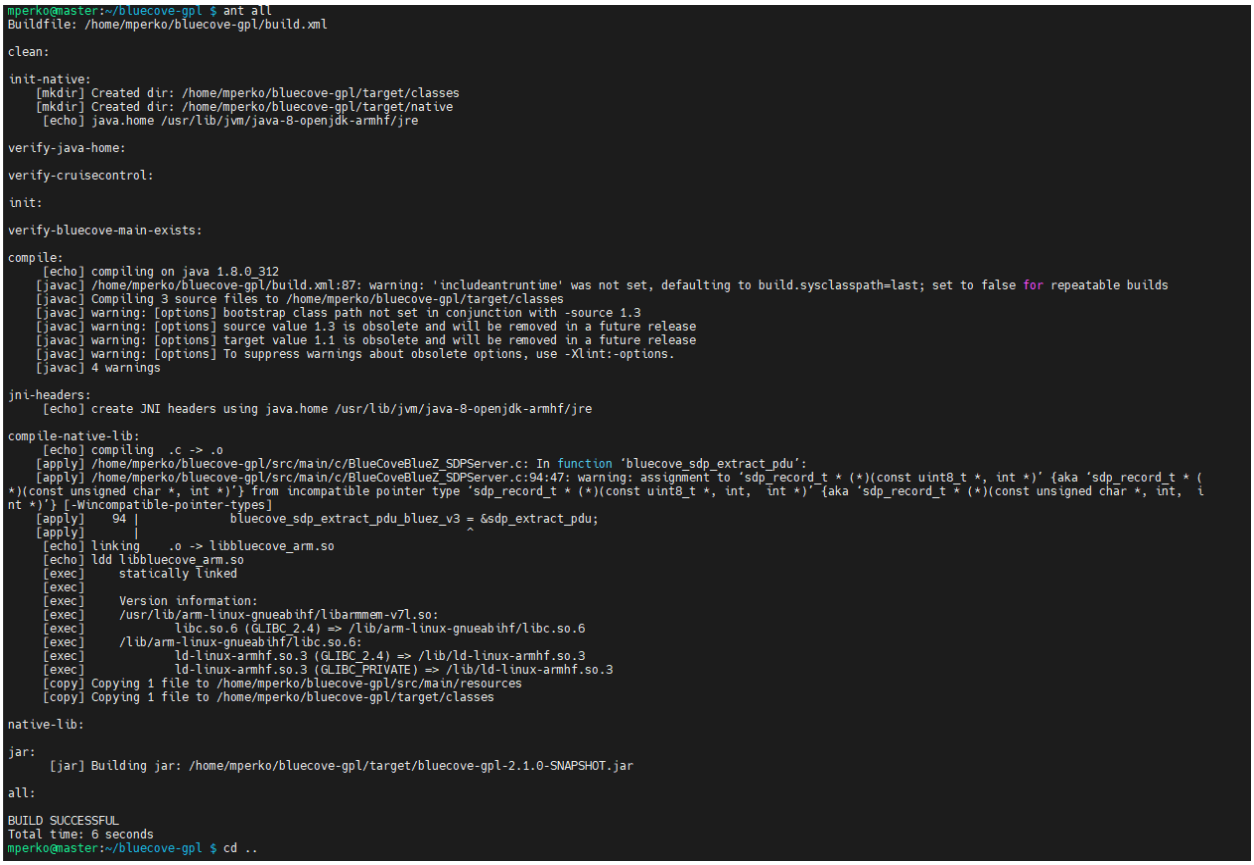
Ant all

Te nakon toga se pozicinirамo u bluecove-gpl datoteku te isto naredbom izgradio bluecove-gpl datoteku. Bluecove-gpl ovisi o bluecove biblioteci tako da moramo uvijek prvo izgraditi bluecove te tek onda ostale module bluecove-a.



