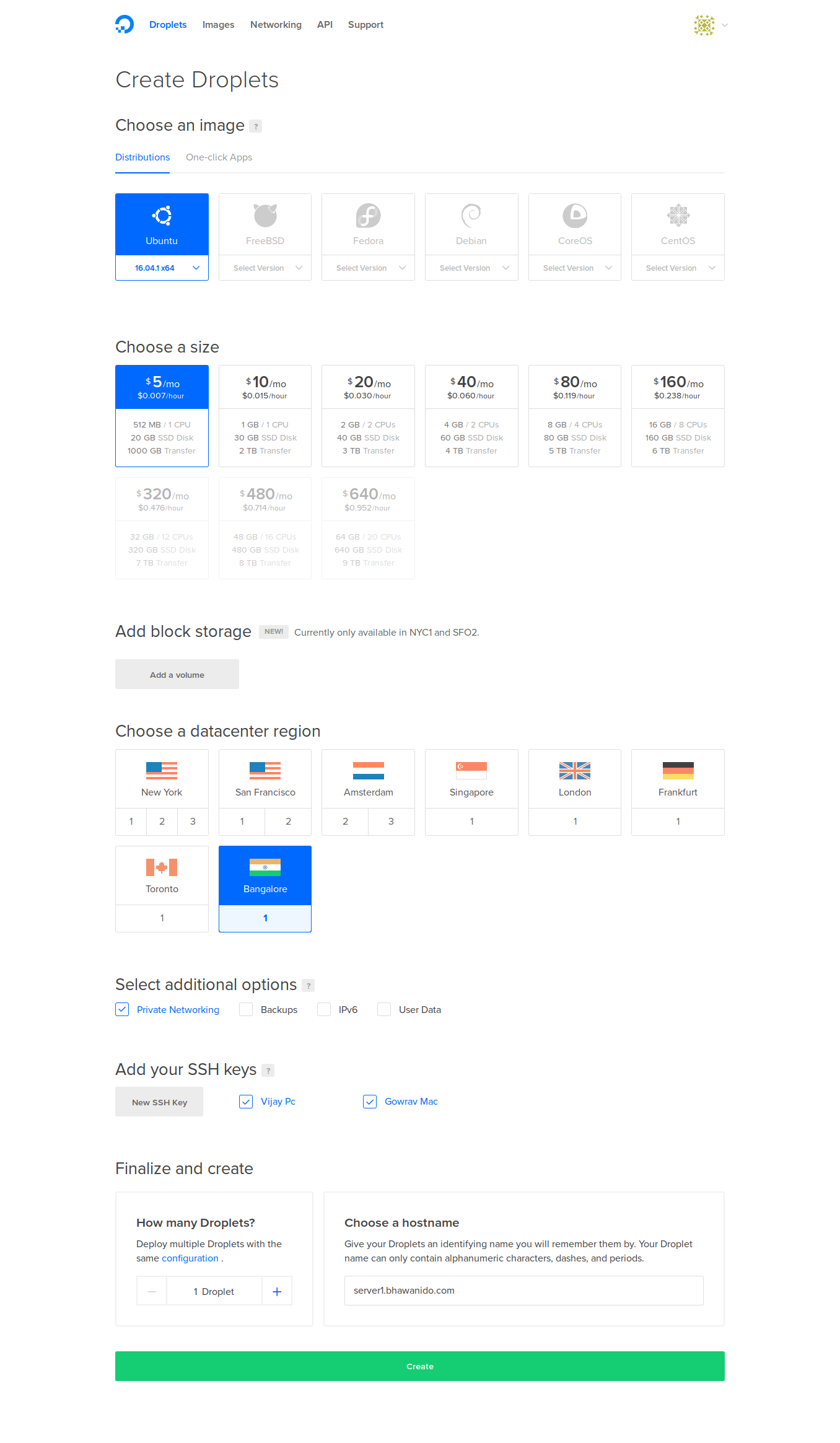
# MariaDB Galera Cluster 10.1 Installation on DigitalOcean Ubuntu 16.04

