

## Repository

You may access our public GitHub repository for the script at [https://github.com/zhanfuer/Group10\\_ass2\\_owncloud](https://github.com/zhanfuer/Group10_ass2_owncloud)

## OwnCloud

Our group has decided to use OwnCloud as our web application in which we scale across multiple servers. OwnCloud is a file hosting platform: allowing users to upload, access and manage their files in the cloud.

### Task 1. Scaling OwnCloud to support for a large & changing user demand

Our concept (figure 1) includes several servers in order to scale for a large user demand.

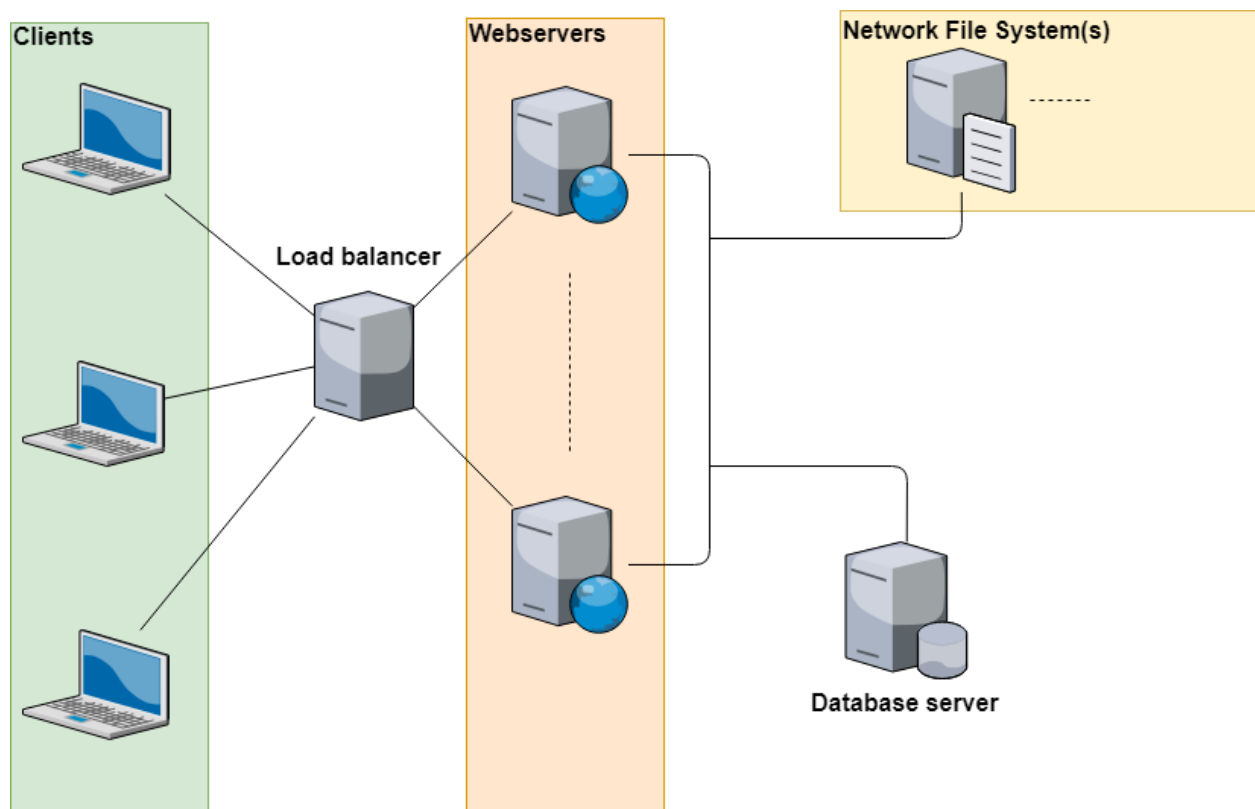


Figure 1: Concept

### Load Balancer

The load balancer balances the load between multiple servers. Clients would connect to the load balancer, and the load balancer would then forward them to a specific web server. Originally, one server was taking all the load. This would've been the first bottleneck when there are too many clients making requests to the web server.

The load balancer is configured using Nginx, using reverse proxies to distribute traffic across web servers. Nginx is also configured to use IP hashing (`ip_hash`), which means a client will be forward to the same web server until their current session ends. This is because each web server has its own set of client sessions and they are not shared between web servers.

