

# Vedlegg A - Systemkonfigurasjon

---

## Innhold:

1. [Generelt](#)
2. [GitHub](#)
3. [Basiskonfigurasjon](#)
  - 3.1. [Docker](#)
  - 3.2. [minikube](#)
4. [K8s konfigurasjon](#)
  - 4.1. [django-applikasjon](#)
  - 4.2. [mysql](#)
  - 4.3. [sched-applikasjon](#)

## 1 Generelt

Dette vedlegget tar for seg hvordan maskinene ble konfigurert til å gjennomføre testene.

I prosjektet ble tre like maskiner benyttet. Disse ble utdelt av Cyberingeniørskolen og modellen er en ThinkBook 14 G2 ITL. Alle maskinene har samme maskinvare, men det er forekomster av ulike versjoner med tanke på OS og programmer. Maskinvaren til maskinene er:

- 16GB RAM
- 512GB SSD
- 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz prosessor

For enkelthetsskyld blir maskinene navngitt hhv. A, B og C. Maskin A ble benyttet til test én, maskin B ble benyttet til test to og maskin C ble benyttet til test tre. Maskinene ble tanket med unix-basert operativsystem. Operativsystemet på maskin A og C er Ubuntu og maskin B er Pop! OS. [Tabell 1](#) viser hvilke versjoner av operativsystemer og programvarer som er installert på maskinene.

**Tabell 1: Oversikt maskiner**

Maskin	OS-versjon	Docker-versjon	minikube-versjon	kubectl-versjon
A	Ubuntu 20.04.4 LTS	20.10.14	v1.25.2	v1.23.3
B	Pop!_OS 21.10	20.10.12	v1.25.2	v1.23.3
C	Ubuntu 20.04.4 LTS	20.10.14	v1.25.2	v1.23.3

## 2 GitHub

Git ble benyttet til versjonskontroll av applikasjonen som er laget og et fellesområde hvor filer tilknyttet dette prosjektet er blitt lagret. Alle konfigurasjonsfiler, applikasjoner og vedlegg ligger i GitHub. Alle repoene er samlet i en organisasjon som finnes på denne lenken: <https://github.com/CISK-2022-bachelorgruppe>

Organisasjonen inneholder tre repoer. Dette er [applikasjoner](#), [kubernetes-config](#) og [vedlegg](#).

## 3 Basiskonfigurasjon

Dette kapitlet tar for seg basiskonfigurasjon på maskinene som ble benyttet i prosjektet. Dette er for å kunne lage et tilnærmet likt labmiljø som er benyttet i dette prosjektet, og for å kunne ha likt utgangspunkt for testing. I basiskonfigurasjonen ligger det hovedsakelig to programmer. Dette er Docker og Minikube. De neste kapitlene vil gjennomgå de prosedyrene som ble utført ved installasjon på maskinene som ble benyttet i prosjektet.

**MERK:** Denne installasjonen av programvarene vil installere de nyeste versjonene som eksisterer. For å etterprøve testene i rapporten er det hensiktsmessig og spesifisere de versjonene som ble benyttet i testene! ved installasjon av programvarene!

For å se hvilke versjoner av de ulike programvarene som ble installert, se [Tabell 1](#)

### 3.1 Docker

I dette prosjektet blir Docker benyttet som en driver for minikube og må derfor installeres før minikube. For installasjon av Docker, ble [installasjonsguiden \(https://docs.docker.com/engine/install/ubuntu/\)](https://docs.docker.com/engine/install/ubuntu/) til Docker Inc fulgt.

Under vil kommandoene som ble benyttet i installasjonen bli listet opp.

Først ble all gammel konfigurasjon av Docker fjernet med følgende kommando:

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

Så ble Docker installert med et repo. Med de neste kommandoene oppdateres og installeres det pakker som gjør det mulig å tillate repo over HTTPS.

```
$ sudo apt-get update

$ sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Her legges Docker Inc sin offisielle GPG-nøkkel til maskinen:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Så eksekveres en kommando for å sette opp et stabilt repo og gjør det klart til installering:

```
$ echo \
"deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list
> /dev/null
```

Til slutt installeres siste versjon av Docker Engine med følgende kommandoer:

**MERK:** Skal testen etterprøves, bør det spesifiseres hvilken versjon som skal lastes ned her, for å få samme versjon som ble benyttet under forsøket i dette prosjektet. Dette er for å skape et likt testmiljø

```
$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-
compose-plugin
```

Nå er Docker installert på maskinen.

### 3.2 minikube

For å utføre konseptbevisene som er laget, trengs et K8s *cluster*. På grunn av bacheloroppgavens tidsbegrensning var det ikke tid nok til å sette opp et fullskala K8s *cluster*. Derfor ble det installert minikube på egne maskiner. minikube lager et virtuelt K8s *cluster* som tillater å teste funksjoner som finnes i fullt oppsatte *cluster*e.

For installasjon av minikube, ble [installasjonsguiden \(https://minikube.sigs.k8s.io/docs/start/\)](https://minikube.sigs.k8s.io/docs/start/) til "The Kubernetes Authors" fulgt.

Under vil kommandoene som ble benyttet i installasjonen bli listet opp.

For å installere siste versjon av minikube på x86-64 Linux med binær nedlastning:

```
$ curl -LO  
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-  
amd64  
$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Etter dette legges brukeren til gruppen docker, så Docker kan kjøres uten å benytte sudo foran hver kommando.

```
$ sudo usermod -aG docker $USER && newgrp docker
```

For å starte minikube og gjøre installasjonen ferdig ved førstegangsinstallasjon:

```
$ minikube start --driver=docker
```

For å installere kubectl må denne kommandoen skrives:

```
$ minikube kubectl -- get pods -A
```

## 4 K8s konfigurasjon

For å kjøre opp applikasjonen, beskrevet i bachelorrapporten kapittel 4.1, i K8s er det benyttet yaml-filer. Disse yaml-filene ligger i repoet *kubernetes-config*, se [Vedlegg B - Kildekode](#).

### 4.1 django-applikasjon

Til django-applikasjonen benyttes kun objektene *Deployment* og *Service*. *Deploymenten* henter et Docker image fra Docker Hub som er utviklet ifb. med dette prosjektet. Docker imaget som er benyttet er [sjohans1/django-bachelor:6.0](https://hub.docker.com/r/sjohans1/django-bachelor/) (<https://hub.docker.com/r/sjohans1/django-bachelor/>). I tillegg er det definert fem miljøvariabler i denne *Deploymenten*. Dette er variabler som setter opp forbindelse med mysql-tjenesten og inneholder databasenavn, databasebruker, databasebrukersens passord, databasens IP/DNS og databasens port. *Servicen* er satt opp som en NodePort og gir derfor tilgang til django-applikasjonen ved

hjelp av en port. Denne gjør slik at *django-entripoint*, som er navnet på *Servicen*, blir eksponert som på porten 30001. Dette vil si at det går an å nå tjenesten fra utsiden av K8s *clusteret* ved hjelp av ip-adressen til minikube og porten 30001.

Når yaml-filene skal legges inn i K8s blir *kustomization.yaml* benyttet. Dette bidrar til at kun én kommando må skrives for å deployere én mikrotjeneste. Skal mikrotjenesten django-applikasjon deployeres, vil det derfor være nok å ha en terminal åpen i mappen django i *kubernetes-config*-repoet og kjøre kommandoen:

```
$ minikube kubectl -- apply -k .
```

## 4.2 mysql

Til mysql-databasen benyttes objektene *PersistentVolume*, *Secret*, *ConfigMap*, *StatefulSet* og *Service*.

I *PersistentVolume* defineres hvor i minikube dataen, som linkes mot dette volumet, skal lagres. Stien til denne mappen i minikube er: *"/data/bachelor-mysql"*

I *Secret* defineres passordet til databasebrukeren, mens i *ConfigMap* bestemmes databasens navn. Disse verdiene er hhv. *"cGFzc3dvcmQK"* og *"bachelor\_db"*.

*StatefulSet*et henter et Docker image, på samme måte som django-applikasjonen over, fra Docker Hub. Imaget er derimot ikke spesielt utviklet til dette prosjektet og er derfor et offisielt Docker image av mysql. Docker imaget og versjonen som er benyttet er mysql:5.7. Siden dette er et *StatefulSet* er det montert et volum som bidrar til persistent lagring av data. Det er også definert en *PersistentVolumeClaim* i *StatefulSet*et som kobles opp mot *PersistentVolumet*.

I tillegg er det definert to miljøvariabler. Disse variablene ligger ikke i selve *StatefulSet*et, men de ligger i objektene *Secret* og *ConfigMap*. Variablene bestemmer databasepassordet til databasen og databasenavnet til databasen som skal opprettes ved deployering av dette *StatefulSet*et. *Service* er en vanlig ClusterIP K8s *Service*, noe som gjør at databasen kun kan aksesseres fra innsiden av K8s *clusteret*.

## 4.3 sched-applikasjon

Til sched-applikasjonen benyttes objektet *Deployment*.

*Deploymenten* henter et Docker image, kalt sched:latest, fra lokal maskin, og kjører dette opp med to miljøvariabler. Den ene variabelen bestemmer intervallet til GET-forespørsler, altså tiden det skal ta mellom hver GET-forespørsel, mens den andre variabelen bestemmer hvilken IP/DNS API-et befinner seg på. Her er django-entripoint:8000 benyttet, ettersom *django-entripoint* er *Servicen* til django-applikasjonen.

Ved test én skal sched-applikasjonen benyttes og blir deployert automatisk via scriptet i mikrotjenesten pod-sletting.

## Vedlegg B - Kildekode

### Innhold:

- [Tabell 1: Gitrepo oppsummering](#)
- [Tabell 2: Gitrepo versjoner test én](#)
- [Tabell 3: Gitrepo versjoner test to](#)
- [Tabell 4: Gitrepo versjoner test tre](#)

Kildekodene danner mange ulike filer og er ikke hensiktsmessig å skrive ut. Kildekodene kan derimot finnes på internett i GitHub hvor de er samlet i en organisasjon som finnes på denne lenken: <https://github.com/CISK-2022-bachelorgruppe>. For hver test ble det benyttet forskjellig kildekode så for å finne riktig kildekode til riktig test, se tabell [2](#), [3](#) og [4](#).

