

DOCKER WORKSHOP

En introduksjon til bruk av Docker.

Bouvet backend gruppa, høsten 2023.



bouvet



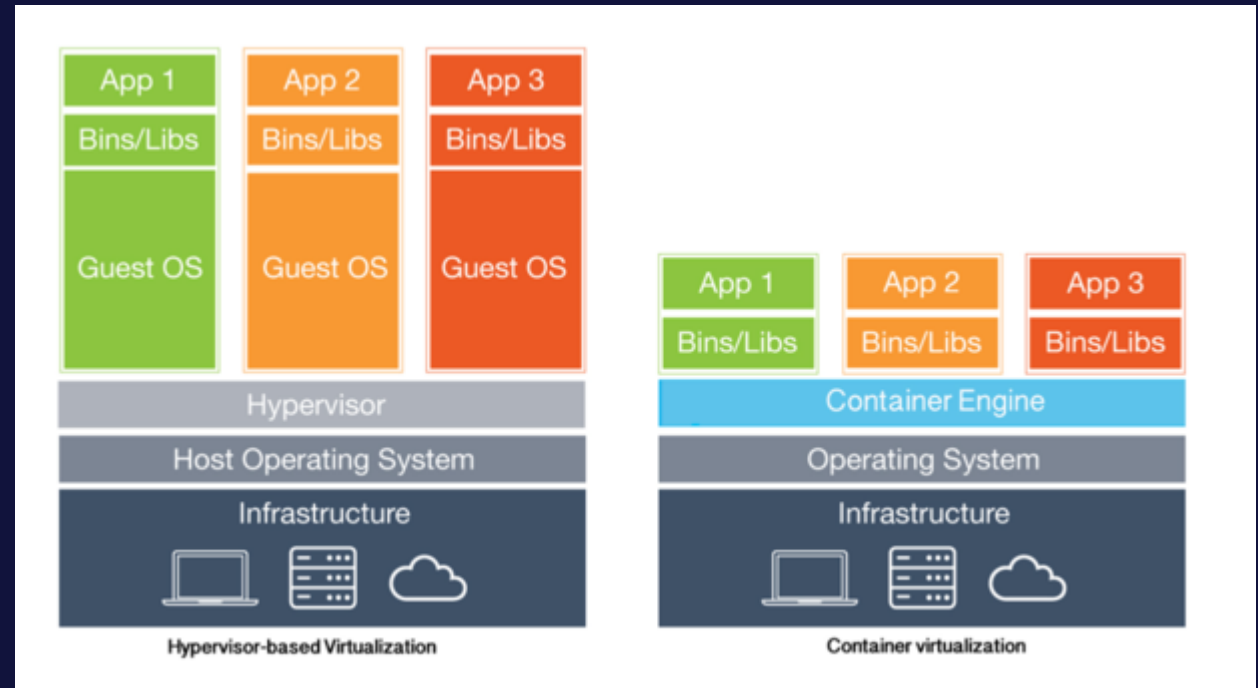
Hva er Docker?

Docker er en plattform for applikasjoner.

Docker er en teknologi som gir oss muligheten til å pakke inn programmer i små containere som deler på maskinvare og operativ systemer.

Fordeler:

- Isolering av applikasjon
- Betydelig bedre sikkerhet
- Dele på maskinvare / nettverk etc.
- Benytte ferdige imager å bygge videre på
- Flyttbart mellom datasystemer.
- Fleksibel for legacy systemer etc.
- Automatisere utrulling av applikasjoner.
- Lagre imager i sentrale lagre.
- Lett å bygge og slette kontainere.
- Kort oppstartstid for containere.
- Mulighet for bruk i kubernetes etc.



Hvorfor bruke Docker?

Å paketere programmer, tjenester, databaser etc i små Kontainere, gjør at man kan utnytte maskinvaren mye bedre enn ved å ha separate maskiner eller Virtuelle maskiner for hver applikasjon.