### **Web Servers**

All web servers provide the same service. Each web server has Apache (with PHP) installed, and are configured to accept traffic only from the load balancer. Each web server connects to the network file system to access the files associated with OwnCloud. They also connect to the database server (when running OwnCloud).

The web servers are limited to accept only up to 10GB of data for file uploading.

### **Network File System(s)**

The network file system acts as a local file system, except it is used by multiple web servers. The network file system only has 50GB of space (the disks are SSDs for faster read and write performance), for now, however, this can be expanded easily in the future. The network file system is managed by GlusterFS. More network file systems can be added with ease so long they have *peered* with other network file systems.

The network file system's disk is formatted using XFS, this format allows for concurrent writing which will speed up file uploads.

### **Database Server**

The database server is configured with MySQL. A new database and user is created both called *owncloud*. The password is set to the same password provided when running the *run.sh* bash script.

### **Security**

Clients may only connect to the load balancer, from here the load balancer will then forward them to one of the web servers. The web servers then serve OwnCloud. The only security breach then onwards would have to be performed through OwnCloud.

In terms of DoS/DDoS protection limitations can be enforced to limit the number of requests a client can perform per second. This would be done using Nginx. More can be found [here](#).

## Automation

We have written Ansible scripts (playbooks) to automate the setup of a new cluster to run OwnCloud. This would speed up development when needing to deploy OwnCloud quickly to meet user demands or when updating OwnCloud. The playbooks are flexible to create a number of network file systems or a number of web servers for changing user demand.

### Running the automated script

1. Unzip this zip file (task1.zip).
2. Create a new instance on Google Cloud, name it 'controller', and tick 'allow full access to Google's APIs'.
3. Edit the newly created Google Cloud instance and add a public key for SSH. Connect to the new instance via SFTP.
4. Copy the folder for task one into your user directory.
5. Connect to the server using SSH, and go to your home directory and CD into the task one folder.
6. Run `bash run.sh {project-id} {password}`, replacing project-id with the project you created the instance in, and password being the password you want for your owncloud account and database.
7. Once the script has completed it should have created more instances. Using the lb01's external IP, go to `http://{ lb01 external IP here }/owncloud`

If you do not want to use SFTP, you can connect via SSH instead and clone our repository. Use the following steps instead (once the new instance is created):

1. Sudo apt-get install git
2. Git clone [https://github.com/zhfanrui/Group10\\_ass2\\_owncloud](https://github.com/zhfanrui/Group10_ass2_owncloud)
3. Bash run.sh {project-id} {password}

### **Task 3. What else would be important in a DevOps approach?**

We have decided to automate backing up the database along with restoring the database. Automating database restoration and backups accelerates deployment, and this is important because DevOps enforces more deployment than for example Agile Scrum software development. An example scenario would be if we were to update OwnCloud to a newly updated version, there could be a bug which causes data loss in the database. In this event first: we would need a backup of the database, and second: we would need to restore a previous version of the database. In addition, fix or roll back to a previous version of OwnCloud.

#### **How can it be implemented?**

In this task, it will call 2 task regularly (mysqldump, rsync).

**Mysqldump** is to export the data stored in the database. We write the command to backup it every 1 hour and it will be stored for 15 days and then deleted. Also, in order to optimize the disk space, we zip the sql file. This is just a demonstration of the config and it can be changed to whatever you want. Mysqldump is added to CRON to automatically perform the backups every 1 hour. This should be changed to something like eight hours, since doing it every hour would mean it will slow down the system.

**Rsync** is used to sync the folders, both for data folder and software folder. It will protect the security of data. If the main disk is broken, we can use the backup version to recover in order to get the minimal loss.

#### **What else could be done?**

What if we wanted to perform maintenance on one of the NFS servers, or what if one of the NFS servers went down? In this case, the playbook is flexible to allow for replicated volumes across many servers. This would mean that user data is still available even if one of the servers went down.

### **Instructions**

1. Clone all the repository (or task3 zip, and upload it using SFTP).
2. cd task3
3. bash run.sh [project\_id] [password] [db\_host] [nfs\_host]
4. It will create a new instance named backup01 and it will automatically backup.
5. To restore a backup, use restore.sh.