Slika58: bluecove izgradnja

Izvor: vlastita izrada

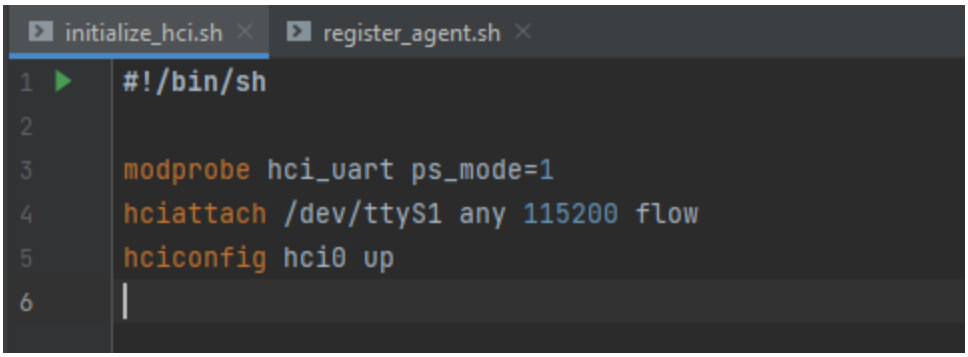


Slika59: bluecove-gpl izgradnja

Izvor: vlastita izrada

8.1. Pomočne skripte

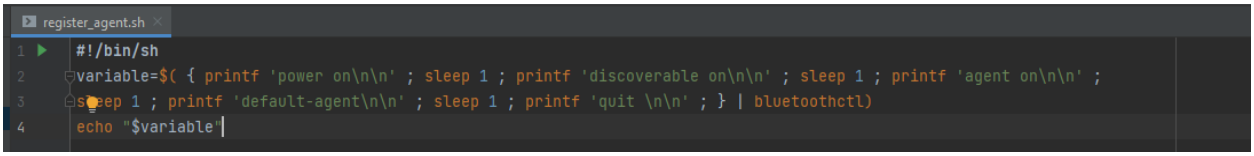
initialize_hci.sh skripta na služi za inicijalizaciju hci-ija odnosno bluetooth device-a. Prva naredba na dodaje kernel modul u ps modu. Druga naredba je zapravo konfiguracija u kojoj govorimo na kojem portu će biti device te koji mu je baud rate. Te na samom kraju inicijaliziramo sami device.



Slika60: bash skripta

Izvor: vlastita izrada

Druga skripta koju ćemo koristiti priprema piše u bluetoothctl komandnu liniju naredbe za upaliti device, napraviti mogućim za otkrivanje, te staviti na device zadanog agenta.



Slika61: bash skripta

Izvor: vlastita izrada

Navedene skripte ćemo pozivati kroz java kod.

8.2. Bluetooth server

Krenut ćemo od zavisnosti koje su nam potrebne. Sve zavisnosti su osobni odabir. Postoji niz alternativnih biblioteka.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.bluetooth.*;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import java.io.*;
import java.net.InetAddress;
import java.net.NetworkInterface;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.stream.Collectors;
```

Slika62: java kod

Izvor: vlastita izrada

U samoj main metodi ćemo inicijalizirati server. Sljedeće tri metode zapravo konfiguriraju sve što je potrebno za normalan rad servera.

Te ćemo na kraju pokrenuti server.

```
public static void main(String[] args) throws IOException {
    SimpleServer simpleServer = new SimpleServer();



    simpleServer.startupCommands();
    simpleServer.runCommand("hciconfig hci0 name k3s-portability");
    simpleServer.runAgentAndCheckChanges();
    simpleServer.bluetoothStartup();