# Preparing the Server

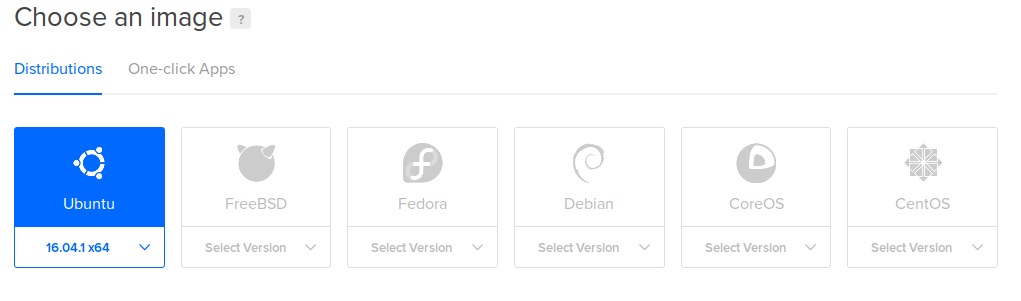
**follow the following Steps**

1. Login to digitalocean.com & Create a Droplets

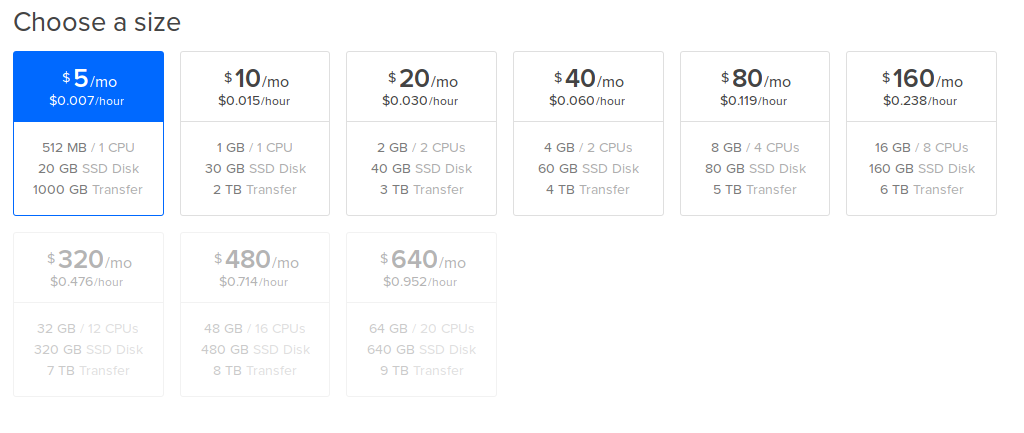


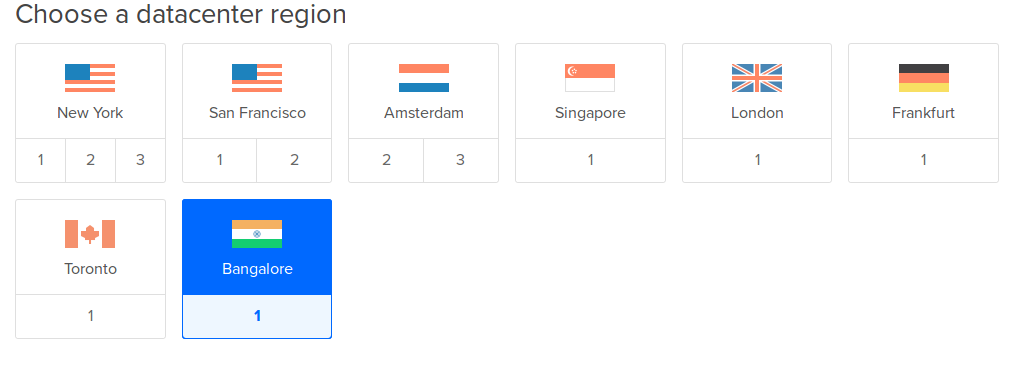
2. Create a Droplets to follow the instructions

