

Group 45 DCSG2003 Oblig 3

Raphael Storm Larsen, Sara Stentvedt Luggenes

Week 10 - Gluster File Storage

Gluster installation

Task 1

We start by installing Gluster on all three servers:

```
apt-get -y install glusterfs-server glusterfs-client  
systemctl enable glusterd  
systemctl start glusterd
```

Task 2

Then we make the bricks(byggekløsser). These bricks is just folders for the distributed storage, and in the real world it should have been real harddrives. We use ordinary folders since we use VMs, and dont want to use a lot on resources on volumes. We make a folder where the images in bookface is stored, and another folder for configuration(we do this on all three servers):

```
mkdir /bf_brick  
mkdir /config_brick
```

Then we make two folders, which will be the place we "mounts" the distributed filesystems. These are the folders we are gonna use(we do this on all three servers):

```
mkdir /bf_images  
mkdir /bf_config
```

Task 3

Then we have to add two of the servers to the cluster, we do this commands on server 1:

```
gluster peer probe 192.168.134.194  
gluster peer probe 192.168.130.56
```

The ip-addresses above is for server 2 and server 3.

We make sure the commands above are successful by using the command `gluster peer status` on server 1. This will give the following output, which verifies that both server 2 and server 3 is added to the cluster:

```
# OUTPUT
Number of Peers: 2

Hostname: 192.168.134.194
Uuid: beba9453-41df-44c9-9581-d9253bf3eef2
State: Peer in Cluster (Connected)

Hostname: 192.168.130.56
Uuid: 4fef7622-32ad-4ea1-bde3-2bab6e71ac6f
State: Peer in Cluster (Connected)
```

Task 4

Now we create the two volumes to be distributed. We call one volume "bf_images" and the other one "bf_config". The following commands have to be done at only one server, we do it on server 1:

```
gluster volume create bf_images replica 3 192.168.134.127:/bf_brick
192.168.134.194:/bf_brick 192.168.130.56:/bf_brick force
gluster volume start bf_images
gluster volume create bf_config replica 3 192.168.134.127:/config_brick
192.168.134.194:/config_brick 192.168.130.56:/config_brick force
gluster volume start bf_config
```

The ip-addresses above is for server 1, server 2 and server 3. The data in bf_images and bf_config is gonna be replicated 3 times, one for each server. This is not space efficient, but in environments where high-availability and high-reliability are critical, it is the most optimal solution.

Task 5

To check whether the volumes now is a part of the file system, we run the command `df -h` on server 1:

Filesystem	Size	Used	Avail	Use%	Mounted on
tmpfs	393M	1.4M	392M	1%	/run
/dev/vda1	39G	16G	23G	42%	/
tmpfs	2.0G	0	2.0G	0%	/dev/shm
tmpfs	5.0M	0	5.0M	0%	/run/lock
/dev/vda15	105M	5.3M	100M	5%	/boot/efi
tmpfs	393M	4.0K	393M	1%	/run/user/1000

We have created the folders bf_images and bf_config, but they does not appear in the list above. This is because we have not mounted the folders/volumes to the servers yet. This have to be done on every server, included server1.

To make the folders `bf_images` and `bf_config` available on each server, we have to run two mount commands on each server:

```
mount -t glusterfs serverX:bf_config /bf_config
mount -t glusterfs serverX:bf_images /bf_images
```

NOTE: The `serverX` part in the commands above have to be changed to the servers own ip-address. With this means; 192.168.134.127 on server1, 192.168.134.194 on server2, and 192.168.130.56 on server3.

After we have run the mount commands on a server, we can verify that the folders is indeed mounted on the server by running this command again: `df -h`. This give us following output:

Filesystem	Size	Used	Avail	Use%	Mounted on
tmpfs	393M	1.4M	392M	1%	/run
/dev/vda1	39G	16G	23G	42%	/
tmpfs	2.0G	0	2.0G	0%	/dev/shm
tmpfs	5.0M	0	5.0M	0%	/run/lock
/dev/vda15	105M	5.3M	100M	5%	/boot/efi
192.168.134.127:bf_config	39G	17G	23G	43%	/bf_config
192.168.134.127:bf_images	39G	17G	23G	43%	/bf_images
tmpfs	393M	4.0K	393M	1%	/run/user/1000

Note that the `bf_config` and `bf_images` folders is now appearing on the list. This means that the folders/volumes is succcessfully mounted. The output above is from server1, and the ip-address listed is therefore 192.168.134.127. On the other servers, the ip-address will be the current servers ip-address. Everything that is saved in the folders on one server, will become available on the other servers as well.