[Tabell 1](#) viser hvilke kildekoder som kan finnes i hvilket repo:

**Tabell 1: Gitrepo oppsummering**

Gitrepo	Hva finnes i repoet?
<a href="#">applikasjoner</a>	I dette repoet finnes egenutviklede mikrotjenester som danner én applikasjon. Mikrotjenestene er django-applikasjon, pod-sletting, python-script-get og sched.
<a href="#">kubernetes-config</a>	Dette repoet inneholder alle kubernetes yaml-filer som benyttes ved deployering av f. eks. <i>Statefulset</i> , <i>Deployments</i> , <i>PersistentVolume</i> osv.
<a href="#">vedlegg</a>	Dette repoet har vedleggene til dette prosjektet, men inneholdt også yaml-filene til test 3, se <a href="#">tabell 4</a>

## Versjoner i GitHub

**Tabell 2: Gitrepo versjoner test én**

Gitrepo	Commit-hash:	Lenke
applikasjon	<a href="#">16caeede03f35f435224556b7a7f0c5a696c78c7</a>	<a href="https://github.com/CISK-2022-bachelorgruppe/applikasjoner/tree/16caeede03f35f435224556b7a7f0c5a696c78c7">https://github.com/CISK-2022-bachelorgruppe/applikasjoner/tree/16caeede03f35f435224556b7a7f0c5a696c78c7</a>
kubernetes-config	<a href="#">b895df683cd9758f0dd3b6cd559106a5cae26b61</a>	<a href="https://github.com/CISK-2022-bachelorgruppe/kubernetes-config/tree/b895df683cd9758f0dd3b6cd559106a5cae26b61">https://github.com/CISK-2022-bachelorgruppe/kubernetes-config/tree/b895df683cd9758f0dd3b6cd559106a5cae26b61</a>

**Tabell 3: Gitrepo versjoner test to**

Gitrepo	Commit-hash:	Lenke
applikasjon	<a href="#">76d350309044b92898ae797c95100c09ffa1c232</a>	<a href="https://github.com/CISK-2022-bachelorgruppe/applikasjoner/tree/76d350309044b92898ae797c95100c09ffa1c232">https://github.com/CISK-2022-bachelorgruppe/applikasjoner/tree/76d350309044b92898ae797c95100c09ffa1c232</a>
kubernetes-config	<a href="#">b895df683cd9758f0dd3b6cd559106a5cae26b61</a>	<a href="https://github.com/CISK-2022-bachelorgruppe/kubernetes-config/tree/b895df683cd9758f0dd3b6cd559106a5cae26b61">https://github.com/CISK-2022-bachelorgruppe/kubernetes-config/tree/b895df683cd9758f0dd3b6cd559106a5cae26b61</a>

**Tabell 4: Gitrepo versjoner test tre**

Gitrepo	Commit-hash:	Lenke
vedlegg/test_3_hpa	<a href="#">86f3047459bb0e858809a3049841976329110567</a>	<a href="https://github.com/CISK-2022-bachelorgruppe/vedlegg/tree/86f3047459bb0e858809a3049841976329110567/test_3_h">https://github.com/CISK-2022-bachelorgruppe/vedlegg/tree/86f3047459bb0e858809a3049841976329110567/test_3_h</a>

# Vedlegg C - Fremgangsmåte test én

---

## Innhold:

1. Introduksjon
2. Fremgangsmåte
3. Gjennomføring
  - Ved testens start
  - Sched er deployert
  - Pod 5 ble akkurat slettet
  - Figuren viser at pod 18 lages
  - Sched termineres
  - Testen er ferdig

## 1. Introduksjon

---

Dette er en detaljert veiledning i hvordan test én ble gjennomført! Denne fremgangsmåten går ut ifra at alle filer som er nødvendige for å eksekvere test én, er lastet ned til hjem-mappen på maskinen "~/". Testen ble gjennomført på maskin A.

**MERK:** Se [Vedlegg B - Kildekode](#) for å se hvor og hvilke versjoner av kildekodene som tilhører test én

## 2. Fremgangsmåte

---

For å sikre at minikube ikke inneholder endrede filer som kan påvirke testingen, installeres minikube på nytt ved starten av hver test. Dette ble gjort med følgende kommandoer:

```
$ minikube delete
$ minikube start --driver=docker
```

Når minikube var ferdiginstallert, ble utrulling av mysql-databasen og django-applikasjonen startet med:

```
$ cd $HOME/kubernetes-config/mysql
$ minikube kubectl -- apply -k .

$ cd $HOME/kubernetes-config/django
$ minikube kubectl -- apply -k .
```

Deretter ble sched-appen klargjort for kjøring i minikube. Dette gjøres med:

```
$ cd $HOME/applikasjon/sched  
$ eval \$(minikube docker-env)  
$ docker build -t sched .
```

For å starte selve testen med å tvangstoppe podder og ta tiden på dette ble følgende kommandoer benyttet:

```
$ cd $HOME/applikasjon/pod-sletting/  
$ ./skript.sh
```

Testen vil lagre sine resultater i mappen hvor kjører *skript.sh* ifra. Siden *skript.sh* eksekveres fra mappen *\$HOME/applikasjon/pod-sletting/*, vil resultatene legges i en mappe *\$HOME/applikasjon/pod-sletting/resultater/*

### 3. Gjennomføring

---

Under vil seks figurer listes opp, hvor tittelen til hver figur er en kort beskrivelse av når figuren ble tatt eller hva som kan observeres på figuren. Alle figurene er tatt underveis i *skript.sh* sin utførelse.