Slike kontainere kan dele på samme maskin eller kjøres i et Cluster av maskiner som f.eks under kubernetes.

Da kan man flytte kontainere mellom noder ette behov, eller kjøre flere kontainere på samme maskin, eller spredt utover flere noder for last balansering eller feil håndtering.

En stor fordel er at man kan benytte seg av forhånds lagde Kontainer imager med os, databaser etc, som man selv kan Bygge videre på. Da slipper man å vedlikeholde os og annen Infrastruktur avhengigheter selv.



Hva er forskjellen på VM og kontainer?

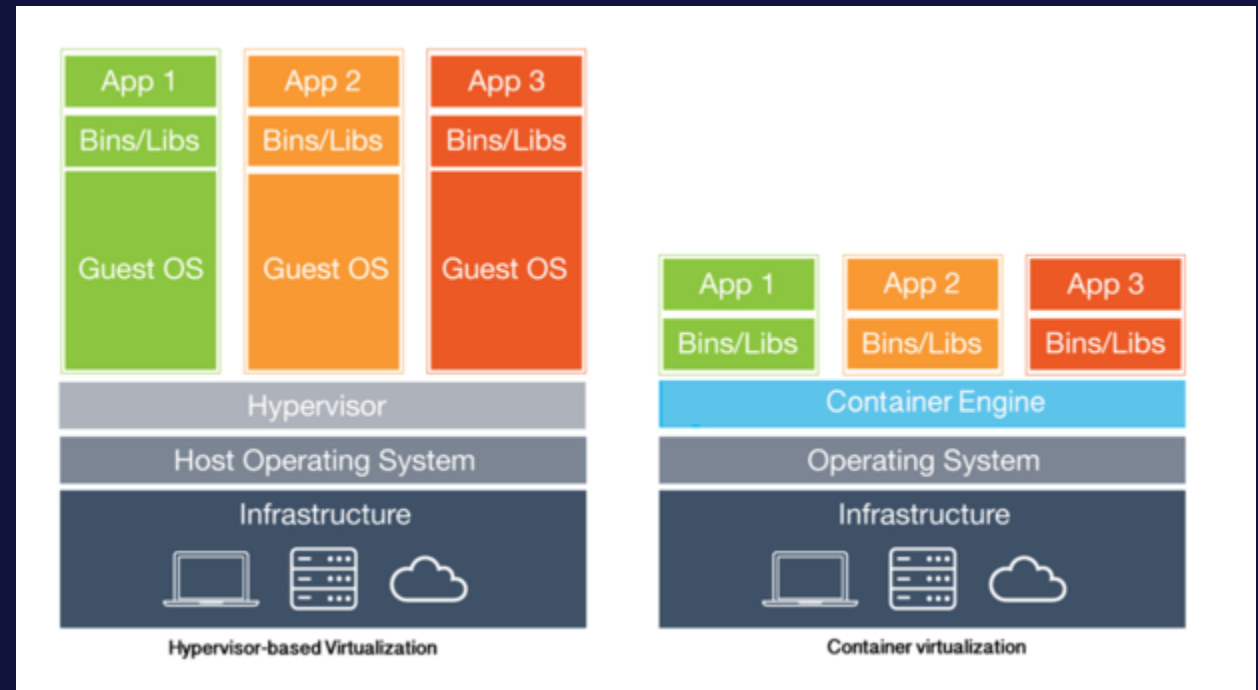
Kontainer(e) deler på maskinvare, Operativsystem og selve kontainer systemet.

Virtuelle maskiner deler på maskinvaren, men Bringer selv operativ systemer, nødvendige Komponenter og applikasjonen.