## Testing - Task 1

First, I set up an Ansible Controller instance. I name it 'controller', and allow it full access to the Google's API. I then download the contents from the GitHub repository ([https://github.com/zhfanrui/Group10\\_ass2\\_owncloud](https://github.com/zhfanrui/Group10_ass2_owncloud))

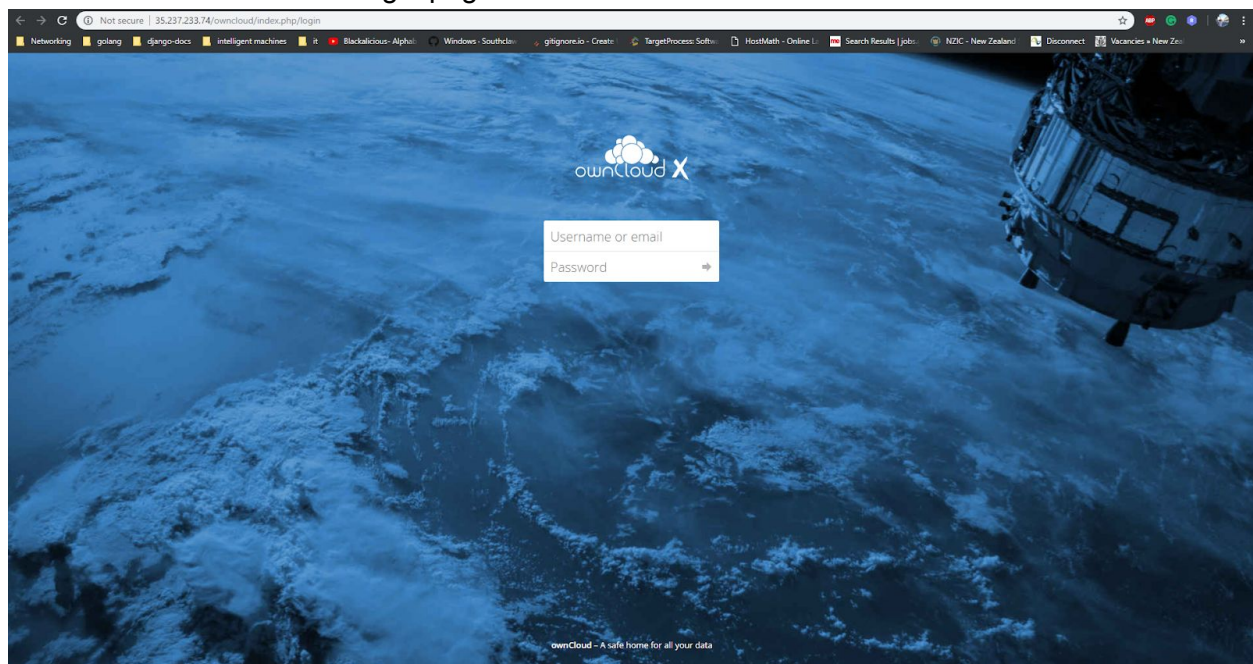
Second, I copy these contents over using SFTP to my user directory. I then run the run script using bash run.sh software-capstone testing, where software-capstone is my current project-id and testing will be my user password and database password.