## Ved testens start

april 29 14:23

ole-kristian@Dromedar: ~

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a

<0> all  
<1> default

<a> Atta...  
<ctrl-d> Delet...  
<d> Descr...  
<e> Edit...  
<?> Help...  
<ctrl-k> Kill

NAME ↑	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
django-applikasjon-58cc7cd4fb-mp8zh	●	1/1	0	Running	172.17.0.4	minikube	5m17s
mysql-set-0	●	1/1	0	Running	172.17.0.3	minikube	5m24s

Pods(default)[2]

ole-kristian@Dromedar: ~/Skrivebord/Test1

```
ole-kristian@Dromedar:~/Skrivebord/Test1$ ./skript.sh
Skriptet vil lagre loggfiler m.m i denne mappen: /home/ole-kristian/Skrivebord/Test1
(Husk at appen "sched" må bygges å legges inn i minikube før denne testen startes)

Starter skriptet om 10 sekunder
```

## Sched er deployert

april 29 14:24

ole-kristian@Dromedar: ~

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a

<0> all  
<1> default

<a> Atta...  
<ctrl-d> Delet...  
<d> Descr...  
<e> Edit...  
<?> Help...  
<ctrl-k> Kill

NAME ↑	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
django-applikasjon-58cc7cd4fb-mp8zh	●	1/1	0	Running	172.17.0.4	minikube	5m45s
mysql-set-0	●	1/1	0	Running	172.17.0.3	minikube	5m52s
sched-applikasjon-579c4f75d-j2rbh	●	1/1	0	Running	172.17.0.5	minikube	20s

Pods(default)[3]

ole-kristian@Dromedar: ~/Skrivebord/Test1

```
ole-kristian@Dromedar:~/Skrivebord/Test1$ ./skript.sh
Skriptet vil lagre loggfiler m.m i denne mappen: /home/ole-kristian/Skrivebord/Test1
(Husk at appen "sched" må bygges å legges inn i minikube før denne testen startes)

Starter skriptet om 10 sekunder

-----
|  Skrer av WiFi  |
-----

Sched-applikasjonen deployeres nå!
deployment.apps/sched-applikasjon created

Venter i 2 minutter før første pod slettes
```

## Pod 5 ble akkurat slettet


april 29 14:28

ole-kristian@Dromedar: ~

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a

<0> all  
<1> default

<a> Atta...  
<ctrl-d> Delet  
<d> Descr  
<e> Edit  
<?> Help  
<ctrl-k> Kill



**Pods(default)[3]**

NAME ↑	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
django-applikasjon-58cc7cd4fb-6htn6	●	1/1	0	Running	172.17.0.7	minikube	3s
mysql-set-0	●	1/1	0	Running	172.17.0.3	minikube	9m36s
sched-applikasjon-579c4f75d-j2rbh	●	1/1	0	Running	172.17.0.5	minikube	4m4s

ole-kristian@Dromedar: ~/Skrivebord/Test1

Venter i 2 minutter før første pod slettes

Sletter pod: 1: django-applikasjon-58cc7cd4fb-mp8zh  
pod "django-applikasjon-58cc7cd4fb-mp8zh" force deleted

Sletter pod: 2: django-applikasjon-58cc7cd4fb-j96b5  
pod "django-applikasjon-58cc7cd4fb-j96b5" force deleted

Sletter pod: 3: django-applikasjon-58cc7cd4fb-v5d2w  
pod "django-applikasjon-58cc7cd4fb-v5d2w" force deleted

Sletter pod: 4: django-applikasjon-58cc7cd4fb-dw2ml  
pod "django-applikasjon-58cc7cd4fb-dw2ml" force deleted