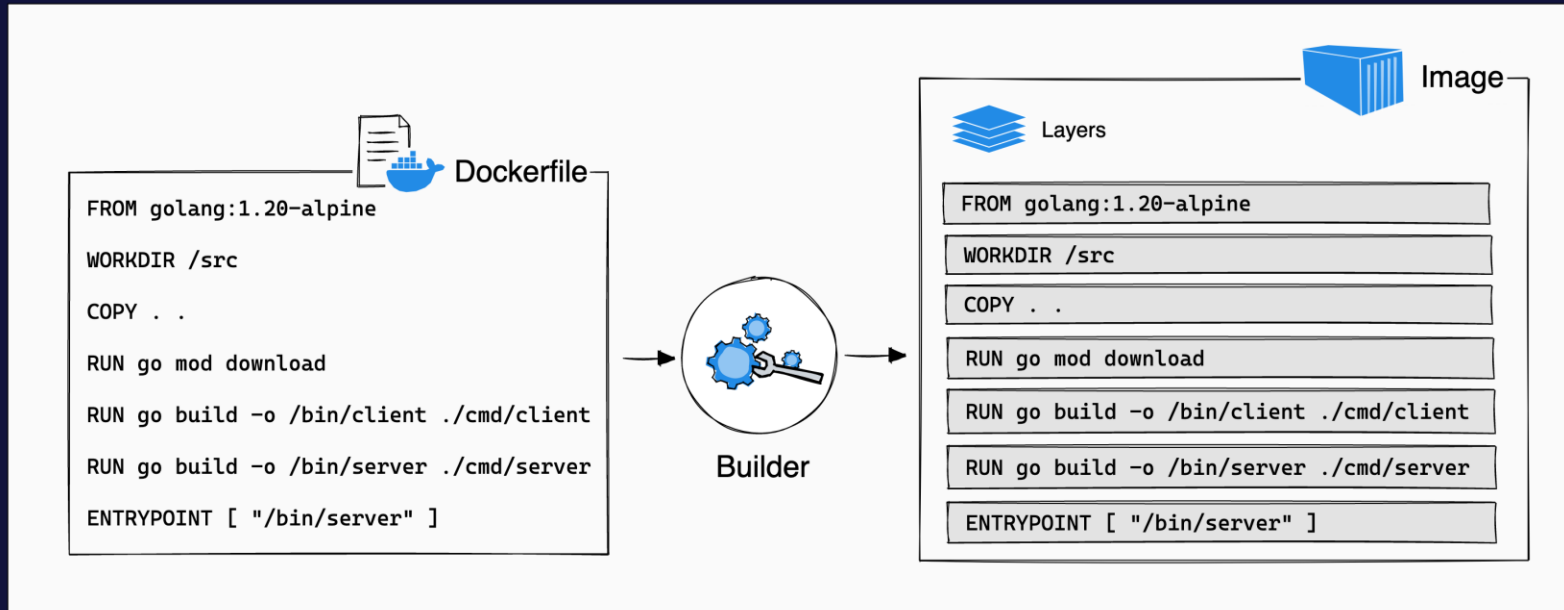
Virtuelle maskiner trenger vesentlig mere Maskinvare som minne og lagring. Krever mye mere administrasjon og vedlikehold og starter tregere.

Kontainere er ofte svært små og bygger på andre imager for nødvendige støtte systemer og deler på operativsystemet og maskinvaren i bunn.

Kontainere er lette og raske å starte og å fjerne. Uten bruk av Volumes lagrer de ikke data etter at de stoppes / slettes.



Anatomien til en Docker kontainer



Kontainere er lagvis laget. Man starter med en grunn image som f.eks alpine linux, eller en som også inneholder en spesialisering for dot.net e.l.

På denne bygger man lagvis på sine egne avhengigheter og til slutt selve applikasjonen eller tjenesten e.l.

Trenger man lagring utover levetiden til kontaineren, så må man sette opp et volume som lagrer data også etter at kontaineren er stoppet.