To make the containers be able to save files to the folders, we have to give the folders the correct rights:

```
chmod 777 /bf_config
chmod 777 /bf_images
```

Task 6

When a server is rebooting, we have to make sure that the disks are mountet correctly on the server. We have made a script called `boot.sh` that will take care of the mounting on reboot. The script is run automatically on every reboot, because of crontab. Below is the crontab file we have on our three servers:

```
# 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
```

```
* * * * * cp /home/ubuntu/dcsg2003/configuration/worker/haproxy.cfg
/home/ubuntu/bookface/haproxy.cfg
* * * * * cp /home/ubuntu/dcsg2003/configuration/worker/docker-compose.yml
/home/ubuntu/bookface/docker-compose.yml
```

We can see in the crontab above, that a script called boot.sh is run on every rebooting on a server. The script boot.sh is shown below:

```
#!/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
yes | bash /home/ubuntu/startdb.sh

# Start Docker
systemctl start docker
```

The script `boot.sh` will try to mount the disks if they are not already mounted (probably not). Then the script will sleep for 5 seconds, to give the commands time to finish. Then an if-sentence will check whether the volumes are mounted or not, and send a message to our Bookface Discord-webhook. After the disks are mounted (or not), `cockroach` will start. In the case of a failed mount, this script will simply exit without attempting again. This was done to avoid infinite looping, and in the case that the mount has failed, we suspect that direct involvement is necessary anyways. As such it's easier to simply have the script ping our discord in the case of an error.

Week 11 - Docker Swarm

Task 1

Install Docker on all the servers with these commands:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
apt-get update
apt-get install -y apt-transport-https ca-certificates curl software-properties-
common python3-openstackclient python3-octaviaclient docker-ce git
```

To make sure that we are able to download Docker images from other places, we have to add the code below to the file `/etc/docker/daemon.json` on every server:

```
{
  "insecure-registries":["192.168.128.23:5000"]
}
```

Then restart Docker: `systemctl restart docker`

Task 2

Docker starts as a service automatically every time the server is rebooting. We don't want this, because we don't want a server to run any containers before Gluster is connected properly. We therefore have to turn off the automatic restart of Gluster when the server is rebooting. We do this by running these commands:

```
systemctl disable docker
systemctl disable docker.service
systemctl disable docker.socket
```

We add this line of code in the `boot.sh` script mentioned above:

```
systemctl start docker
```

This line will start Docker every time the script is run, but after the volumes are mounted and everything is ok.

Task 3 - Make the Docker swarm

Now we are ready to create the swarm. We run this code on server 1: `docker swarm init`. Server 1 is now going to be the manager in the swarm, server 2 and server 3 is going to be the workers. When we run the `docker swarm init` command, we get the following output(on server 1):

```
docker swarm join --token SWMTKN-1-
5msr1ct8s2de7vrwkj9c2zzcx50gvvg1e1ty8vvc10eiyarfic-6x253ue84qc8psdwqjnacuwl1
192.168.134.127:2377
```

This code we run on server 2 and server 3. This will make the other servers join the swarm, with the role of workers.

Task 4 - Download Bookface repo

Now we need to download the Bookface git-repo on server 1:

```
git clone https://github.com/hioa-cs/bookface.git
```

Task 5 - Make a haproxy file to the database

Then we have to make a haproxy file in the bookface folder on server 1:

```
cd bookface
nano haproxy.cfg
```

In the haproxy file we write this:

```
global
    log            127.0.0.1 local2

    pidfile        /tmp/haproxy.pid
    maxconn        4000

defaults
    mode            http
    log             global
    option          httplog
    option          dontlognull
    option http-server-close
    option forwardfor except 127.0.0.0/8
    option          redispatch
    retries         3
```

```

        timeout http-request      10s
        timeout queue             1m
        timeout connect           10s
        timeout client            1m
        timeout server            1m
        timeout http-keep-alive 10s
        timeout check             10s
        maxconn                   3000

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

frontend db
    bind *:26257
    mode tcp
    default_backend databases

backend databases
    mode tcp
    balance roundrobin
    server server1 192.168.134.127:26257
    server server2 192.168.134.194:26257
    server server3 192.168.130.56:26257

```

Task 6 - Start bookface

Bookface is ready to get started on server 1:

