

Group 45 DCSG2003 Oblig 2

Raphael Storm Larsen, Sara Stentvedt Luggenes | 17.03.2023

Week 5 - Docker

1) Docker Installation

We create a new virtual machine "www3", and install docker on it. We run the following commands, which are also added to the script "initWorker.sh", a script that automatically downloads and installs all necessary software used by any type of worker.

```
# Docker Installation taken from initWorker.sh

# Older versions of Docker went by the names of docker, docker.io, or docker-
engine. Uninstall any such older versions before attempting to install a new
version:
sudo apt-get -y remove docker docker-engine docker.io containerd runc

# Update the apt package index and install packages to allow apt to use a
repository over HTTPS:
sudo apt-get -y update
sudo apt-get -y install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \

# Add Docker's official GPG key:
sudo apt-get -y update
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg

# Use the following command to set up the repository:
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null

# Update the apt package index:
sudo apt-get -y update

# Fix potentially broken mask
sudo chmod a+r /etc/apt/keyrings/docker.gpg
sudo apt-get -y update

# Install Docker Engine, containerd, and Docker Compose.
sudo apt-get -y install docker-ce docker-ce-cli containerd.io docker-compose-
```

```
plugin

# Fix potentially broken docker build
sudo echo "{ \"insecure-registries\" : [\"192.168.128.23:5000\"] }" >
/etc/docker/daemon.json
```

2) Bookface optimization

By giving the tables indexes, we can optimize the database for faster lookups. We do this by running this command in cockroachDB sql-shell:

```
create index on pictures (pictureid);
```

By running the code `explain SELECT picture FROM pictures WHERE pictureid = '7VAgCDqHlHeLtNK5wjiApbjy1Vj63L.jpg'`; we now see that the database only searched for 1 row to find the picture with this id. This is because this picture-id is the first picture in the database. Without indexes in the database, running the code above would take longer time and give the output "FULL SCAN" instead of "1 row" in the "estimated row count" line.

```
root@localhost:26257/defaultdb> use bf;
SET

Time: 2ms total (execution 1ms / network 1ms)

root@localhost:26257/bf> select picture from users limit 1;
      picture
-----
qYez99UZC4y1kIF59r5MIFBwmz6SpE.jpg
(1 row)

Time: 5ms total (execution 1ms / network 4ms)

root@localhost:26257/bf> explain SELECT picture FROM pictures WHERE pictureid = 'qYez99UZC4y1kIF59r5MIFBwmz6SpE.jpg';
info
-----
distribution: local
vectorized: true

• index join
  estimated row count: 1
  table: pictures@pictures_pkey
• scan
  estimated row count: 1 (<0.01% of the table; stats collected 1 day ago; using stats forecast for 5 hours in the future)
  table: pictures@picturesindex
  spans: [/'qYez99UZC4y1kIF59r5MIFBwmz6SpE.jpg' - /'qYez99UZC4y1kIF59r5MIFBwmz6SpE.jpg']

index recommendations: 1
1. type: index replacement
   SQL commands: CREATE INDEX ON pictures (pictureid) STORING (picture); DROP INDEX pictures@picturesindex;
(15 rows)

Time: 31ms total (execution 2ms / network 29ms)
```

We also run this commands for indexing the other tables in the database as well:

```
create index on users (name);
create index on users (userid);
create index on users (lastpostdate);
create index on posts (userid);
create index on posts (postdate);
```

```
create index on comments (postid,postdate);
create index on comments (postid);
```

3) Docker image for webserver

We make our docker file with all configs integrated, mainly because we couldn't figure out how shared volumes worked. This is our dockerfile as of 13.13.2023:

```
FROM 192.168.128.23:5000/ubuntu:20.04
MAINTAINER raphaesi@stud.ntnu.no
MAINTAINER saraslu@stud.ntnu.no

ENV DEBIAN_FRONTEND=noninteractive

ARG fpLimit
ENV BF_FRONTPAGE_LIMIT=$fpLimit

# Install Apache
RUN apt-get update && apt-get install -y apache2 libapache2-mod-php php-pgsql net-
tools php-memcache

# Configure files for apache
RUN rm /var/www/html/index.html
RUN mkdir /var/www/html/images

# Add content to apache folder
ADD code/* /var/www/html/
ADD config.php /var/www/html

# Misc
EXPOSE 80

ENTRYPOINT ["/usr/sbin/apache2ctl","-D","FOREGROUND","-k","start"]
```

In order to easily change the frontpage limit, we have added it as an environment variable, which can be set as a command line parameter when running the docker instance.