The script should create the following instances,

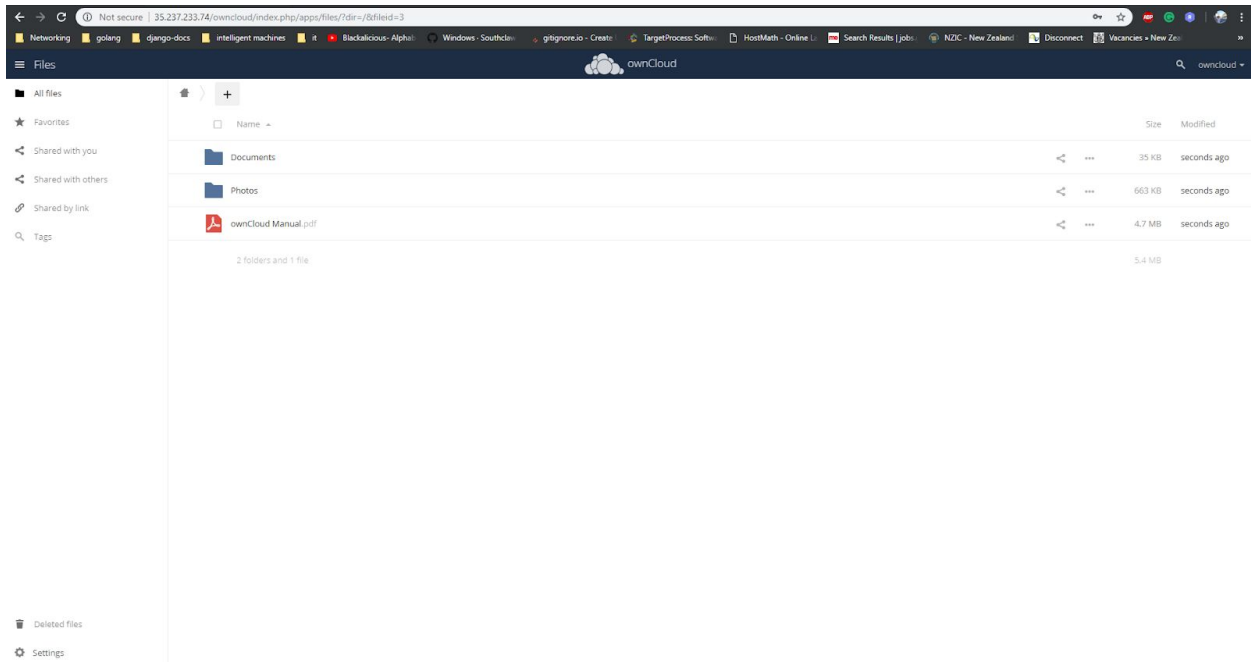
Name	Zone	Recommendation	Internal IP	External IP	Connect
<input checked="" type="checkbox"/> controller	us-east1-b		10.142.0.3 (nic0)	35.237.139.3	SSH
<input type="checkbox"/> db01	us-east1-b		10.142.0.4 (nic0)	35.227.62.176	SSH
<input type="checkbox"/> lb01	us-east1-b		10.142.0.8 (nic0)	35.237.233.74	SSH
<input type="checkbox"/> nfs01	us-east1-b		10.142.0.7 (nic0)	35.231.198.1	SSH
<input type="checkbox"/> ubuntu	us-east1-b		10.142.0.2 (nic0)	35.190.186.150	SSH
<input type="checkbox"/> ws01	us-east1-b		10.142.0.5 (nic0)	35.237.92.21	SSH
<input type="checkbox"/> ws02	us-east1-b		10.142.0.6 (nic0)	35.237.192.63	SSH

The controller instance was created before, the rest were created using the automated script.

Third, I visit <http://35.237.233.74/owncloud> where 35.237.233.74 is the IP address for the load balancer. I then land at the login page.



I login using owncloud as the username, and testing as the password.  
I then land at my user home,



I try to upload a file, first starting with 2KB using the plus icon. This works great! However, when I tried an 8.8MB file, it does not work. There were no errors in the error log for apache, nginx or owncloud. So I opened Chrome Developer Tools to debug.

/owncloud/core/css/images	GET	OK	http/1.1	http
WinSCP-5.13.4-Setup.exe /owncloud/remote.php/webdav	PUT	413 Request Entit...	http/1.1	http
notifications?format=json /owncloud/ocs/v2.php/apps/notifications/api/...	GET	200 OK	413 Request Entity Too Large	http
getstoragestats.php?dir=%2F /owncloud/index.php/apps/files/ajax	GET	200 OK	http/1.1	http