Sletter pod: 5: django-applikasjon-58cc7cd4fb-h27fg  
pod "django-applikasjon-58cc7cd4fb-h27fg" force deleted

```
ole-kristian@Dromedar: ~  
Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a  
<0> all  
<1> default  
<a> Atta...  
<ctrl-d> Delet  
<d> Descr  
<e> Edit  
<?> Help  
<ctrl-k> Kill  
NAME ↑ PF READY RESTARTS STATUS IP NODE AGE  
django-applikasjon-58cc7cd4fb-w9qx9 ● 0/1 0 ContainerCreating n/a minikube 1s  
mysql-set-0 ● 1/1 0 Running 172.17.0.3 minikube 15m  
sched-applikasjon-579c4f75d-j2rbh ● 1/1 0 Running 172.17.0.5 minikube 10m  
ole-kristian@Dromedar: ~/Skribebord/Test1  
Sletter pod: 10: django-applikasjon-58cc7cd4fb-rmfx2  
pod "django-applikasjon-58cc7cd4fb-rmfx2" force deleted  
Sletter pod: 11: django-applikasjon-58cc7cd4fb-fgwdw  
pod "django-applikasjon-58cc7cd4fb-fgwdw" force deleted  
Sletter pod: 12: django-applikasjon-58cc7cd4fb-f98kp  
pod "django-applikasjon-58cc7cd4fb-f98kp" force deleted  
Sletter pod: 13: django-applikasjon-58cc7cd4fb-qttrt  
pod "django-applikasjon-58cc7cd4fb-qttrt" force deleted  
Sletter pod: 14: django-applikasjon-58cc7cd4fb-95mz8  
pod "django-applikasjon-58cc7cd4fb-95mz8" force deleted  
Sletter pod: 15: django-applikasjon-58cc7cd4fb-dkpxr  
pod "django-applikasjon-58cc7cd4fb-dkpxr" force deleted  
Sletter pod: 16: django-applikasjon-58cc7cd4fb-8lsvh  
pod "django-applikasjon-58cc7cd4fb-8lsvh" force deleted  
Sletter pod: 17: django-applikasjon-58cc7cd4fb-nwzgg  
pod "django-applikasjon-58cc7cd4fb-nwzgg" force deleted
```



## Sched termineres

april 29 14:41

ole-kristian@Dromedar: ~

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a

<0> all  
<1> default

<a> Atta...  
<ctrl-d> Delet...  
<d> Descr...  
<e> Edit...  
<?> Help...  
<ctrl-k> Kill

**Pods(default)[3]**

NAME ↑	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
django-applikasjon-58cc7cd4fb-nkxpd	●	1/1	0	Running	172.17.0.4	minikube	31s
mysql-set-0	●	1/1	0	Running	172.17.0.3	minikube	22m
sched-applikasjon-579c4f75d-j2rbh	●	1/1	0	TerminatingΔ	172.17.0.5	minikube	17m

ole-kristian@Dromedar: ~/Skrivebord/Test1

Sletter pod: 24: django-applikasjon-58cc7cd4fb-xww7d  
pod "django-applikasjon-58cc7cd4fb-xww7d" force deleted

Sletter pod: 25: django-applikasjon-58cc7cd4fb-lx5c8  
pod "django-applikasjon-58cc7cd4fb-lx5c8" force deleted

Sletter pod: 26: django-applikasjon-58cc7cd4fb-ll4pl  
pod "django-applikasjon-58cc7cd4fb-ll4pl" force deleted

Sletter pod: 27: django-applikasjon-58cc7cd4fb-qrx4r  
pod "django-applikasjon-58cc7cd4fb-qrx4r" force deleted

Sletter pod: 28: django-applikasjon-58cc7cd4fb-7stcp  
pod "django-applikasjon-58cc7cd4fb-7stcp" force deleted

Sletter pod: 29: django-applikasjon-58cc7cd4fb-lqrsj  
pod "django-applikasjon-58cc7cd4fb-lqrsj" force deleted

Sletter pod: 30: django-applikasjon-58cc7cd4fb-7tl8z  
pod "django-applikasjon-58cc7cd4fb-7tl8z" force deleted  
deployment.apps "sched-applikasjon" deleted  
Ferdig med testing. Venter i 30 sekunder

## Testen er ferdig

april 29 14:41

ole-kristian@Dromedar: ~

Context: minikube  
Cluster: minikube  
User: minikube  
K9s Rev: v0.25.18  
K8s Rev: v1.23.3  
CPU: n/a  
MEM: n/a

<0> all  
<1> default

<a> Atta...  
<ctrl-d> Delet...  
<d> Descr...  
<e> Edit...  
<?> Help...  
<ctrl-k> Kill

**Pods(default)[2]**

NAME ↑	PF	READY	RESTARTS	STATUS	IP	NODE	AGE
django-applikasjon-58cc7cd4fb-nkxpd	●	1/1	0	Running	172.17.0.4	minikube	67s
mysql-set-0	●	1/1	0	Running	172.17.0.3	minikube	23m

ole-kristian@Dromedar: ~/Skrivebord/Test1

Sletter pod: 30: django-applikasjon-58cc7cd4fb-7tl8z  
pod "django-applikasjon-58cc7cd4fb-7tl8z" force deleted  
deployment.apps "sched-applikasjon" deleted  
Ferdig med testing. Venter i 30 sekunder  
Eksporterer databasen fra podden mysql  
tar: Removing leading '/' from member names  
ole-kristian@Dromedar:~/Skrivebord/Test1\$

**MERK:** For videre informasjon om scriptene som er benyttet i dette prosjektet, les kildekoden som finnes i [Vedlegg B - Kildekode](#)

# Vedlegg D - Resultater test én

---

Resultatene tilknyttet test én er en databasetabell med 204341 rader. Dette vil ikke være hensiktsmessig å skrive ut, og vil derfor kun være tilgjengelig i digital form. Resultatet kan lastes ned med denne lenken:

<https://github.com/CISK-2022->

[bachelorgruppe/vedlegg/blob/5fd20070b93977b06d85901d9f903a55bd395348/Vedlegg%20D%20-%20Resultater%20test%20%C3%A9n.ods](https://github.com/CISK-2022-bachelorgruppe/vedlegg/blob/5fd20070b93977b06d85901d9f903a55bd395348/Vedlegg%20D%20-%20Resultater%20test%20%C3%A9n.ods)

# Vedlegg E - Fremgangsmåte test to

---

## Innhold:

1. [Introduksjon](#)
2. [Programvare](#)
3. [Oppsett](#)
  - 3.1. [Forklaring av teknisk innhold](#)
    - 3.1.1. [test-gjennomføring.sh](#)
    - 3.1.2. [req.py](#)

## 1. Introduksjon

---

Dette vedlegget tar for seg en detaljert beskrivelse av både kode og gjennomføring til test 2. Testen ble gjennomført på maskin B.