To build our image we use the command below. Set the desired number of frontpage posts with the parameter.

```
# Use while in same directory as dockerfile
docker build -t "webserver:latest" --build-arg fpLimit=<frontpageLimit> .
```

4+10) Launch bookface webserver

To launch our bookface webserver instances, we use the following commands:

```

docker run -P --name "dwww1" --restart unless-stopped -p 32768:80 -d
webserver:latest
docker run -P --name "dwww2" --restart unless-stopped -p 32769:80 -d
webserver:latest
docker run -P --name "dwww3" --restart unless-stopped -p 32770:80 -d
webserver:latest
docker run -P --name "dwwwFallback" --restart unless-stopped -p 32771:80 -d
fallbackwebserver:v$fversion

```

We have four webserver, where three of them are bookface webserver, and one of them is a fallback webserver, which has its own docker image. The purpose of this instance is to display a "Bookface is down" message in the case that bookface is down (which happens quite a lot). All instances have the option "restart" set to "unless-stopped". This makes the instances restart automatically after a reboot, which is not the default behaviour. We have also decided on some standardized ports that'll work with our haproxy configuration. This is necessary as docker instances use randomized ports if not manually set.

5) Balancer configuration for docker

Our current haProxy configuration file is pasted below. Under the "docker" backend you will find connections to all docker instances.

```

global
    log /dev/log      local0
    log /dev/log      local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd

listeners
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

    # Default SSL material locations
    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    # See: https://ssl-config.mozilla.org/#server=haproxy&server-
version=2.0.3&config=intermediate
    ssl-default-bind-ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-
GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-
AES256-GCM-SHA384
    ssl-default-bind-ciphersuites
    TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
    ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
    log      global
    mode     http
    option   httplog

```

```
option dontlognull
timeout connect 5000
timeout client 50000
timeout server 50000
errorfile 400 /etc/haproxy/errors/400.http
errorfile 403 /etc/haproxy/errors/403.http
errorfile 408 /etc/haproxy/errors/408.http
errorfile 500 /etc/haproxy/errors/500.http
errorfile 502 /etc/haproxy/errors/502.http
errorfile 503 /etc/haproxy/errors/503.http
errorfile 504 /etc/haproxy/errors/504.http

frontend AETA
    bind *:80
    mode http
    default_backend docker

backend docker
    mode http
    balance roundrobin
    server dwww1 192.168.130.6:32768 weight 1
    server dwww2 192.168.130.6:32769 weight 1
    server dwww3 192.168.130.6:32770 weight 1

listen stats
    bind *:1936
    stats enable
    stats uri /
    stats hide-version
    stats auth raphaesi:boing
    stats auth saraslu:boing
```

6) Topology Map

12) Docker Swarm Definition

Docker swarm is a technology integrated into the docker software. A docker swarm let's you make an abstraction of multiple physical servers, or in our case, openstack virtual machines. All of these "workers" included in the swarm will act sort of like a cloud. Instead of pushing docker instances on a single machine, it's pushed to the swarm. This swarm of workers will automatically balance load amongst each other, and will keep redundancy even if one of the workers goes down.