```
docker stack deploy -c docker-compose.yml bf
```

It take some time for the images to download, run this command to see the status:

```
docker service ls
```

When the output looks like the output below, the images is downloaded correctly:

ID	NAME	MODE	REPLICAS	IMAGE
7oq1am7tbwqq	bf_db_balance	replicated	1/1	
192.168.128.23:5000/haproxy:latest				*:1936->1936/tcp
xv2xl7xg4iqr	bf_memcache	replicated	1/1	
192.168.128.23:5000/memcached:latest				

```
4vef6mk83t2r    bf_web                replicated    3/3
192.168.128.23:5000/bf:v17              *:80->80/tcp
```

Now bookface should be operative. Check with the command below whether we get some HTML output or not:

```
curl http://localhost
```

The output looks like this, and we can verify that bookface is indeed operative.

Output abbreviated...

```
</tr>
<tr>
  <! userline: 855878343670792193>
  <td class=row ><a href='/showuser.php?user=855878343670792193'><img
src='/images/1veBXrMLr36Wme9FIPwubGgJcgXiTw.jpg'></a></td>
  <td class=row ><a href='/showuser.php?user=855878343670792193'>Canaan
Lamb</a></td>
  <td class=row >0</td>
</tr>
<tr>
  <! userline: 855877007372419073>
  <td class=lightrow ><a href='/showuser.php?user=855877007372419073'><img
src='/images/6ym3wVvuAKwdLeIuLbkqgm09T5tQyq.jpg'></a></td>
  <td class=lightrow ><a href='/showuser.php?
user=855877007372419073'>Brinley Barber</a></td>
  <td class=lightrow >0</td>
</tr>
</table>
load time: 0s

</html>
```

Task 7 - Database migration

Now we will import the data from our current bookface setup, to the new setup. First we have to fix a small issue from week 9. The table "posts" did not include a column for pictures, this we have to fix. Go to the database in the new cluster(on server 1):

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

We delete the old table called posts, and make a new table with an extra column for pictures:


```
use bf;

drop table posts;
CREATE table posts ( postID INT PRIMARY KEY DEFAULT unique_rowid(), userID INT,
text STRING(300), name STRING(150), image STRING(32), postDate TIMESTAMP DEFAULT
NOW());
GRANT SELECT,UPDATE,INSERT on TABLE bf.* to bfuser;
```

Now we have to make it not possible to get a bookface to import from another solution without a "password". We write this in the cockroach shell:

```
insert into config ( key, value ) values ( 'migration_key', 'mjau' );
```

In the command above, "mjau" is our secret password. A bookface will not be able to import data from another solution without this key.

Then log out from the cockroach shell.

It is time to upgrade the current bookface version to version 17. Run this code on server1:

```
docker run -d --name=bf-import-helper -p 20080:80 -e BF_DB_HOST=192.168.130.246 -e
BF_DB_PORT=26257 -e BF_DB_USER=AETAadmin -e BF_MEMCACHE_SERVER=192.168.130.6:11211
-e BF_DB_NAME=bf 192.168.128.23:5000/bf:v17
```

In the code above, BF_DB_HOST means old database ip address, BF_DB_USER means username on old database, BF_MEMCACHE_SERVER means memcache server ip address, and BF_DB_NAME means old database name.

Now verify that the docker container started with `docker ps`, and run this command to verify that the old bookface still is up:

```
curl localhost:20080
```

After this, we are now ready to import the old bookface database to the new gluster cluster. Do this on server 1:

```
curl -s "http://localhost/import.php?entrypoint=192.168.134.127&key=mjau"
```

In the command above, entrypoint means the ip address to where the help-container is running, in this case on server 1. Key is the secret password we made earlier, mjau.

Now, the bookface data is gonna be replicated to all three server thanks to GlusterFS. This command is gonna take a while, probably half an hour or so.

Task 8 - Openstack balancer

Now we have to stop using the old balancer server, and use the balancer functionality on Openstack instead. This is for efficiency and simplicity. If we still use the old balancer server, bookface is gonna shut down if the balancer shuts down. We avoid this by using the openstack balancer.

Kopier først "RC"-filen fra manager til server1 med scp. Inne i server1 må dere "source" filen og sjekke at alt er i orden med kommandoen:

Do this on manager: Copy the RC file from manager to server 1:

```
scp DCSG2003_V23_group45-openrc.sh ubuntu@10.212.173.5:/ubuntu
```

Source the file with this command on server 1:

```
source DCSG2003_V23_group45-openrc.sh
```

Verify that everything is ok with: `openstack server list`. If the command exist, then it is ok.

Then we make a loadbalancer with openstack(on server 1):

```
openstack loadbalancer create --name bflb --vip-subnet-id c3ea9f88-8381-46b0-80e0-910c676a0fbd
openstack loadbalancer show bflb
openstack loadbalancer listener create --name bflistener --protocol HTTP --protocol-port 80 bflb
openstack loadbalancer pool create --name bfpool --lb-algorithm ROUND_ROBIN --listener bflistener --protocol HTTP
openstack loadbalancer member create --address 192.168.134.127 --protocol-port 80 bfpool
openstack loadbalancer member create --address 192.168.134.194 --protocol-port 80 bfpool
openstack loadbalancer member create --address 192.168.130.56 --protocol-port 80 bfpool
```

We should make a new floating ip to the balancer, so we can go live with our new bookface. We make a floating ip with the command line like this:

```
openstack floating ip create --description bf_swarm_ip 730cb16e-a460-4a87-8c73-50a2cb2293f9
```

The output we get is:

```
FLOATING_IP_ID: 5337587f-47b4-4d98-922e-981ed2231c28
```

Now we go back to the load balancer and finds the id below vip_port_id:

```
ubuntu@manager:~$ openstack loadbalancer show bflb
```

Field	Value
admin_state_up	True
availability_zone	None
created_at	2023-03-21T19:24:46
description	
flavor_id	None
id	1a971daf-0e20-457e-8a91-f8e7426da4ac
listeners	82440adf-195c-4a65-96ac-fc20fa7d7b59
name	bflb
operating_status	ONLINE
pools	24792824-25e5-4fc7-acef-d750681b2df5
project_id	4661480d37154198b27092c5c15a765c
provider	amphora
provisioning_status	ACTIVE
updated_at	2023-03-28T12:30:25
vip_address	192.168.132.39
vip_network_id	35fe8b76-cc31-4b48-b3f0-5c48eef8d289
vip_port_id	255d0c0d-5e31-4c6b-9e46-c806663fa748
vip_qos_policy_id	None
vip_subnet_id	c3ea9f88-8381-46b0-80e0-910c676a0fbd
tags	

The vip_port_id is 255d0c0d-5e31-4c6b-9e46-c806663fa748.

We connect the floating_ip_id from above and the vip_port_id together:

```
openstack floating ip set --port 255d0c0d-5e31-4c6b-9e46-c806663fa748 5337587f-47b4-4d98-922e-981ed2231c28
```

Task 9 - Go live!

We change the floating ip address the uptime challenge should be registered with. The ip address below is the floating ip to the openstack load balancer:

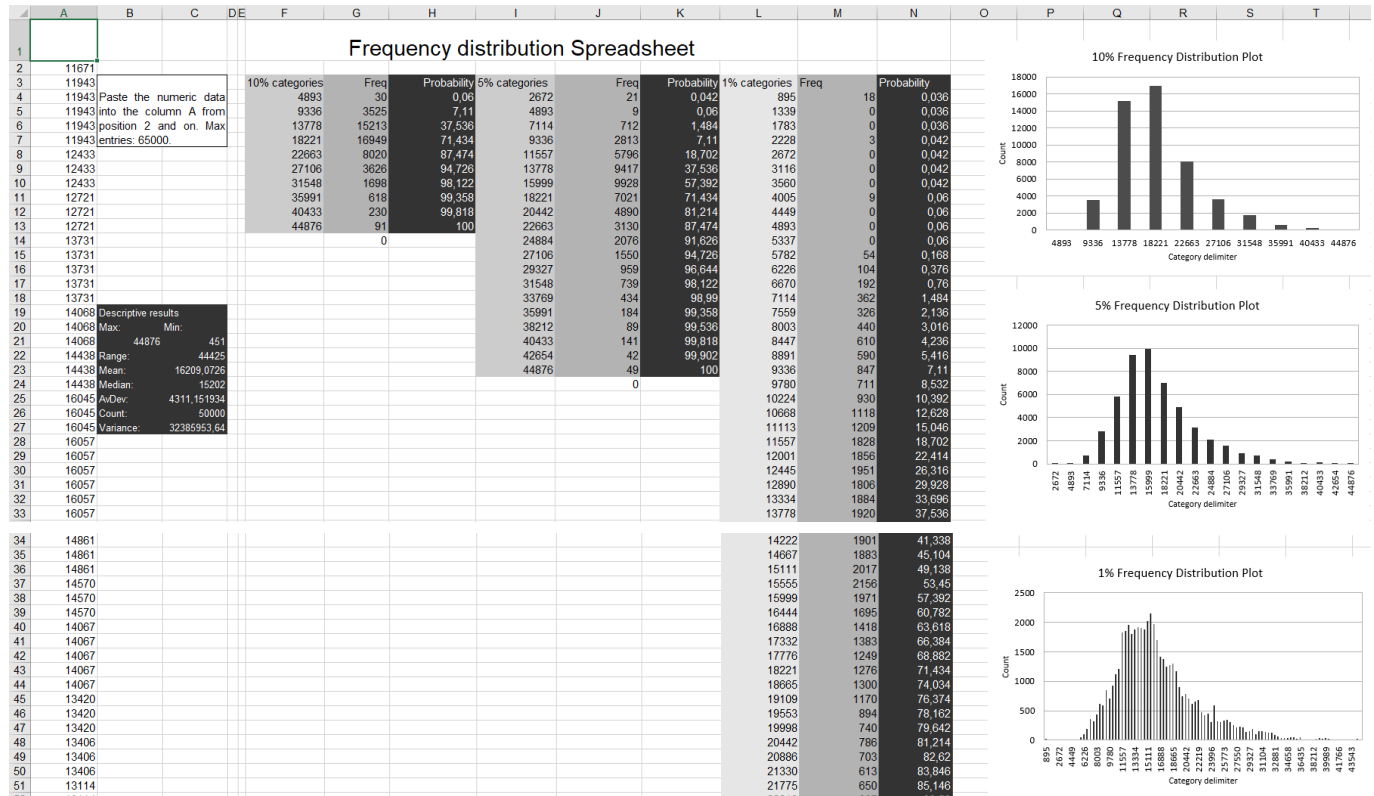
```
uc set endpoint 10.212.175.95
uc status
```

uc traffic on

Week 12 - Capacity calculation through statistical distributions

Task 1

50k.dat



- By analysing the images above, there is a large range in how many users the game is hosting throughout the given timeperiod. It is a range of 44425 users, from most to least users at a given time, where the maximum value is 44876 and the minimum value is 451. The value 451 is probably indicating some sort of downtime for the game, while the value 44876 tells us how many users the servers should be able to host at any given time, since this is the extreme end of the scale. By looking at the 5% table in the middle of the document, we can see that there is 29 000 users in the game or less, in 95% of the time. In 5% of the time, there is approximately 30 000 or more users, which make this large number of users pretty rare.

Begin.dat

End.dat

2. By looking at the descriptive results in the images above, we can see that the popularity of the game probably have increased over the timeperiod. The mean amount of users at the beginning is 13139, and the mean amount of users at the end is 18822. The range, minimum value and maximum value also confirms this hypothesis.
3. Based on the data given in the begin.dta and end.dta, the popularity of the game have increased with approximately $18822 - 13139 = 5683$ users over the timeperiod.

Task 2

We run the following command and get the following output in JSON format:

```
curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}'`

{"status":"success","data":{"resultType":"vector","result":[{"metric":
{"__name__":"last_download_time","instance":"192.168.129.65:9001","job":"mongodb",
"name":"AETA","tags":"dcsg2003_23"},"value":
[1679759424.672,"11.698165893554688"]}]]}}
```

We install the package jq to be able to make the output above look nicer:

```
apt-get install -y jq
```

The we run the command again, and pipeline it to the jq package. The output is as follows:

```
curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}' | jq
```

Output:

```
{
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {
          "__name__": "last_download_time",
          "instance": "192.168.129.65:9001",
          "job": "mongodb",
          "name": "AETA",
          "tags": "dcsg2003_23"
        },
        "value": [
          1679759761.415,
          "9.776767015457153"
        ]
      }
    ]
  }
}
```

To get the download time in a single integer, we run the command again as follows:

```
curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}' | jq -r '.data.result[].value[1] '

# OUTPUT

9.776767015457153
```

We then download the `frontpage_scale.sh` script, and added some configuration code:

```
#!/bin/bash
source /home/ubuntu/DCSG2003_V23_group45-openrc.sh
source /home/ubuntu/dcs2003/configuration/base.sh