A**. Choose a droplets Image**

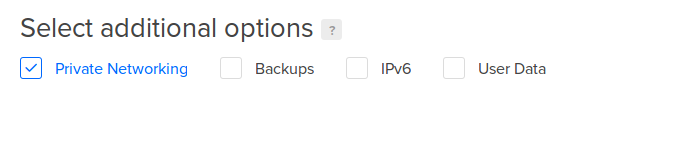


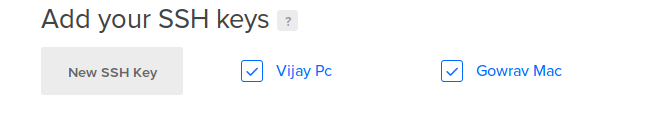
B**. Choose a size package**

 **C.** Choose a Data Center Region



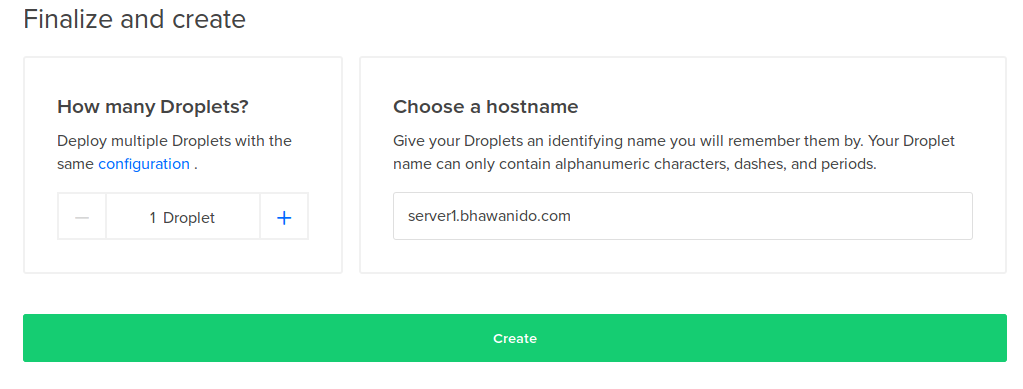
D. Select Additional Options

 E. Add Your Ssh Key(optional)



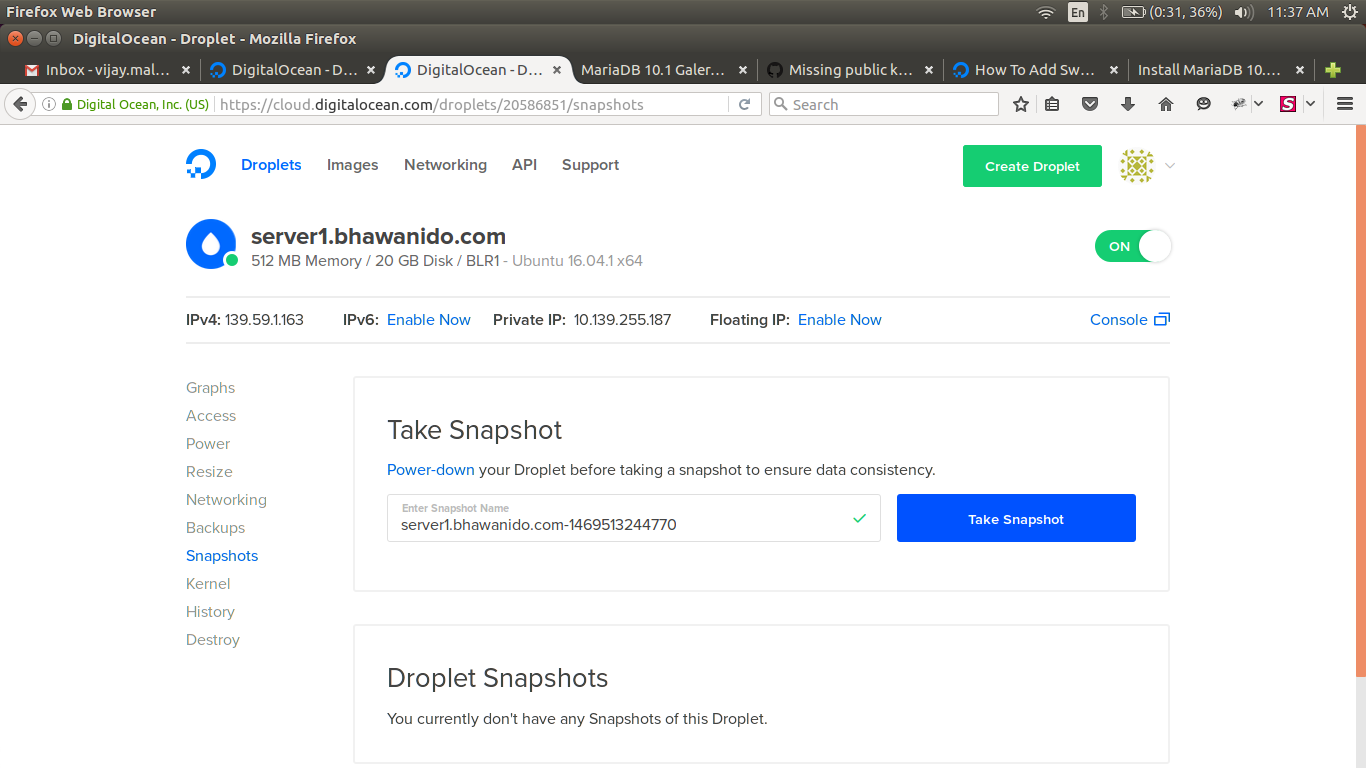
F. Finalize And Create Droplets

**type your droplets name**

 G. Take a Snap Shot after ceate your Droplets

Click on droplets to open your droplets panel

click to snapshot & take a snap shot



**Add a Swap Space to Increase Your Droplets Capacities**

Follow the command to add swap partition

1. Check the System for Swap Information

$ **sudo swapon --show**

If you don't get back any output, this means your system does not have swap space available currently.

You can verify that there is no active swap using the free utility:

**$ free -h**

**Output**

total used free shared buff/cache available

Mem: 488M 36M 104M 652K 348M 426M

Swap: 0B 0B 0B

As you can see in the "Swap" row of the output, no swap is active on the system.

## Check Available Space on the Hard Drive Partition

$ **df -h**

**Output**

Filesystem Size Used Avail Use% Mounted on

udev 238M 0 238M 0% /dev

tmpfs 49M 624K 49M 2% /run

/dev/vda1 20G 1.1G 18G 6% /

tmpfs 245M 0 245M 0% /dev/shm