13) Cloud initialization script

The download instructions stated above are already ripped straight from our finished cloud initialization script called "initWorker". However, in later weeks we have added more functionality to the script. Because the parts of the script related to docker was already listed above, we will just dump the rest of the script here:

```
#!/bin/bash

crVersion="22.2.5" #Cockroach database version


# Older versions of Docker went by the names of docker, docker.io, or docker-
engine. Uninstall any such older versions before attempting to install a new
version:
sudo apt-get -y remove docker docker-engine docker.io containerd runc


# Update the apt package index and install packages to allow apt to use a
repository over HTTPS:
sudo apt-get -y update
sudo apt-get -y install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release \
    git


# Add Docker's official GPG key:
sudo apt-get -y update
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg


# Use the following command to set up the repository:
echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null


# Update the apt package index:
sudo apt-get -y update


# Fix potentially broken mask
sudo chmod a+r /etc/apt/keyrings/docker.gpg
sudo apt-get -y update


# Install Docker Engine, containerd, and Docker Compose.
sudo apt-get -y install docker-ce docker-ce-cli containerd.io docker-compose-
plugin


# Add to docker swarm cluster as worker
docker swarm join --token SWMTKN-1-
0wljbwbm6k4zd2yqtypryu6us24jvnfg231j3ztnpy24u9uv3s-496ykad1sirmhbjaq895yy8xq
192.168.130.103:2377


# Fix potentially broken docker build
sudo echo "{ \"insecure-registries\" : [\"192.168.128.23:5000\"] }" >
/etc/docker/daemon.json
```

```
# Sync time using NTP
apt-get install -y ntpdate
ntpdate -b ntp.justervesenet.no

# Set correct timezone
timedatectl set-timezone Europe/Oslo

# Authenticate manager for root user
echo "ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQgQDPU7c1M98QrF5D52HBMHBDAiDvTFEvA6Sdonl4oqCn3NavjmCM5P
ZBk4No7z8H4SX3P6VU98RE7pb/Xug5EMa9AQImWTRzausAShtx0L9yGoXzOXZjctFcTDoN60jV2/Bc8ai8
e48A7x1Zt89Hcvc+ktE5Xd1GUXVXrTK26JPHwn5EkEti4rXlhTf42jJwYGS97vgiLxQG1TRs5/5e4d9zj2
iwAb5m/ml3TxsBpnauTcCeIYTN2R8b6Cl6DIwRFiYFW+lAoR395vrDayD+VPjUS9S1Gqfn73deV39vx0Kz
qor0KuA2kqL5N2YMXwMP3lTOek98hJuFUR01bfQIkc4kJNJfumka82NlwwNIIsJIITci/+pNgqiWHhTyPcF
MsFNFKJoQf5idNYT6PAGZmcsMOqqonTXfd65Pn91qeiq25I3cpgXZ/Z4S5Vmjr20a0roTe40o3sho7Z6fz
AIhKTNhcE/zDjBntKwRH8n73RpyFN08zYXCa4lRn8YSmtwfTMfk= ubuntu@manager" >
/root/.ssh/authorized_keys

# Download and install cockroach database
wget https://binaries.cockroachdb.com/cockroach-v$crVersion.linux-amd64.tgz
tar xzf cockroach-v$crVersion.linux-amd64.tgz
cp cockroach-v$crVersion.linux-amd64/cockroach /usr/local/bin
mkdir /bfdata

# Download and install glusterFS
apt-get -y install glusterfs-server glusterfs-client
systemctl enable glusterd
systemctl start glusterd

# Make brick directories for glusterfs
mkdir /bf_brick
mkdir /config_brick

# Make brick mounting directories
mkdir /bf_images
mkdir /bf_config

# Clone git repository
git clone https://glpat-yEr3xU4_23z-
jJ6UGC29@gitlab.stud.idi.ntnu.no/raphaesi/dcsg2003.git /home/ubuntu/dcsg2003
git config --global --add safe.directory /home/ubuntu/dcsg2003
sudo chown -R ubuntu:ubuntu /home/ubuntu/dcsg2003

# Make Scripts Executable
chmod +x /users/ubuntu/dcsg2003/configuration/chmodder.sh
bash /users/ubuntu/dcsg2003/configuration/chmodder.sh

# Set crontab file
crontab < /home/ubuntu/dcsg2003/configuration/worker/cron
```

In order to create a new worker, we use command below. Set the name of the virtual machine in place of serverName. This will create one with a medium large flavor, this can also be swapped for something more or less powerful. Also, always remember to run the `DCSG2003_V23_group45-openrc.sh` script before using openstack commands, as they will not function without it. It must also be included in any script using these commands.

```
openstack server create --image db1bc18e-81e3-477e-9067-eecaa459ec33 --flavor
gx3.4c4r --security-group default --user-data
"/home/ubuntu/dcsg2003/configuration/initWorker.sh" --key-name "manager"
<serverName>
```

14) Docker swarm set up

Partially Done

This task is only partially done, as we have only set up the swarm, and not yet used it for anything.

To create the swarm we need one manager and some workers. We obviously want our manager VM to be the manager of the swarm, so we run the command below on it. The ip is the internal ip of manager. This command will also output a token: `SWMTKN-1-0wljwbwm6k4zd2yqtypryu6us24jvnfg231j3ztnpy24u9uv3s-496ykad1sirmhbjq895yy8xq` wich we save for later use.

```
docker swarm init --advertise-addr 192.168.130.103
```

Then we must add our worker VM's to the swarm. Included in the initWorker script is a line:

```
docker swarm join --token SWMTKN-1-
0wljwbwm6k4zd2yqtypryu6us24jvnfg231j3ztnpy24u9uv3s-496ykad1sirmhbjq895yy8xq
192.168.130.103:2377
```

This command makes the worker join the swarm using the token and ip of manager.

Week 6 - Shell Scripting

1) Base Script

a) We make our base script, and paste the color commands into it. We then make our color test script:

```
#!/bin/bash
source ~/dcsg2003/configurations/manager/base.sh
info "Dette er en infotekst"
error "Dette er en feilmelding"
warn "Dette er en advarsel"
ok "Her er alt OK"
```



```
ubuntu@manager: ~  
ubuntu@manager:~$ bash ./dcs2003/oblig2/fargetest.sh  
Dette er en infotekst  
Dette er en feilmelding  
Dette er en advarsel  
Her er alt OK  
ubuntu@manager:~$
```

When we run the script, this is our output:

b) No, the script and base.sh are not required to be in the same folder if an absolute path is used with the *source* command.