}
```

Slika63: java kod

Izvor: vlastita izrada

```
public void startupCommands() throws IOException {
    runCommand("ln -s libbluetooth.so.3 /usr/lib/libbluetooth.so");
    runCommand("modprobe hci_uart ps_mode=1");
    runCommand("hciattach /dev/ttyS1 any 115200 flow");
    runCommand("hciconfig hci0 up");
    runCommand("hcitool -i hci0 cmd 0x08 0x0008 1E 02 01 06 0C 09 5A 69 70 61 62 6F 78 32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00");
//      runCommand("hcitool -i hci0 cmd 0x08 0x0009 1E 02 01 1A 1A 16 0F 0F " HEX ZA ADVERTISE  "00 00 00 00 00 00");
    runCommand("hciconfig hci0 leadv 0");
}
```

Slika64: java kod

Izvor: vlastita izrada

```
public void runCommandAndSaveSsid() throws IOException {
    Process process = Runtime.getRuntime().exec( command: "iwgetid wlan0 --raw");
    saveSsid(process);
}

public void runAgentAndCheckChanges() throws IOException {
    Process process = Runtime.getRuntime().exec( command: "bash /etc/init.d/register_agent.sh");
    BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
    String line;
    while ((line = reader.readLine()) != null) {
        LOG.info(line);
        if (line.contains("Default agent request successful")) {
            LOG.info("agent success");
            return;
        }
    }
    runAgentAndCheckChanges();
}
```

Slika65: java kod

Izvor: vlastita izrada

U prvom dijelu inicijaliziramo device. Te taj device postavljamo u otkriveni mod putem DiscoveryAgent.GIAC agenta.

```
public void bluetoothStartup() throws IOException {
    LocalDevice local = LocalDevice.getLocalDevice();
    System.out.println("Device name: " + local.getFriendlyName());
    System.out.println("Bluetooth Address: " +
            local.getBluetoothAddress());
    boolean res = local.setDiscoverable(DiscoveryAgent.GIAC);
    System.out.println("Discoverability set: " + res);


    UUID uuid = new UUID( uuidValue: "1101",  shortUUID: true);
    String connectionString = "btspp://localhost:" + uuid + ";name=Sample SPP Server";

    StreamConnectionNotifier streamConnNotifier = (StreamConnectionNotifier) Connector.open(connectionString);


    System.out.println("\nServer Started. Waiting for clients to connect...");
    StreamConnection connection = streamConnNotifier.acceptAndOpen();

    RemoteDevice dev = RemoteDevice.getRemoteDevice(connection);
    System.out.println("Remote device address: " + dev.getBluetoothAddress());
    System.out.println("Remote device name: " + dev.getFriendlyName( alwaysAsk: true));
    running = true;
    InputStream inStream = connection.openInputStream();
    OutputStream outStream = connection.openOutputStream();
    Gson gson = new GsonBuilder().disableHtmlEscaping().serializeNulls().create();
    boolean alreadyLogged = false;
    while (running) {
        String received = null;
```

Slika66: java kod

Izvor: vlastita izrada

Nakon toga kreiramo UUID „1101“ jer je to službeni generički UUID za SPP uslugu. Pomoću navedenog UUID-a kreiramo niz veze. Te pomoću tog niza otvorimo vezu.

Te devicu predamo niz veze. Inicijaliziramo zastavu čijom promjenom ćemo ugasiti server. Inicijaliziramo ulazni i izlazni tok za primanje i slanje poruka.

Na kraju inicijaliziramo json graditelja s dodatnim parametrima kake bi izbjegli poteškoće u parsiranju poruka.

```
while (running) {
    String received = null;
    try {
        int bytes_read = inStream.read(buffer);
        if (bytes_read == -1) {
            connection.close();
            connection = streamConnNotifier.acceptAndOpen();
            dev = RemoteDevice.getRemoteDevice(connection);
            inStream = connection.openInputStream();
            outStream = connection.openOutputStream();
            continue;
        }
        received = new String(buffer,  offset: 0, bytes_read);
    } catch (StringIndexOutOfBoundsException e) {
        if (!alreadyLogged) {
            alreadyLogged = true;
            LOG.info("Nothing to read");
            LOG.info("Please check your client connection");
        }
    }

    LOG.info("Message: " + received + " From: "