tmpfs 5.0M 0 5.0M 0% /run/lock

tmpfs 245M 0 245M 0% /sys/fs/cgroup

tmpfs 49M 0 49M 0% /run/user/1001

The device under /dev is our disk in this case. We have plenty of space available in this example (only 4.1G used). Your usage will probably be different

## **Create a Swap File**

Now that we know our available hard drive space, we can go about creating a swap file within our filesystem. We will create a file of the swap size that we want called swapfile in our root (/) directory.

The best way of creating a swap file is with the fallocate program. This command creates a file of a preallocated size instantly.

Since the server in our example has 512MB of RAM, we will create a 4 Gigabyte file in this guide. Adjust this to meet the needs of your own server:

**$ sudo fallocate -l 4G /swapfile**

**We can verify that the correct amount of space was reserved by typing:**

**$ ls -lh /swapfile**

**Output:** -rw-r--r-- 1 root root 4.0G Apr 25 11:14 /swapfile

Our file has been created with the correct amount of space set aside.

## Enabling the Swap File

**$ sudo chmod 600 /swapfile**

Verify the permissions change by typing:

**$ ls -lh /swapfile**

**Output**

-rw------- 1 root root 4.0G Apr 25 11:14 /swapfile

As you can see, only the root user has the read and write flags enabled.