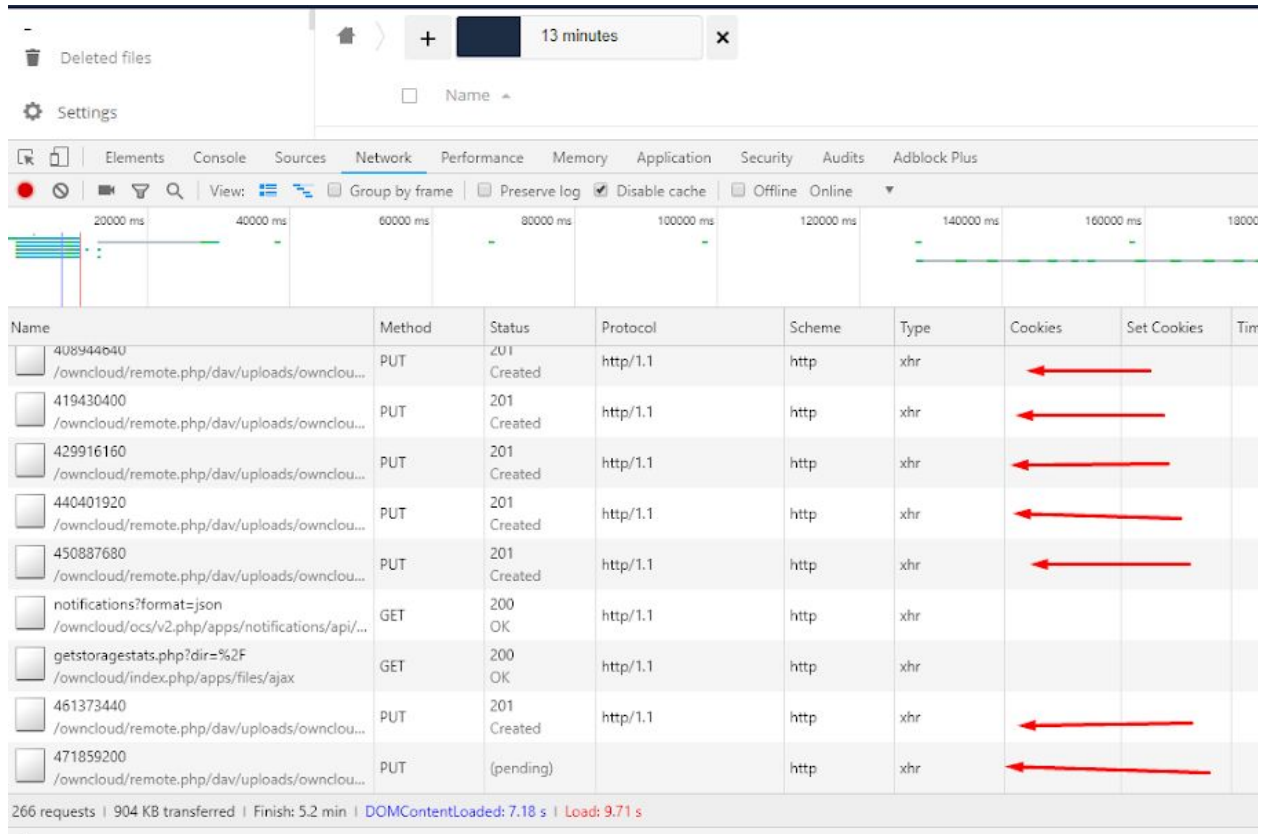
It seems we had HTTP error 413 returned. This meant that the PUT/POST data was too large and therefore I adjusted the nginx configuration and php.ini configuration to support a larger size.

```
server
{
    listen 80;
    server_name 35.231.198.1;

    location / {
        proxy_pass http://new_servers ;
        client_max_body_size 2G;
    }
}
```

This fixed the problem, and so Steven added the solution to the automated script. I then ran the test all over again and tried again.

I was then able to upload a 1.8GB zip file, which was split into partitions by the client.



I then download the file to test download. The speed was great because I have gigabit fibre in my flat too but also because the server was optimized using an SSD for the NFS with XFS file system for concurrent writing. I was able to download the file within 8 minutes (total size being 1.8GB).

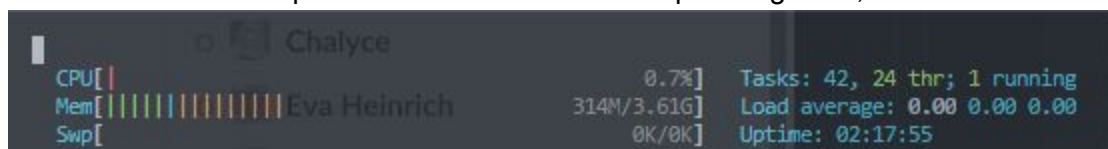
### Everything works in the frontend, great! What about the backend?

First, I SSH into nfs01 from the controller instance using ssh dy1zan@nfs01, where dy1zan was my username.

I then CD into /var/www/owncloud and chmod 777 the data folder so I can access it. Finally, CD into the data folder and into my user folder (owncloud), I can see my files.