2) Count database users script

Below is the code copy pasted from the script */configuration/database/countDbEntries.sh*

```
#!/bin/bash  
  
userCount=$(cockroach --insecure --host=localhost sql --execute="use bf;SELECT  
COUNT(userid) FROM users;" | head -3 | tail -1)  
postCount=$(cockroach --insecure --host=localhost sql --execute="use bf;SELECT  
COUNT(postid) FROM posts;" | head -3 | tail -1)  
commentCount=$(cockroach --insecure --host=localhost sql --execute="use bf;SELECT  
COUNT(commentid) FROM comments;" | head -3 | tail -1)  
  
echo "Users: $userCount Posts: $postCount Comments: $commentCount"
```

This is the output of the script when run on the database VM:

```
ubuntu@db2:~$ bash ./dcs2003/configuration/database/countDbEntries.sh  
Users: 25266 Posts: 11507 Comments: 10296
```

4) Exit Values

An exit value is sort of a "status code" for a command. After every linux command has been run, the variable `$?` is filled with a number representing a specific error. 0 means no error, while 1-255 signifies an error. The exit value variable is overwritten after every command, so it's important to use it immediatly if you wish to utelize it in a script. Below is an edited snippet from our "downDetector" script, wich tests if a virtual machine is online by attempting an SSH connection. If there is an error with the connection, it is registered as a downed VM and immediatly restarted.

```
ip=<ip of server>
ssh -o "StrictHostKeyChecking no" -q ubuntu@$ip exit
if [ $? -ne 0 ]
then
    #Restart VM
done
```

7) Openstack commands

In our script "production", we move a floating IP back and fourth between our balancer and our fallback webserver, depending on the argument given. As always with openstack commands, it's critical to source the openrc file in the script.

```
#!/bin/bash
source /home/ubuntu/DCSG2003_V23_group45-openrc.sh
source /home/ubuntu/dcsg2003/configuration/base.sh

floatingIp="10.212.169.121"
fallBackName="www3"
fallBackPort="32771"

if [[ $1 == "up" ]]
then
    openstack server remove floating ip www3 $floatingIp
    openstack server add floating ip balancer $floatingIp

    echo "Bookface has successfully reinstated production"
elif [[ $1 == "down" ]]
then
    openstack server remove floating ip balancer $floatingIp
    openstack server add floating ip $fallBackName $floatingIp

    echo "Bookface has successfully closed production"
else
    echo "Error: Invalid argument \"$1\""
fi
```

8) Mise en place

Our procedure for closing and opening production can be found under /prosedyrer in our gitlab repo. It's extremely simple thanks to the script "production" previously shown.

Prosedyre for starting/stopping av produksjon

1. På *manager* VM'en, bruk kommando *production down*. Dette vil redirekte trafikk til en alternativ host, og frigjøre bookface for store konfigurasjonsendringer.
2. Når du er ferdig, kjør kommandoen *production up*. Dette vil åpne trafikk tilbake til bookface.

Week 7 - Memcache

Alternative 2, Docker Method

To install memcache, we used the 2. alternative, by installing a memcache docker image on our webserver, www3. By running this command, we create and starts the official memcache image:

```
docker run --name=memcache -p 11211:11211 -d 192.168.130.6:5000/memcached
memcached -m 1g
```

The ip-adress is the webserver's ip-adress. We give the image 1 GB RAM to use.

First we need to install this library: `apt-get install -y php-memcache`. Without this library, we will get an error when we make changes to the config.php file in bookface. We install this library in the Dockerfile, not on the webserver. Then we add these lines to the config.php:

```
$memcache_server = "192.168.130.6:11211";
$memcache_enabled = 1;
$memcache_enabled_pictures = 1;
```

The ip-adress in the code above is webserver.

These lines are added to the config.php in all of the webserver images (we have three). The output of the docker ps command will now look like this:

```
root@www3:/# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
ed5e6c6f6a0c	webserver:latest	"/usr/sbin/apache2ct..."	2 weeks ago
Up 18 hours	0.0.0.0:32770->80/tcp, :::32770->80/tcp	dwww3	
414c47dad476	webserver:latest	"/usr/sbin/apache2ct..."	2 weeks ago
Up 18 hours	0.0.0.0:32769->80/tcp, :::32769->80/tcp	dwww2	
367938a83d1b	webserver:latest	"/usr/sbin/apache2ct..."	2 weeks ago
Up 18 hours	0.0.0.0:32768->80/tcp, :::32768->80/tcp	dwww1	
6ddb0327815d	fallbackwebserver:v4	"/usr/sbin/apache2ct..."	2 weeks ago
Up 18 hours	0.0.0.0:80->80/tcp, :::80->80/tcp	dwwwFallback2	
659d225688ac	webserver:v18	"/usr/sbin/apache2ct..."	3 weeks ago
Up 18 hours	0.0.0.0:32772->80/tcp, :::32772->80/tcp	cwww	
d8d7e161a78a	192.168.128.23:5000/memcached	"docker-entrypoint.s..."	4 weeks ago
Up 18 hours	0.0.0.0:11211->11211/tcp, :::11211->11211/tcp	memcache	

We can, by looking at the picture, verify that the docker memcache instance is indeed running.

Check whether the memcache setup works:

By running this command, we get the following output:

```
ubuntu@manager:~$ curl -s http://10.212.169.121 | grep -i memcache
<! Memcache is enabled !><! using memcache for randomized users !>
<! using memcache for randomized posters !>
<! Using memcache to display counters !>
```

This verify that memcache is indeed working as expected.

Week 8 - backup

1) Backup VM

First, we create our new backup VM: "backup". We also attach our newly created openstack volume to it. As for SSH, we decide to use an alternative solution to the one proposed in the assignment. We give the manager ssh key to the backup VM, so that it might contact the database.

3) Backup Script

Then to make a working backup solution. There are four components to our system. On manager, you have the script "backupDatabase.sh". On the backup VM, there is "backup.sh", and "restore.sh". And on database, there is "checkDbStatus.sh", which is a script that was written for a previous assignment. The purpose of the script is to check the status of cockroach, and restart it if it's down.

The first step in the backup process is to run backupDatabase.sh on manager. This can safely be done with the database running and backup VM shut off. We use crontab to automatically backup each morning 08:00, when our bookface has the least visitors.

backupDatabase.sh will first exit bookface production, and reroute visitors to our fallback webserver. Then, the backup VM is started using an openstack command. The script will then repeatedly attempt an SSH connection towards backup until it succeeds. Once in, it will run "backup.sh" and exit. Once this script is done, the backup is complete. The backup VM is turned back off, and production is reinstated.

```
backupDatabase.sh
#!/bin/bash
source /home/ubuntu/DCSG2003_V23_group45-openrc.sh
source /home/ubuntu/dcsg2003/configuration/base.sh

name="backup"
ip=$(openstack server list --name $name | tail -2 | head -1 | cut -d'|' -f5 |
grep -o -P '(?<=192.168.).*(?= )' | awk '{print "192.168."$0}' | xargs)

discord_log "Initiating daily backup..."

# Turn production off
bash /home/ubuntu/dcsg2003/configuration/manager/production.sh down

# Start backup server
openstack server start $name

# Wait for valid ssh connection
```

```

until ssh root@$ip exit
do
    sleep 5
done

# Execute backup script
ssh root@$ip bash /home/ubuntu/dcsg2003/configuration/backupServer/backup.sh

# Turn off backup server
openstack server stop $name

# Restore Production status
bash /home/ubuntu/dcsg2003/configuration/manager/production.sh up

```

Once the "backup.sh" script is run on the backup server, it will first mount the volume. Then it will make an SSH connection towards the database VM, use it to kill the cockroach process. With the database down, it will start copying all the content of /bf_data to a local temporary folder. Once the download is complete, "checkDbStatus.sh" is run via SSH on the database VM, which will detect the DB as down, and immediately restart it. Meanwhile, the copied files are compressed into a tarball, which is then moved to the backup volume. We also have a mechanism in place that deletes all but the three newest backups on every backup. This frees up a lot of space, but makes our backups have a three day expiration date.

Once the backup is done, the script exits. The script on manager will continue, and the VM will be shut down until it is run again the next morning.

```

backup.sh
#!/bin/bash
source /home/ubuntu/dcsg2003/configuration/base.sh

dbIp="192.168.130.246"
ssh -o "StrictHostKeyChecking no" ubuntu@$dbIp exit

# Manager turns on backup-server.

# Mount volume to backup
mount /dev/vdb /backup

# Deletes all but the 3 newest backups
ls -t bfdata* | tail -n +4 | xargs rm -f

# Turn off cockroach database
pid=$(ssh root@$dbIp ps aux | grep "cockroach" | grep -v "grep" | awk '{print $2}' | head -1)
ssh root@$dbIp kill $pid

# Delete temporary folder
rm -r /tmp/bfdata

# Copy bfdata to /tmp/bfdata
scp -r root@$dbIp:/bfdata /tmp/bfdata/

```

```

if [ $? -ne 0 ]
then
discord_log "There was an error when copying database files."
ssh root@$dbIp bash /home/ubuntu/dcsg2003/configuration/database/checkDbStatus.sh
exit
done