            + dev.getBluetoothAddress());
```

Slika67: java kod

Izvor: vlastita izrada

```
        if (isJSONValid(received)) {
            JsonObject jsonObject = JsonParser.parseString(received).getAsJsonObject();
            String action = jsonObject.get("action").getAsString();
            System.out.println(action);
            switch (action) {
                case "GET_SSID_LIST":
                    System.out.println("starting scan");
                    scanResponse();
                    String listJson = gson.toJson(getList());
                    outStream.write(listJson.getBytes(StandardCharsets.UTF_8));
                    outStream.write("\r\n".getBytes(StandardCharsets.UTF_8));
                    System.out.println(listJson);
                    break;
                case "GET_CURRENT_SSID":
                    if (!getIPAddress( useIPv4: true,  type: "wlan0").equals("ERROR")) {
                        runCommandAndSaveSsid();
                        HashMap<String, Object> currentSsid = new HashMap<>();
                        System.out.println(getSsid());
                        currentSsid.put("ssid", getSsid());
                        String connectedSsid = gson.toJson(currentSsid);
                        System.out.println(connectedSsid);
                        outStream.write(connectedSsid.getBytes(StandardCharsets.UTF_8));
                    } else {
                        HashMap<String, Object> nullSsid = new HashMap<>();
                        nullSsid.put("ssid", null);
                        String returnNull = gson.toJson(nullSsid);
                        System.out.println(returnNull);
                        outStream.write(returnNull.getBytes(StandardCharsets.UTF_8));
                    }
                    outStream.write("\r\n".getBytes(StandardCharsets.UTF_8));
                    break;
                case "CONNECT_SSID":
```

lika69: java kod

Izvor: vlastita izrada

```
                case "CLOSE_CONNECTION":
                    connection.close();
                    break;
            }
        }
    }
}
```

lika69: java kod

Izvor: vlastita izrada

Zbog nepreglednosti opisat ćemo samo najvažnije funkcionalnosti ostatak koda bit će dostupan na github repozitoriju na sljedećem linku:

https://github.com/MarkoPerko/k3s

9. Zaključak

Za upravljanje upogonjavanjima, krajnji korisnici moraju komunicirati s Kubernetes API-jem. Uvođenju Kubernetesa pristupa se putem alata naredbenog retka kubectl (ili putem rješenja treće strane). Koristeći sučelje kubectl, korisnici mogu kreirati nova upogonjavanja, pregledati postojeća upogonjavanja i prekinuti stare module dok stvaraju nove replike modula koji će zauzeti njihova mjesta. A putem k3s sve je to moguće na mnogo manje zahtjevan način u smislu računalnih resursa.

Korisnici mogu primijeniti Kubernetes ažuriranja, a zatim dopustiti automatizaciju kako bi se osiguralo kontinuirano pridržavanje novih željenih stanja. Nakon što se napravi promjena u predlošku specifikacije čahure,

Korisnik postavlja zahtjeve koristeći Kubernetes upogonjavanja, a Kubernetes preuzima kako bi osigurao da moduli ispunjavaju te nove zahtjeve, implementirajući sve potrebne promjene.

Kubernetes pruža značajne prednosti, ali s tim prednostima dolaze i izazovi. Konkretno, kontejnerizacija stvara gotovo neograničene nove izvore napada putem kojih može doći do prijetnji gdje treće strane potencijalno mogu dobiti pristup mreži.

10. Popis literature

1. https://kubernetes.io

2. https://rancher.com

3. http://www.zemris.fer.hr/~sgros/stuff/rjecnik.shtml

4. https://carpie.net/articles

5. https://blog.gigamon.com/2021/09/30/kubernetes-deployment/

6. https://github.com/rancher

7. https://helm.sh

8. https://www.docker.com

9. https://www.bmc.com/blogs/kubernetes-daemonset/#:~:text=Deleting%20a%20DaemonSet%20is%20a,DaemonSet%20without%20deleting%20the%20pods.

11. Popis slika

Slika69: java kod 68