Installere Docker

På en Linux maskin eller under WSL på Windows

- `sudo apt-get update`
- `sudo apt-get install ca-certificates curl gnupg`
- `sudo install -m 0755 -d /etc/apt/keyrings`
- `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg`
- `sudo chmod a+r /etc/apt/keyrings/docker.gpg`
- `echo \`
- `"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \`
- `"$(. /etc/os-release && echo "$VERSION_CODENAME)" stable" | \`
- `sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`
- `sudo apt-get update`
- `sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
- `sudo docker run hello-world`

Installere podman / podman desktop

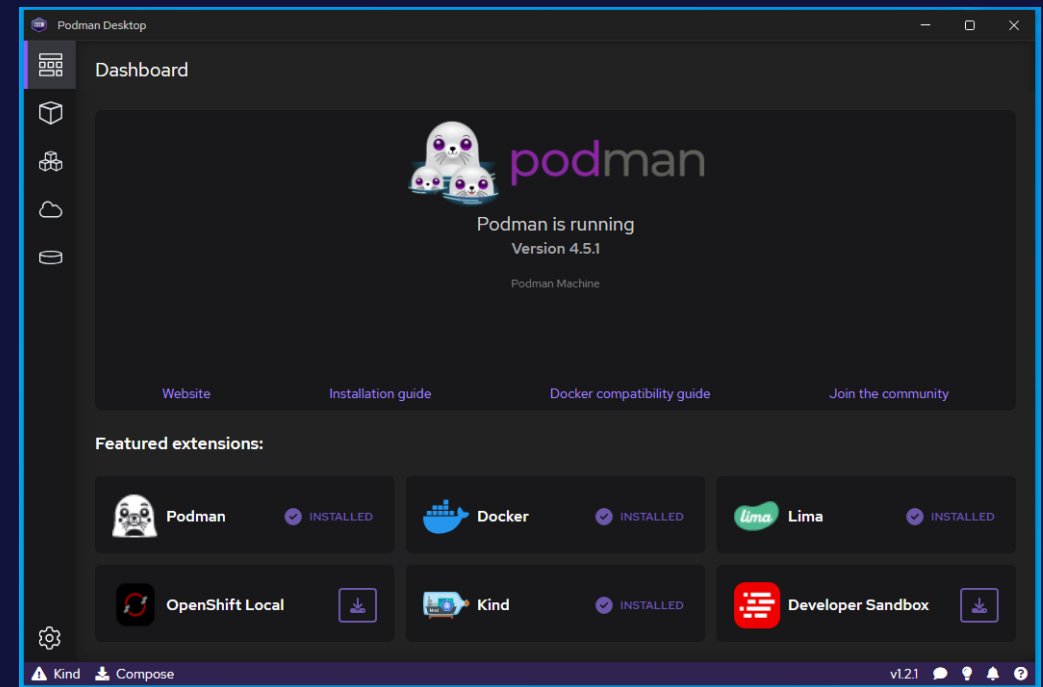
En opensource alternativ til Docker

Last ned Podman Desktop for Windows fra:

<https://podman-desktop.io/downloads/windows>

For å installere under WSL eller på Linux maskiner, kan Man enkelt skrive i wsl / bash:

```
sudo apt -y install podman
```



Enkel image script

Et enkelt Python basert eksempel

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route("/")  
def hello():  
    return "Hello World!"
```

```
FROM python:3-alpine  
RUN python -m pip install --upgrade pip  
RUN pip install flask==2.1.*
```