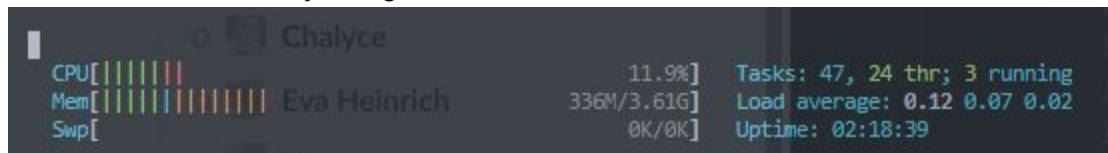
### Performance,

I have overlooked the performance for ws01 while uploading a file, the CPU is idle.





However ws02 is clearly doing work,



I then tested opening a new browser tab and going to the same address. My session is maintained and I am still logged in to OwnCloud.

I was not able to thoroughly test the load balancer since this would require many clients from different IP addresses. It balances based on IP since IP hashing was used.

### Testing - Task 3

I upload the task3 folder to my home directory using SFTP. I then CD into the task3 folder using SSH.

I then execute the command based on my instances that task 1 created,

	Name ^	Zone	Recommendation	Internal IP	External IP	Connect
<input type="checkbox"/>	ansible-controller	us-east1-b		10.142.0.4 (nic0)	104.196.125.83	SSH ▾ ⋮
<input type="checkbox"/>	db01	us-east1-b		10.142.0.3 (nic0)	35.237.139.3	SSH ▾ ⋮
<input type="checkbox"/>	lb01	us-east1-b		10.142.0.8 (nic0)	35.237.192.63	SSH ▾ ⋮
<input checked="" type="checkbox"/>	nfs01	us-east1-b		10.142.0.7 (nic0)	35.237.233.74	SSH ▾ ⋮
<input type="checkbox"/>	ubuntu	us-east1-b		10.142.0.2 (nic0)	35.190.186.150 ↗	SSH ▾ ⋮
<input type="checkbox"/>	ws01	us-east1-b		10.142.0.5 (nic0)	35.227.62.176	SSH ▾ ⋮
<input type="checkbox"/>	ws02	us-east1-b		10.142.0.6 (nic0)	35.237.92.21	SSH ▾ ⋮