# Name of company
COMPANY_NAME="AETA"

# IP address of server 1 or whoever is the docker swarm leader
SERVER_IP="192.168.134.127"

# What is the lowest desired number of frontpage users?
FRONTPAGE_COUNT_LIMIT=100

# What increments do we increase in?
SCALE_UP_INCREMENT=25

# What increments do we decrease in?
SCALE_DOWN_INCREMENT=50

# What is the lower download time limit we would like to stay above?
DOWNLOAD_TIME_LOWER_THRESHOLD=1.9

# What is the upper download time limit we would like to stay below?
DOWNLOAD_TIME_UPPER_THRESHOLD=10.0

# SAFETY VALVE: Set this to "0" if you want the scaling to actually take place
SAFETY_VALVE=0

#####
# SAFETY CHECK: Is bc installed?

if ! which bc > /dev/null; then
echo "You need to install the package 'bc' first"
exit 1
fi

#####
# Define function for scaling

function scale {
    # This script is assumed to run on the manager and SSH to a server
    # in order to run the docker service upgrade command.
```

```

# we also assume that we need to use sudo.

COMMAND="sudo docker service update --env-add BF_FRONTPAGE_LIMIT=$1 bf_web"

# This could be improved, because what happens when server 1 is unavailable?

SSH_COMMAND="ssh -t ubuntu@$SERVER_IP "

if [ "$SAFETY_VALVE" -eq "0" ]; then
    # Safety valve is off, we're running the command
    $SSH_COMMAND $COMMAND
    discord_log "Scaling to $1 frontpage users."

else
    # Safety valve is on, we only print what we would do
    echo "Safety valve is on, this is what would be executed: $SSH_COMMAND $COMMAND"
fi

}

#####
# Check if the site is up. Exit if it's down.

STATUS=$( curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_status{name="'$COMPANY_NAME'"}' | jq -r '.data.result[].value[1] ')

if [ "$STATUS" -gt "0" ]; then
    echo "Site is considered up"
else
    echo "Site is considered down, we should stop here"
    exit 1
fi

#####
# The site is up, so we can proceed with checking its performance
# Get current download times:
DOWNLOAD_TIME=$( curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="'$COMPANY_NAME'"}' | jq -r '.data.result[].value[1]
')
NUMBER_OF_FRONTPAGE_USERS=$( curl -s -g
'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=frontpage_count{name="'$COMPANY_NAME'"}' | jq -r '.data.result[].value[1] ')

echo "Observed download time: $DOWNLOAD_TIME"

# check if we are below the lower threshold. If we are, we scale up
if (( $(echo "$DOWNLOAD_TIME < $DOWNLOAD_TIME_LOWER_THRESHOLD" | bc -l) )); then
    NEW_FRONTPAGE_COUNT=$( echo "$NUMBER_OF_FRONTPAGE_USERS + $SCALE_UP_INCREMENT"
| bc )
    echo "Download time was lower, we have some capacity to spare. Scaling up to
$NEW_FRONTPAGE_COUNT"
    scale $NEW_FRONTPAGE_COUNT

```



```
# check if we are above the higher threshold. If we are, scale down, but not lower
than the limit
elif (( $(echo "$DOWNLOAD_TIME > $DOWNLOAD_TIME_UPPER_THRESHOLD" | bc -l) )); then

    # We can't go lower than the bottom
    if [ "$NUMBER_OF_FRONTPAGE_USERS" -eq "$FRONTPAGE_COUNT_LIMIT" ]; then
        echo "We should go lower, but we are already at the limit"
        exit 0
    fi

    # Lowering the number of frontpage users
    NEW_FRONTPAGE_COUNT=$(( echo "$NUMBER_OF_FRONTPAGE_USERS -
$SCALE_DOWN_INCREMENT" | bc ))

    if [ "$NEW_FRONTPAGE_COUNT" -lt "$FRONTPAGE_COUNT_LIMIT" ]; then
        echo "We should scale down, but can't go lower then the limit, so we end
up at $FRONTPAGE_COUNT_LIMIT"
        NEW_FRONTPAGE_COUNT=$FRONTPAGE_COUNT_LIMIT
    else
        echo "Scaling down to $NEW_FRONTPAGE_COUNT as new frontpage_limit"
        scale $NEW_FRONTPAGE_COUNT
    fi
fi
fi
```

This script scales the amount of frontpage users up and down, depending on factors such as download time, amount of frontpage users, and our own personal preferences. We add this script to crontab, and it is run every 15 minute. The way this script is configured above, will make bookface automatically add frontpage users by 25 if the download time is 1.9 seconds or less. If the download time is more than 5.0 seconds, bookface will automatically remove 50 frontpage users, and by this reduce the amount of frontpage users.

Task 3

We need to get the download time for bookface the last 24 hours. First, we run this command and get the data in this format(the command and output is for the last 1 hour only):