**MERK:** Se [Vedlegg B - Kildekode](#) for å se hvor og hvilke versjoner av kildekodene som tilhører test to

## 2. Programvare

---

Versjoner for programvarer benyttet i denne testen utover det som er beskrevet i oppgaven er:

Programvare	Versjon
Python	3.9.7
Requests (python pakke)	2.25.1
pip3	20.3.4

## 3. Oppsett

---

Som i test én slettes eventuelle tidligere minikube *cluster*e for å sikre at det ikke ligger tidligere konfigurasjon som kan påvirke testingen. Deretter lages et nytt med docker som driver.

```
$ minikube delete
$ minikube start --driver=docker
```

Deretter ble django og mysql fra den egenutviklede applikasjonen satt opp i det nye *clusteret*. Oppsettet av testen fortsatte ikke før siste kommando i kodeboksen under viste at begge podder hadde "Running" status.

```
$ cd $HOME/git/kubernetes-config/k8s-config  
$ minikube kubectl -- apply -k ./django/ && minikube kubectl -- apply -k  
./mysql/  
$ minikube kubectl -- get pods --watch
```

Deretter ble NodePort til `django-entrypoint` testet med kommandoen

```
$ curl $(minikube ip):30001
```

Når kommandoen over returnerte en HTML-side er oppsettet ferdig og testen er klar for å utføres.

Da ble kommandoene under eksekvert for å starte selve testen:

```
$ cd $HOME/git/CISK-2022-bachelorgruppe/applikasjoner/python-script-get  
$ ./test-gjennomføring $(minikube ip) 30001 200 10 $HOME/git sj $HOME/git
```

### 3.1 Forklaring av teknisk innhold:

Scriptene som forklares under er lagd ifb. med mikrotjenesten python-script-get. [Kapittel 1](#) forklarer hvor scriptene kan finnes.



### 3.1.1 test-gjennomføring.sh

Dette er et script som vil gjøre hele testen selv. Det har tre **for**-løkker inne i hverandre som hører til antall podder, antall tråder og antall gjennomføringer. Nedenfor er et utsnitt av de viktigste linjene fra scriptet.

```
for pod in {1..10}
do
    minikube kubectl -- scale --replicas=$pod -f $path/k8s-
bachelor/k8s-config/django/django-deployment.yaml
    sleep 12
    for z in {1..10}
    do
        fil="$path/bachelor-applikasjon/python-script-
get/resultater/$tid/$pod-podder.$z-trader-$tid.txt"
        for i in $(seq 1 $gjennomforinger)
        do
            /usr/bin/time -a -o "$fil" -f "%E" nice bash -c "python3
$path/bachelor-applikasjon/python-script-get/req.py --host $host --port
$port --antall $foresporsler --thr $z"

            if [ $i != 10 ]; then
                sleep 12
            else
                echo "Test med $z tråd(er) ferdig!"
            fi
        done
    done
done
```

De viktigste linjene å legge merke til er linjen `minikube kubectl -- scale --replicas=$pod -f $path/k8s-bachelor/k8s-config/django/django-deployment.yaml` og linjen `/usr/bin/time -a -o "$fil" -f "%E" nice bash -c "python3 $path/bachelor-applikasjon/python-script-get/req.py --host $host --port $port --antall $foresporsler --thr $z"`. Den første linjen skalerer opp antall podder mellom hver hundrede gjennomføring av req.py scriptet, mens den andre linjen gjør selve testen ved å ta tiden på hvor lenge det tar å gjennomføre 200 forespørsler med `$z` tråder mot `$pod` podder.

### 3.1.2 req.py

Nedenfor vises et utsnitt av python scriptet. Som vist vil alle trådene samarbeide om å nå 200 forespørsler. Forespørslene fordeles ikke nødvendigvis likt utover trådene, men neste forespørsel blir delt ut til neste tråd som er ledig. De siste forespørslene er det **master**tråden som utfører selv for å sikre at det blir sendt akkurat 200 forespørsler.

```
def forespørsel(host, port, antall, thr):
    global value
    while value < antall-thr:
        r = requests.get(f"http://{host}:{port}")
        if r.status_code == 200:
            with threadLock:
                value += 1
            if value % 50 == 0:
                print(f"Det er gjennomført {value}/{antall} tester.")

value = 0
threads = []
for i in range(args.thr):
    t = threading.Thread(target=forespørsel, args=(args.host, args.port,
args.antall, args.thr))
    threads.append(t)
    t.start()
```