```
# install app  
COPY hello.py /
```

```
# final configuration  
ENV FLASK_APP=hello  
EXPOSE 8000  
CMD flask run --host 0.0.0.0 --port 8000
```


Kjøre kontainere basert på egne imager

- `podman build . --tag hello_bouvét`
- `podman run --rm -p 127.0.0.1:8000:8000 hello_bouvét`
- Videre kan man pushe imager til repositories lokalt eller f.eks dockerhub for distribusjon til andre servere e.l.
- Åpne nettleser og naviger til: <http://127.0.0.1:8000/>

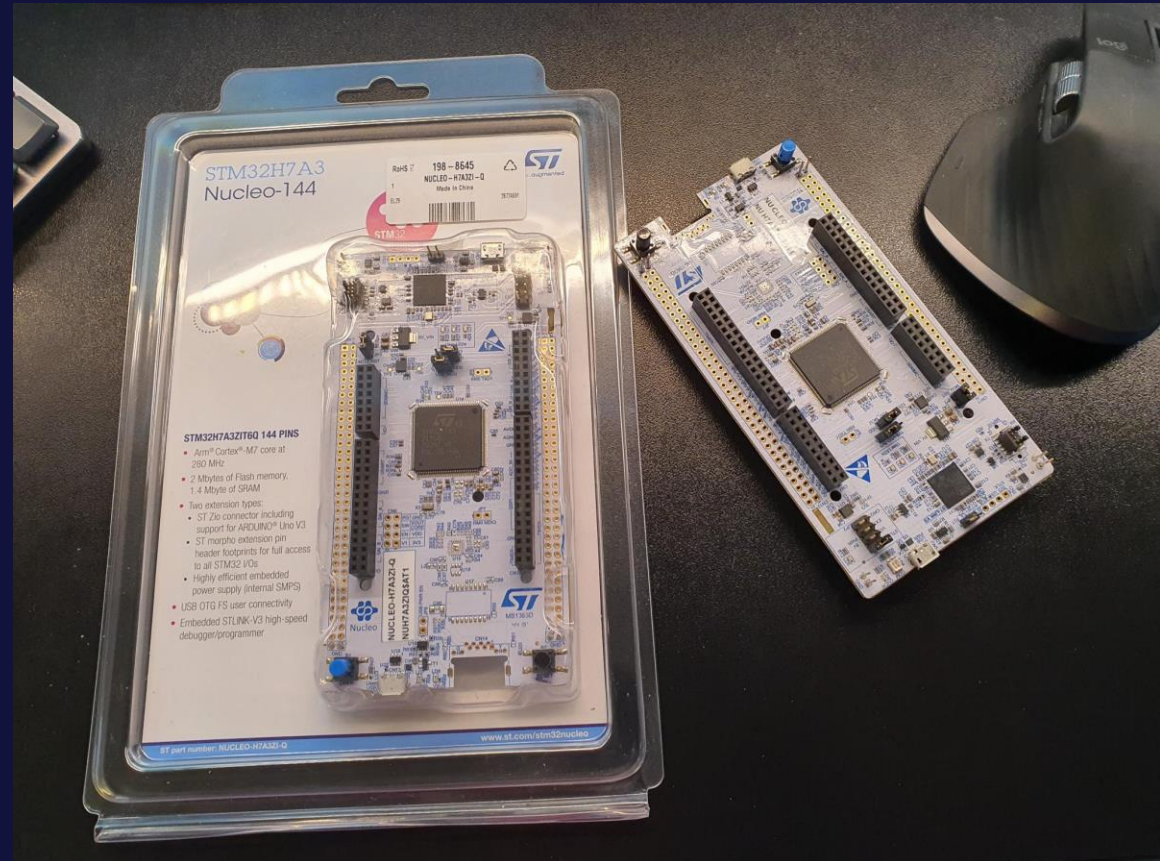
Oppgave: Lag en enkel kontainer

- Start med å installere docker / podman enten lokalt eller på sky tjeneste.
- Forsikre deg om at det virker ved å kjøre 'hello-world' kontainer.
- Lag det lille Python scriptet og benytt docker script til å bygge din egen kontainer.
- Start opp kontaineren. Sjekk at den kjører, eks: 'docker container ls'
- Prøv å gå til <http://127.0.0.1:8000/>

Utviklermiljø i en Konteiner

Utvikle for mikro kontrollere fra din egen datamaskin.

- Hvordan utvikle for STM32
 - Egen PC med kryss kompilering.
 - Raspberry PI e.l. med kompilering
 - Skreddersydd konteiner?



Utviklermiljø i en Konteiner

Dockerfile

- FROM ubuntu:latest
 - # Download Linux support tools
- RUN apt-get update && \
- apt-get clean && \
- apt-get install -y \
- build-essential \
- wget \
- curl \
- nano
- # Set up a development tools directory
- WORKDIR /home/dev
- ADD dev /home/dev
- RUN wget -qO- https://developer.arm.com/-/media/Files/downloads/gnu-rm/10.3-2021.10/gcc-arm-none-eabi-10.3-2021.10-x86_64-linux.tar.bz2 | tar -xj
- ENV PATH \$PATH:/home/dev/gcc-arm-none-eabi-10.3-2021.10/bin
- WORKDIR /home/app

Utviklermiljø i en Konteiner

Bygge og kjøre vår kryss kompilerings Konteiner

- For å bygge Konteineren:
 - `podman build -t stenbror/gcc-arm .`
- For å kjøre konteineren i interaktiv modus:
 - `podman run --rm -it --privileged -v "$(PWD):/home/app" stenbror/gcc-arm:latest bash`

Utviklermiljø i en Konteiner

Følgende skjer i konteineren og til slutt utenfor i PS.

- Start med å gå inn i bygge mappen:
 - `cd /home/app/dev`
- Sjekk om du har test.c og build.sh filene
 - `ls -l`
- Kjør bygge skriptet:
 - `./build.sh`
- Se om du har en kompilert fil både inni konteineren og utenfor.
 - `ls -l (Konteiner) / dir (Windows maskinen i powershell / Command Line)`

Podman kommandoer

- Usage:
- `podman pod [command]`
- Available Commands:
- `create` Create a new empty pod
- `exists` Check if a pod exists in local storage
- `inspect` Displays a pod configuration
- `kill` Send the specified signal or SIGKILL to containers in pod
- `logs` Fetch logs for pod with one or more containers
- `pause` Pause one or more pods
- `prune` Remove all stopped pods and their containers
- `ps` List pods
- `restart` Restart one or more pods
- `rm` Remove one or more pods
- `start` Start one or more pods
- `stats` Display a live stream of resource usage statistics for the containers in one or more pods
- `stop` Stop one or more pods
- `top` Display the running processes of containers in a pod
- `unpause` Unpause one or more pods

Podman kommandoer

- Kjør 'podman --help' som lister kommandoer og hva de gjør.
- Build => Bygger image fra Dockerfile «oppskrift»
- Run => Kjører en kommando i en ny konteiner. Eks. /bin/bash
- Top => Lister alle kjørende prosesser i en konteiner.
- Start => Starte en eller flere konteinere.
- Stop => stoppe en eller flere konteinere.
- 'Podman system info' => Detaljer om podman miljøet.

Podman kommandoer

Styring av Volumer.