```
curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}[1h]' | jq

# OUTPUT

{
  "status": "success",
  "data": {
    "resultType": "matrix",
    "result": [
      {
        "metric": {
          "__name__": "last_download_time",
          "instance": "192.168.129.65:9001",
          "job": "mongodb",
```

```

        "name": "AETA",
        "tags": "dcsg2003_23"
    },
    "values": [
        [
            1679758369.944,
            "12.337580919265747"
        ],
        [
            1679758429.945,
            "12.337580919265747"
        ],
        [
            1679758489.944,
            "10.641442060470581"
        ]
    ]
}
]
}
}

# The output goes on but is abbreviated for this document

```

We need to further improve the output to make the data in a useful format. We type this command and get the data in a list of floats:

```

curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}[1h]' | jq -r '.data.result[] | .values[] | .
[1] '

# OUTPUT

8.502417087554932
8.502417087554932
8.502417087554932
11.44600796699524
11.44600796699524
11.44600796699524
11.44600796699524
11.44600796699524
11.44600796699524
11.44600796699524

# The output goes on but is abbreviated for this document

```

By changing [1h] in the command above to [24h], we can get the download time for bookface for the last 24 hours. We change this parameter, and pipeline the command to sed. Sed is used to get rid of the period in the floats, and change it to comma, which is the delimiter excel is using.

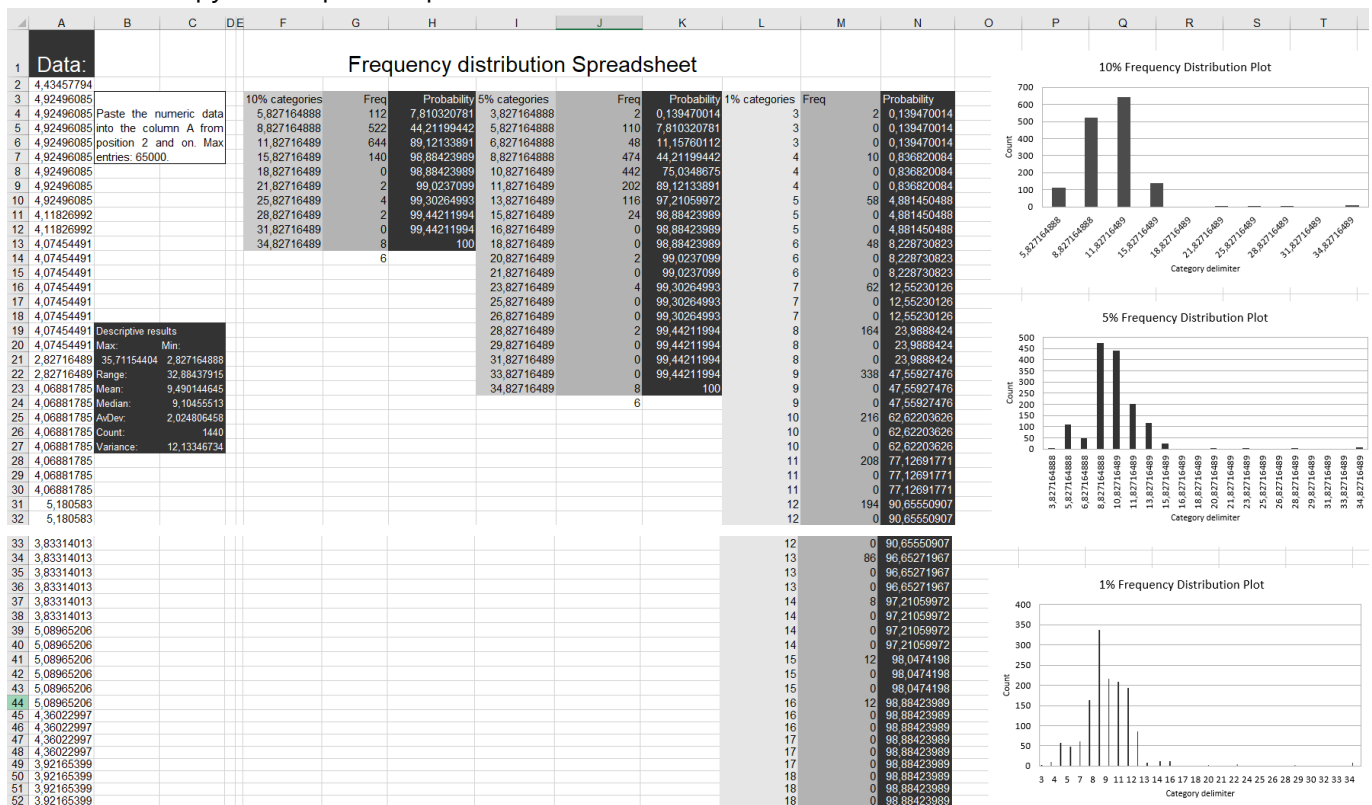
```
curl -s -g 'http://admin:admin@192.168.132.61:9090/api/v1/query?
query=last_download_time{name="AETA"}[24h]' | jq -r '.data.result[] | .values[] |
.[1] ' | sed -e 's/\\./,/'
```

OUTPUT