[...]

```
if args.thr != 0:
    while value < args.antall:
        r = requests.get(f"http://{args.host}:{args.port}")
        if r.status_code == 200:
            with threadLock:
                value += 1
            if value % 50 == 0:
                print(f"Det er gjennomført {value}/{args.antall}
tester.")
```

**MERK:** For videre informasjon om scriptene som er benyttet i dette prosjektet, les kildekoden som finnes i [Vedlegg B - Kildekode](#)

# Vedlegg F - Resultater test to

---

Resultatene tilknyttet test to er fordelt på 100 filer. Dette vil ikke være hensiktsmessig å skrive ut, og vil derfor kun være tilgjengelig i digital form. Resultatet kan finnes i et repo på GitHub med denne lenken:

<https://github.com/CISK-2022->

[bachelorgruppe/vedlegg/tree/5fd20070b93977b06d85901d9f903a55bd395348/Vedlegg%20F%20-%20Resultater%20test%20to](https://github.com/CISK-2022-bachelorgruppe/vedlegg/tree/5fd20070b93977b06d85901d9f903a55bd395348/Vedlegg%20F%20-%20Resultater%20test%20to)

# Vedlegg G - Fremgangsmåte test tre

---

## Innhold:

1. [Innledning](#)
2. [Fremgangsmåte og gjennomføring](#)
3. [Forklaring av teknisk innhold](#)
  - 3.1. [php-apache.yaml](#)
  - 3.2. [hpa-php-apache.yaml](#)
  - 3.3. [Lastgenerering med BusyBox](#)

## 1 Innledning

---

Denne testen er basert på [HorizontalPodAutoscaler Walkthrough \(https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/\)](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/), og er en detaljert veiledning i hvordan test tre ble gjennomført!

**MERK:** Se [Vedlegg B - Kildekode](#) for å se hvor og hvilke versjoner av kildekodene som tilhører test tre

## 2 Fremgangsmåte og gjennomføring

---

Åpne to terminalvinduer, heretter referert til terminal A og terminal B.

1. Start minikube med denne kommandoen i terminal A:

```
$ minikube start --driver docker --extra-config=kubelet.housekeeping-interval=10s
```

For at metrics-server skal fungere i neste steg, så må `--extra-config=kubelet.housekeeping-interval=10s` være med i oppstart av minikube.

Svar fra kommandoene:

```
🤖 minikube v1.25.1 on Ubuntu 20.04
🎉 minikube 1.25.2 is available! Download it:
https://github.com/kubernetes/minikube/releases/tag/v1.25.2
💡 To disable this notice, run: 'minikube config set
WantUpdateNotification false'

🌟 Using the docker driver based on existing profile
👍 Starting control plane node minikube in cluster minikube
🚚 Pulling base image ...
🏃 Updating the running docker "minikube" container ...
🐳 Preparing Kubernetes v1.23.1 on Docker 20.10.12 ...
▪ kubelet.housekeeping-interval=10s
🔍 Verifying Kubernetes components...
▪ Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: storage-provisioner, default-storageclass, metrics-
server
🏁 Done! kubectl is now configured to use "minikube" cluster and "default"
namespace by default
```

## 2. Start den innebygde tilleggsfunksjonen 'metrics-server' i minikube:

```
$ minikube addons enable metrics-server
```

Svar fra kommandoene:

```
▪ Using image k8s.gcr.io/metrics-server/metrics-server:v0.4.2
🌟 The 'metrics-server' addon is enabled
```

For å se om metrics-server fungerer:

```
$ minikube kubectl -- top pods -n kube-system
```

Svar fra kommandoen vil være noe lik denne:

NAME	CPU(cores)	MEMORY(bytes)
coredns-64897985d-q52bj	3m	13Mi
etcd-minikube	25m	48Mi
kube-apiserver-minikube	81m	263Mi
kube-controller-manager-minikube	32m	48Mi
kube-proxy-zrn6b	1m	10Mi
kube-scheduler-minikube	4m	16Mi
metrics-server-6b76bd68b6-g5klg	6m	17Mi
storage-provisioner	2m	9Mi

### 3. Start *Deployment* og eksponer *Serivcen*:

```
$ minikube kubectl -- apply -f php-apache.yaml
```

Svar fra kommandoen:

```
deployment.apps/php-apache created  
service/php-apache created
```

### 4. Lag HPA og sjekk *current* status:

```
$ minikube kubectl -- apply -f hpa-php-apache.yaml
```

Svar fra kommandoen:

```
horizontalpodautoscaler.autoscaling/php-apache created
```

Så sjekkes status til HPA med en gang:

```
$ minikube kubectl -- get hpa
```

Svar fra kommandoen:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
php-apache	Deployment/php-apache	<unknown>/50%	1	10
12s				0

Legg merke til **<unknown>**. Dette kan skje hvis statusen sjekkes, før HPA er ferdig deployert. Derfor ble de spurt om status inntil **0%** vises. HPA fungerer selv om **0%** vises.