# Turn on cockroach database
ssh root@$dbIp bash /home/ubuntu/dcsg2003/configuration/database/checkDbStatus.sh

# Archive bfdata to zip
tar cvzf bfdata_backup_$(date +%d-%m-%y_%H-%M').tgz -C /tmp/ bfdata/
if [ $? -ne 0 ]
then
discord_log "There was an error when compressing database files."
exit
done

# Copy zip file to volume /backup
cp ./bfdata_backup_* /backup
if [ $? -ne 0 ]
then
discord_log "There was an error when sending the backup tarball to volume storage.
Check storate capacity."
exit
done

```

If we need to restore the database back to an earlier state, we have to use the restore.sh script. This script is run on the backupserver. First, open the volume where the backups are stored. When you call the restore script, you must include the timestamp of your chosen backup as a parameter in the exact format as it is written in the backup filenames. Once started, the script will first unzip the data(because the backup-data is already in zipped state and need to be unzipped for the database to use it), wich is the backup that will be restored. The script assumes that the volume is already mounted. The backup-server then ssh into the database and kills the cockroachdb process. The backup-data is copied from the backup-server to the database. When this is done, cockroach is restarted, and checkDbStatus.sh is called. This will turn on the cockroach-database, if it is not already up.

```

restore.sh
#!/bin/bash
dbIp="192.168.130.246"
ssh -o "StrictHostKeyChecking no" ubuntu@$dbIp exit

# Unzip data
rm -r /tmp/bfdata
tar xvzf /backup/bfdata_backup_$1.tgz -C /tmp

# Turn off cockroach database
pid=$(ssh root@$dbIp ps aux | grep "cockroach" | grep -v "grep" | awk '{print $2}')

```

```
| head -1)
ssh root@$dbIp kill $pid

# Copy Data to database
ssh root@$dbIp rm -r /bfdata #!!!!!!! <- DELETES CURRENT DATABASE! BIG DANGER
scp -r /tmp/bfdata root@$dbIp:/bfdata

# Turn on cockroach database
ssh -q root@$dbIp bash
/home/ubuntu/dcsg2003/configuration/database/checkDbStatus.sh
```

Manager back up list

We have multiple scripts and important files on manager. Most of these is already in a manager-folder in our Git-repo for the course. This files include:

- aliasSetter.sh
- backupDatabase.sh
- cron (a file for managing what to add in the servers crontab)
- dbClusterSize.sh
- downDetector.sh
- getIpList.sh
- makeClusterCommands.sh
- pollGit.sh
- production.sh

In addition to this, we also have DCSG2003_V23_group45-openrc.sh and a ./ssh folder with the managers ssh-keys. These are currently not in our git-repo.

Week 9 - Database Clusters

1) Clusters vs Replicas

The difference between a databasecluster and a replicated database is the servers role.

With replication, we have typical one main-databaseserver(master), and multiple worker-databaseservers(slaves). The webserver is able to read and write to the main-server, but will only be able to read the workers. The main will push updates to the workers regularly. This will make a robust setup, where the workers can be read even though main is down. We will also get increased performance, since we distribute the load between multiple servers. Backup can also be runned on a worker.

The negative aspects by using this setup, is the fact that all writing have to be done at the same place(main-databaseserver). The workers also have to be constantly synchronized with main, to make sure of always up-to-date information at bookface.

The databasecluster-setup will consist of multiple, equal databaseservers. Each of these will look like a "main" from the outside, but there will be one intern boss anyway. This group of equal servers is called a cluster. It is this setup we have on bookface. Databaseclusters have higher requirements for response time than replication. Clusters also has greater availability, since the webserver is able to write to all the databases, not just one.

2) Start new bookface architechture

Prework) Create new VM's

We start by creating three servers, which will be our databasecluster:

ID	Name	Status	Networks	Image	Flavor
ec25d3a5-7add-41f0-bd86-07cfaea5423	server3	ACTIVE	imt3003=192.168.134.43	Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64	gx3.4c4r
7fedf1d1-4384-469f-ad76-c22138610b3e	server2	ACTIVE	imt3003=192.168.130.160	Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64	gx3.4c4r
ebb925aa-41c3-467b-97ea-a54772e37640	server1	ACTIVE	imt3003=192.168.133.95	Ubuntu Server 22.04 LTS (Jammy Jellyfish) amd64	gx3.4c4r

These servers are now our "Docker swarm". We have to synchronuize the clocks on the servers regularly, because cluster is very vulnerable to time difference. We do this by installing NTP:

```
apt-get install -y ntpdate
```

Then we can sync the clocks with a sentral server, like justervesenet.no. We put this code in the servers crontab:

```
*/10 * * * * root ntpdate -b ntp.justervesenet.no
```

1) Install cockroach

We install cockroackDB on all the servers:

```
wget https://binaries.cockroachdb.com/cockroach-v22.2.5.linux-amd64.tgz
tar xzf cockroach-v22.2.5.linux-amd64.tgz
cp cockroach-v22.2.5.linux-amd64/cockroach /usr/local/bin
mkdir /bfdata
```

2) Start database cluster

We start cockroach on all the three servers:

```
#Server 1
cockroach start --insecure --store=/bfdata --listen-addr=0.0.0.0:26257 --http-addr=0.0.0.0:8080 --background --
join=192.168.133.95:26257,192.168.130.160:26257,192.168.134.43:26257 --advertise-addr=192.168.133.95:26257 --max-offset=1500ms

#Server 2
cockroach start --insecure --store=/bfdata --listen-addr=0.0.0.0:26257 --http-addr=0.0.0.0:8080 --background --
join=192.168.133.95:26257,192.168.130.160:26257,192.168.134.43:26257 --advertise-addr=192.168.130.160:26257 --max-offset=1500ms

#Server 3
cockroach start --insecure --store=/bfdata --listen-addr=0.0.0.0:26257 --http-
```



```
addr=0.0.0.0:8080 --background --
join=192.168.133.95:26257,192.168.130.160:26257,192.168.134.43:26257 --advertise-
addr=192.168.134.43:26257 --max-offset=1500ms
```

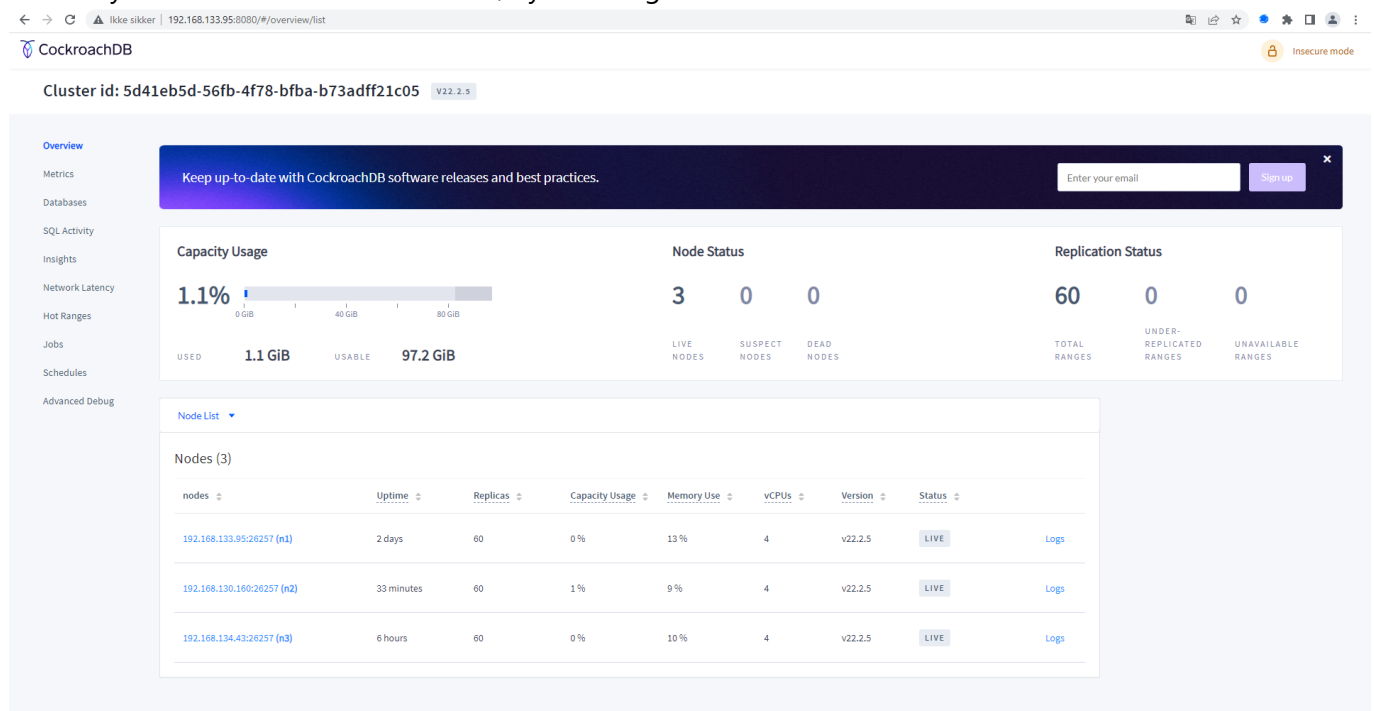
The difference between the three codes above, is the advertise-address. This ip-address is the address to the server where the code is running. The max-offset option is for increased tolerance for time offset between the databases. The default is 500ms.

3) Initialize database cluster

Then we initialize the cluster, by running this command on server 1:

```
cockroach init --insecure --host=192.168.133.95:26257
```

We verify that the database is initialized, by watching the cockroachDB dashboard:



The ip-address in the picture is for server 1, but all of them will work. We can see that all of the nodes in the cluster is alive.

4) Create the bookface database with the new design

First, we start a sql session on server 1:

```
cockroach sql --insecure --host=localhost:26257
```

Then we run following sql code:

```
CREATE DATABASE bf;
CREATE USER AETAAdmin;
GRANT ALL ON DATABASE bf TO AETAAdmin;

USE bf;

CREATE TABLE users (
  userid INT NOT NULL DEFAULT unique_rowid(),
  name STRING(50) NULL,
  picture STRING(300) NULL,
  status STRING(10) NULL,
  posts INT NULL,
  comments INT NULL,
  lastpostdate TIMESTAMP NULL DEFAULT now()::TIMESTAMP,
  createdate TIMESTAMP NULL DEFAULT now()::TIMESTAMP,
  FAMILY "primary" (userid, posts, comments, lastpostdate),
  FAMILY "secondary" (name, picture, status, createdate)
);

CREATE TABLE posts (
  postid INT NOT NULL DEFAULT unique_rowid(),
  userid INT NOT NULL,
  text STRING(300) NULL,
  name STRING(150) NULL,
  postdate TIMESTAMP NULL DEFAULT now()::TIMESTAMP,
  INDEX posts_auto_index_posts_users_fk (userid ASC),
  FAMILY "primary" (postid, userid, text, name, postdate)
);

CREATE TABLE comments (
  commentid INT NOT NULL DEFAULT unique_rowid(),
  userid INT NOT NULL,
  postid INT NOT NULL,
  text STRING(300) NULL,
  postdate TIMESTAMP NULL DEFAULT now()::TIMESTAMP,
  INDEX comments_userid_idx (userid ASC),
  INDEX comments_postid_idx (postid ASC),
  FAMILY "primary" (commentid, userid, postid, text, postdate)
);

CREATE table config ( key STRING(100), value STRING(500) );
create index on users (name);
create index on users (userid);
create index on users (lastpostdate);
create index on posts (userid);
create index on posts (postdate);
create index on comments (postid,postdate);
create index on comments (postid);
```

5) Configure auto-start on reboot

To make sure the databases starts automatically when one or multiple of the servers is rebooting, we have a `boot.sh` script which will run every time the server is rebooting, by using `crontab`, like this:

```
# Crontab file for worker nodes, is automatically updated
* * * * * crontab < /home/ubuntu/dcsg2003/configuration/worker/cron
@reboot bash /home/ubuntu/dcsg2003/configuration/worker/boot.sh
*/10 * * * * root ntpdate -b ntp.justervesenet.no
```

`Boot.sh` look like this:

```
#!/bin/bash
source /home/ubuntu/dcsg2003/configuration/base.sh

ownIp=$(hostname -I | awk '{ print $1 }' | xargs)

# Mount volume if not already
mount -t glusterfs $ownIp:bf_config /bf_config
mount -t glusterfs $ownIp:bf_images /bf_images

chmod 777 /bf_config
chmod 777 /bf_images

sleep 5

# Check if mount is successful
diskImages="bf_images"
diskConfig="bf_config"

if [ $(df -h | grep "$diskImages" | wc -l) -ne 0 ]
then
    echo "$diskImages is mounted"
else
    echo "$diskImages is not mounted"
    discord_log "Failed to mount disk."
    exit
fi

if [ $(df -h | grep "$diskConfig" | wc -l) -ne 0 ]
then
    echo "$diskConfig is mounted"
else
    echo "$diskConfig is not mounted"
    discord_log "Failed to mount disk."
    exit
fi

# Start cockroach db
```

```
bash /home/ubuntu/startdb.sh

# Start Docker
systemctl start docker
```

This script starts cockroachdb, by running another script we have in another path of the file-system. The startdb.sh script is executed at the end of the script above.

The startdb.sh script consist of this code, where the advertise-address is different for each server:

```
cockroach start --insecure --store=/bfdata --listen-addr=0.0.0.0:26257 --http-addr=0.0.0.0:8080 --background --join=192.168.133.95:26257,192.168.130.160:26257,192.168.134.43:26257 --advertise-addr=192.168.134.43:26257 --max-offset=1500ms
```

NOTE: The right privileges for boot.sh and startdb.sh is already given in another script we have; chmodder.sh. This script look like this, and is run for every knew instance we create:

```
#!/bin/bash
files=$(find ../ -name "*.sh")

for file in $files
do
    chmod +x $file
    echo "Updated permission for $file"
done
```

This will make every .sh file executable on the system.