```
8,206752061843872
8,206752061843872
10,79725694656372
10,79725694656372
10,79725694656372
10,79725694656372
10,557878017425537
10,557878017425537
10,557878017425537
10,557878017425537
10,557878017425537
10,557878017425537
```

The output goes on but is abbreviated for this document

Now we can copy the output and paste it into the excel document used in task 1.



By looking at the images above, we can see that mean download time for our bookface is 9,5 seconds. The range is between 35,7 and 2,8 seconds. In 90 percent of the cases, our downloading time is 13 seconds or below, which can be seen in the 5% table in the middle of the picture. In 10% of the cases, our download time is greater than 13 seconds, in the range from 13 to 35 seconds.

Week 15: Consul Setup

- **IP:** 192.168.134.127

- **DATACENTRE:** AETA

```
docker run -d -v /opt/consul:/consul/data --name=consul --net=host -e
'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}' -e
'CONSUL_BIND_INTERFACE=ens3' 192.168.128.23:5000/consul:latest agent -server -
bind=192.168.134.127 --datacenter=AETA -client=0.0.0.0 -ui -bootstrap
```

```
docker run 192.168.128.23:5000/consul:latest join -http-addr=192.168.134.127:8500
-wan 192.168.128.23 192.168.134.127
```

```
curl -X PUT -d '{"ID": "SERVER.AETA", "Name": "bookface", "Address":
"192.168.132.39", "Port": 80 }' http://127.0.0.1:8500/v1/agent/service/register
```

Server 2

```
docker run -d -v /opt/consul:/consul/data --name=consul --net=host -e
'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}'
192.168.128.23:5000/consul:latest agent -server -join=192.168.134.127 --
datacenter=AETA -client=0.0.0.0 -bind=192.168.134.194
curl -X PUT -d '{"ID": "SERVER2.AETA", "Name": "bookface", "Address":
"192.168.132.39", "Port": 80 }' http://localhost:8500/v1/agent/service/register
```

Server 3

```
docker run -d -v /opt/consul:/consul/data --name=consul --net=host -e
'CONSUL_LOCAL_CONFIG={"skip_leave_on_interrupt": true}'
192.168.128.23:5000/consul:latest agent -server -join=192.168.134.127 --
datacenter=AETA -client=0.0.0.0 -bind=192.168.130.56
curl -X PUT -d '{"ID": "SERVER3.AETA", "Name": "bookface", "Address":
"192.168.132.39", "Port": 80 }' http://localhost:8500/v1/agent/service/register
```

At any server

```
dig @192.168.128.23 bookface.service.aeta.dcs2003. ANY
```

At any server, finds our architecture

```
dig @192.168.128.23 server1.node.aeta.dcs2003. ANY
```

Task 6

```
curl -X PUT -d '{"ID": "SERVER.AETAlocal", "Name": "bookfacelocal", "Port": 80 }'
http://127.0.0.1:8500/v1/agent/service/register
curl -X PUT -d '{"ID": "SERVER2.AETAlocal", "Name": "bookfacelocal", "Port": 80 }'
http://127.0.0.1:8500/v1/agent/service/register
curl -X PUT -d '{"ID": "SERVER3.AETAlocal", "Name": "bookfacelocal", "Port": 80 }'
http://127.0.0.1:8500/v1/agent/service/register
```