Svar fra kommandoen etter ett minutt:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
php-apache	Deployment/php-apache	0%/50%	1	10	1
70s					

## 5. Generer mer last med BusyBox i Terminal B:

```
$ minikube kubectl -- run -i --tty load-generator --rm --image=busybox:1.28  
--restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-  
apache; done"
```

Svar fra kommandoen:

```
If you don't see a command prompt, try pressing enter.  
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!  
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!  
OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!
```

## 6. Overvåk HPA i Terminal A:

```
$ minikube kubectl -- get hpa php-apache --watch
```

Denne kommandoen kjøres på et intervall på 15 sekunder.

Svar fra kommandoen etter noen minutter:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
php-apache 58s	Deployment/php-apache	0%/50%	1	10	1
php-apache 75s	Deployment/php-apache	132%/50%	1	10	1
php-apache 90s	Deployment/php-apache	171%/50%	1	10	3
php-apache 105s	Deployment/php-apache	157%/50%	1	10	4
php-apache 2m	Deployment/php-apache	61%/50%	1	10	4
php-apache 2m15s	Deployment/php-apache	65%/50%	1	10	4
php-apache 2m30s	Deployment/php-apache	78%/50%	1	10	4
php-apache 2m45s	Deployment/php-apache	72%/50%	1	10	7
php-apache 3m	Deployment/php-apache	57%/50%	1	10	7
php-apache 3m21s	Deployment/php-apache	43%/50%	1	10	7
php-apache 3m30s	Deployment/php-apache	51%/50%	1	10	7

Overvåk autoskaleringen frem til prosessorkraften er stabil rundt `targetCPUUtilizationPercentage=50` i 5 minutter.

## 7. Stopp lasten BusyBox genererer i Terminal B:

```
$ <ctrl> + c
```



Svar fra kommandoen:

```
OK!OK!OK!OK!OK!^Cpod "load-generator" deleted  
pod default/load-generator terminated (Error)`
```

## 8. Overvåk HPA i Terminal A og se den skalere ned:

```
$ minikube kubectl -- get hpa php-apache --watch
```

Når HPA detekterer at CPU=0% skalerer den automatisk ned til 1 replika. Dette kan ta noen minutter.

Svar fra kommandoen:

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS
AGE					
php-apache 4m46s	Deployment/php-apache	42%/50%	1	10	7
php-apache 5m1s	Deployment/php-apache	5%/50%	1	10	7
php-apache 5m16s	Deployment/php-apache	0%/50%	1	10	7
php-apache 7m	Deployment/php-apache	0%/50%	1	10	7
php-apache 7m52s	Deployment/php-apache	0%/50%	1	10	7
php-apache 9m31s	Deployment/php-apache	0%/50%	1	10	7
php-apache 9m46s	Deployment/php-apache	0%/50%	1	10	6
php-apache 10m	Deployment/php-apache	0%/50%	1	10	1

## 3 Forklaring av teknisk innhold:

Her er en mer detaljert beskrivelse av de forskjellige tekniske komponentene i testen. Det vil ikke bli gjort en detaljert linje-for-linje beskrivelse av .yaml-filene. [Kapittel 1](#) forklarer hvor scriptene kan finnes.

### 3.1 php-apache.yaml

Denne filen lager en *Deployment* og eksponerer den med en *Service*. *Deploymenten* henter et ferdiglaget php-apache image fra Docker Hub. Imaget er `k8s.gcr.io/hpa-example` og har php-image versjon `php:5-apache` og er bygget opp ved hjelp av en Dockerfile som inneholder:

```
FROM php:5-apache
COPY index.php /var/www/html/index.php
RUN chmod a+rx index.php
```

Koden over refererer til en `index.php`-side som utfører komplekse regnestykker som krever mye prosessorkraft. Dette er for å simulere lasten i *clusteret* vårt og er som følger:

```
<?php
$x = 0.0001;
for ($i = 0; $i <= 1000000; $i++) {
    $x += sqrt($x);
}
echo "OK!";
?>
```

### 3.2 hpa-php-apache.yaml

Denne filen konstruerer en HPA som inneholder beskrivelser om at *Deploymenten* php-apache skal ha minst én pod og maks ti podder. Denne HPA-en vil enten øke eller minke antall replikeringer innenfor dette intervallet, for å opprettholde ønsket gjennomsnittlig prosessorutnyttelse på tvers av alle podder på cirka 50%. Algoritmen som HPA benytter for å bestemme antall replikeringer baserer seg på forholdet mellom ønsket metrics-verdi og gjeldende metrics-verdi hentet fra `metrics-server`. Her vises algoritmen, som er forenklet for å lett kunne forstå hva den gjør:

```
ønsketreplikeringer = ceil[GjeldendeReplikas * ( GjeldendeMetricVerdi /
ØnsketMetricVerdi )]
```

Funksjonen `ceil[]` runder opp til neste heltall.

### 3.3 Lastgenerering med BusyBox

For å generere en økt last mot php-apache servicen, eksekveres en lastgenerator som heter BusyBox. Den henter et image busybox:1.28 og oppretter en pod som sender forespørsler til <http://php-apache> helt til podden termineres manuelt.

**MERK:** For videre informasjon om scriptene som er benyttet i dette prosjektet, les kildekoden som finnes i [Vedlegg B - Kildekode](#)