`bash run.sh software-capstone testing 35.237.139.3 35.237.233.74`

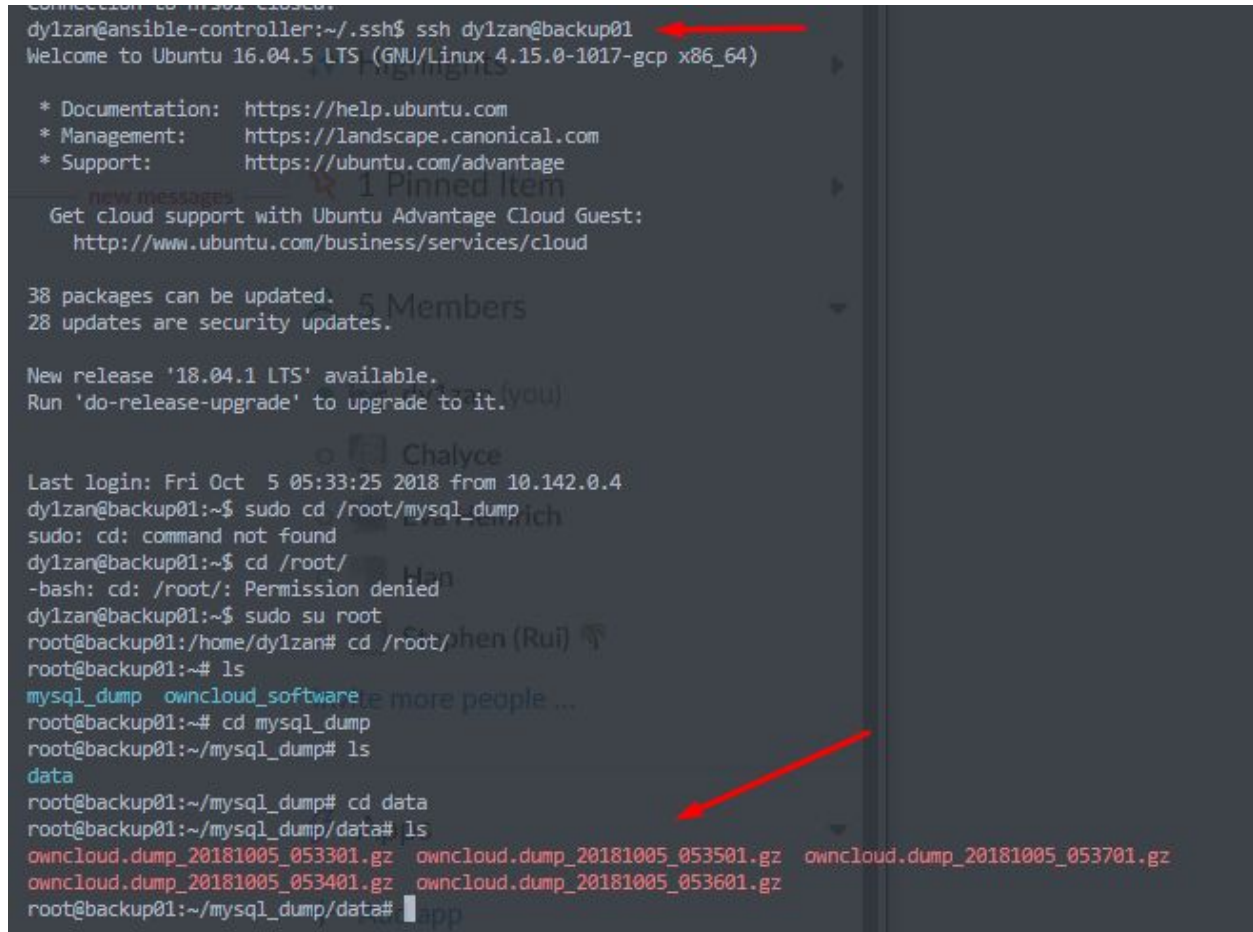
```
dylzan@ansible-controller:~/task1$ cd ..
dylzan@ansible-controller:~$ ls
task1 task3
dylzan@ansible-controller:~$ cd task3
dylzan@ansible-controller:~/task3$ ls
backup.yml main.yml restore.sh roles run.sh tasks templates vars
dylzan@ansible-controller:~/task3$ bash run.sh software-capstone testing 35.237.139.3 35.237.233.74
Executing: /tmp/apt-key-gpghome.jXNub8gDUZ/gpg.1.sh --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367
gpg: key 93C4A3FD7BB9C367: public key "Launchpad PPA for Ansible, Inc." imported
gpg: Total number processed: 1
gpg:      imported: 1
Hit:1 http://security.debian.org stretch/updates InRelease
Ign:2 http://deb.debian.org/debian stretch InRelease

PLAY RECAP *****
35.231.198.1  : ok=10  changed=10  unreachable=0  failed=0
localhost    : ok=9    changed=5   unreachable=0  failed=0
```

Success!



Now, testing if it worked...



```
connection to 158.358.158.158:
dylzan@ansible-controller: ~/.ssh$ ssh dylzan@backup01
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.15.0-1017-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

38 packages can be updated.
28 updates are security updates.

New release '18.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Fri Oct  5 05:33:25 2018 from 10.142.0.4
dylzan@backup01:~$ sudo cd /root/mysql_dump
sudo: cd: command not found
dylzan@backup01:~$ cd /root/
-bash: cd: /root/: Permission denied
dylzan@backup01:~$ sudo su root
root@backup01:/home/dylzan# cd /root/
root@backup01:~# ls
mysql_dump  owncloud_software
root@backup01:~# cd mysql_dump
root@backup01:~/mysql_dump# ls
data
root@backup01:~/mysql_dump# cd data
root@backup01:~/mysql_dump/data# ls
owncloud.dump_20181005_053301.gz  owncloud.dump_20181005_053501.gz  owncloud.dump_20181005_053701.gz
owncloud.dump_20181005_053401.gz  owncloud.dump_20181005_053601.gz
```

The screenshot shows a terminal session where a user named dylzan connects via SSH to an instance named backup01. The user then attempts to navigate to /root/mysql\_dump using 'sudo cd', which fails. After using 'sudo su' to become root, the user successfully navigates to /root/mysql\_dump and then to /root/mysql\_dump/data, where several compressed MySQL dump files are listed. Two red arrows are present: one pointing to the SSH command and another pointing to the 'data' directory listing.

I SSH into the backup01 instance and go to /root/mysql\_dump/data to see the compressed backups for MySQL.