```
Ubuntu-Linux LTS
Manage volumes

Description:
  Volumes are created in and can be shared between containers

Usage:
  podman volume [command]

Available Commands:
  create      Create a new volume
  exists      volume exists
  export      Export volumes
  import      Import a tarball contents into a podman volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused volumes
  rm          Remove one or more volumes

stenbror@PF2RLFT0:~$ podman volume ls
DRIVER      VOLUME NAME
local       531f04ed615a6e31e9a816673d868167725384bfecc234aa9365d0e20e5c0b4e
local       74e00e838b4ed6446d940e465a0ee8c209864101ab2631c4232a6eb71d26f2df
stenbror@PF2RLFT0:~$
```

Volume er den foretrukne måten å ta vare på data generert og brukt av podman / docker konteinere.

Det er her man lagrer data som skal overleve levetiden til den enkelte konteiner, som kan startes, stoppes, slettes og skapes etter behov.

Lage et nytt tomt volum:

- docker volume create my-vol

Liste volumer:

- docker volume ls

Fjerne et volum:

- docker volume rm my-vol

Podman – Nettverk

- For å styre nettverkstrafikk inn og ut av en konteiner, setter man opp en port mapping.
 - `-p 8080:80`
 - 8080 er porten som er tilgjengelig hos «host maskinen» som mappes til konteinerens port 80
 - `-p 8080:80/udp` Setter opp mapping fra «Host» 8080 porten til konteiner port 80 for pakker av typen udp.
 - `-p 8080:80/tcp -p 8080:80/udp`
 - Setter opp flere mapper samtidig.
 - `-p 192.168.1.100:8080:80`
 - Mapper ip/port på host til port 80 i konteiner.

Push / Pull imager

- Man kan både «push» imager lokalt og til sentrale servere, samt «pull» fra det samme.
 - 'podman push dir://tmp/myImage' (Lokalt lagret image)
 - 'podman pull dir://tmp/myImage' (Hente lokalt lagret image)
- Man kan også lagre / hente fra sentrale 'registry' og lokale 'registry' og kan benyttes av kubernetes etc til å distribuere konteinere på flere servere fra lagrede imager.
- Et senere Workshop vil ta for seg Kubernetes.

Proxmox med Kubernetes med docker:

PROXMOX Virtual Environment 8.0.4

Documentation Create VM Create CT root@pam

Server View

Datacenter (cats)

- pve-node1
 - 103 (k8s-worker-node-3)
 - localnetwork (pve-node1)
 - local (pve-node1)
 - local-lvm (pve-node1)
- pve-node2
 - 102 (k8s-worker-node-2)
 - localnetwork (pve-node2)
 - local (pve-node2)
 - local-lvm (pve-node2)
- pve-node3
 - 101 (k8s-worker-node-1)
 - localnetwork (pve-node3)
 - local (pve-node3)
 - local-lvm (pve-node3)
- pve-node4
 - 100 (k8s-master-node)
 - localnetwork (pve-node4)
 - local (pve-node4)
 - local-lvm (pve-node4)

Datacenter

Search

Summary Notes Cluster Ceph Options Storage Backup Replication Permissions Users API Tokens Two Factor Groups Pools Roles Realms HA ACME Firewall Metric Server Resource Mappings Support

Type	Description	Disk usage...	Memory us...	CPU usage	Uptime	Host CPU ...	Host Mem...	Tags
node	pve-node1	10.7 %	68.3 %	10.1% of 4 ...	10 days 05:0...			
node	pve-node2	12.0 %	67.0 %	10.8% of 4 ...	10 days 05:0...			
node	pve-node3	7.4 %	66.7 %	11.6% of 4 ...	10 days 05:0...			
node	pve-node4	6.8 %	64.0 %	24.9% of 4 ...	10 days 05:0...			
qemu	103 (k8s-worker-node-3)	0.0 %	85.0 %	6.7% of 4 ...	10 days 05:0...	6.7% of 4C...	44.8 %	
qemu	102 (k8s-worker-node-2)	0.0 %	86.5 %	7.0% of 4 ...	10 days 05:0...	7.0% of 4C...	45.5 %	
qemu	101 (k8s-worker-node-1)	0.0 %	87.6 %	7.7% of 4 ...	10 days 05:0...	7.7% of 4C...	46.1 %	
qemu	100 (k8s-master-node)	0.0 %	91.3 %	21.2% of 4 ...	10 days 05:0...	21.2% of 4...	48.1 %	
sdn	localnetwork (pve-node1)							
sdn	localnetwork (pve-node2)							
sdn	localnetwork (pve-node3)							
sdn	localnetwork (pve-node4)							
storage	local (pve-node1)	10.7 %						
storage	local-lvm (pve-node1)	17.8 %						
storage	local (pve-node2)	12.0 %						
storage	local-lvm (pve-node2)	5.4 %						
storage	local (pve-node3)	7.4 %						
storage	local-lvm (pve-node3)	5.8 %						
storage	local (pve-node4)	6.8 %						
storage	local-lvm (pve-node4)	17.6 %						

Mere informasjon på nettet:

- <https://docs.podman.io/en/latest>
- <https://docs.docker.com>
- <https://kubernetes.io/docs/home/>
- <https://linuxcontainers.org>
- <https://www.proxmox.com/en/>