We can now mark the file as swap space by typing:

**$ sudo mkswap /swapfile**

**Output**

Setting up swapspace version 1, size = 4 Gib (1073737728 bytes)

no label, UUID=6e965805-2ab9-450f-aed6-577e74089dbf

After marking the file, we can enable the swap file, allowing our system to start utilizing it:

**$ sudo swapon /swapfile**

We can verify that the swap is available by typing:

**$ sudo swapon --show**

**Output**

NAME TYPE SIZE USED PRIO

/swapfile file 4G 0B -1

We can check the output of the free utility again to corroborate our findings:

**$ free -h**

**Output**

total used free shared buff/cache available

Mem: 488M 37M 96M 652K 354M 425M

Swap: 4.0G 0B 4.0G

Our swap has been set up successfully and our operating system will begin to use it as necessary.

More Details goto the link

<https://www.digitalocean.com/community/tutorials/how-to-add-swap-space-on-ubuntu-16-04>

**server Setup**

Install Apache with php and mysql

# Install MariaDB 10.1 on Ubuntu14.04/15.10/16.04

I always like to install software packages from official repository (if there’s one) rather than from my Linux distribution repository. For one thing, I can install the latest stable version. For another, I don’t have to worry about distribution specific modifications. In other words, what I got is a bog-standard package no matter what Linux distribution I use.

This tutorial will guide you through the process of installing the latest stable version of MariaDB and that’s MariaDB 10.1 on Ubuntu 14.04 and 15.10, 16.04. [Ubuntu](https://www.linuxbabe.com/tag/ubuntu) repository has MariaDB 10.0, but no MariaDB 10.1.

## Step1: Install software-properties-common

**$ sudo apt-get install software-properties-common**

## Step2: Fetch MariaDB signing key from Ubuntu’s key server.

**$ apt-key adv --keyserver ha.pool.sks-keyservers.net --recv-keys F1656F24C74CD1D8**

## Step3: add MariaDB repository to your system.

### **ubuntu 14.04**

**sudo add-apt-repository 'deb [arch=amd64,i386] http://sgp1.mirrors.digitalocean.com/mariadb/repo/10.1/ubuntu trusty main'**

### **ubuntu 15.10**

**sudo add-apt-repository 'deb [arch=amd64,i386] http://sgp1.mirrors.digitalocean.com/mariadb/repo/10.1/ubuntu wily main'**

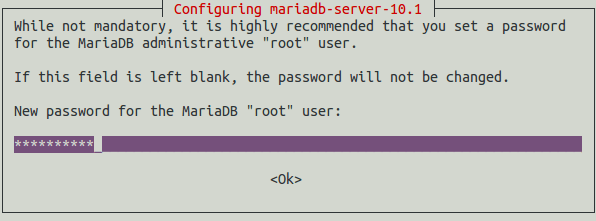
### Ubuntu 16.04

**$ sudo add-apt-repository 'deb [arch=amd64,i386] http://sgp1.mirrors.digitalocean.com/mariadb/repo/10.1/ubuntu xenial main'**

**wi$ sudo apt-get update**

**$ sudo apt-get install mariadb-server**

You will be asked to set a password for the MariaDB root user.



After it’s installed, mysqld process will be automatically started.

Check version

**$ mysql --version**

root@server1:~# mysql --version

**Output:**

mysql Ver 15.1 Distrib 10.1.16-MariaDB, for debian-linux-gnu (x86\_64) using readline 5.2

root@server1:~#

Finally we should execute the secure installation script to remove anonymous user, disable remote root login and remove test database for security reasons.

**$ mysql\_secure\_installation**

You will be asked these questions:

Enter current password for root (enter for none): <-- press enter  
Set root password? [Y/n] <-- y  
New password: <-- Enter the new MariaDB root password here  
Re-enter new password: <-- Repeat the password  
Remove anonymous users? [Y/n] <-- y  
Disallow root login remotely? [Y/n] <-- y  
Reload privilege tables now? [Y/n] <-- y

Test the login to MariaDB with the "mysql command"

**$ mysql -u root -p**

**Output:**

**Welcome to the MariaDB monitor. Commands end with ; or \g.**

**Your MariaDB connection id is 9**

**Server version: 10.1.16-MariaDB-1~xenial mariadb.org binary distribution**

**Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.**

**Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.**

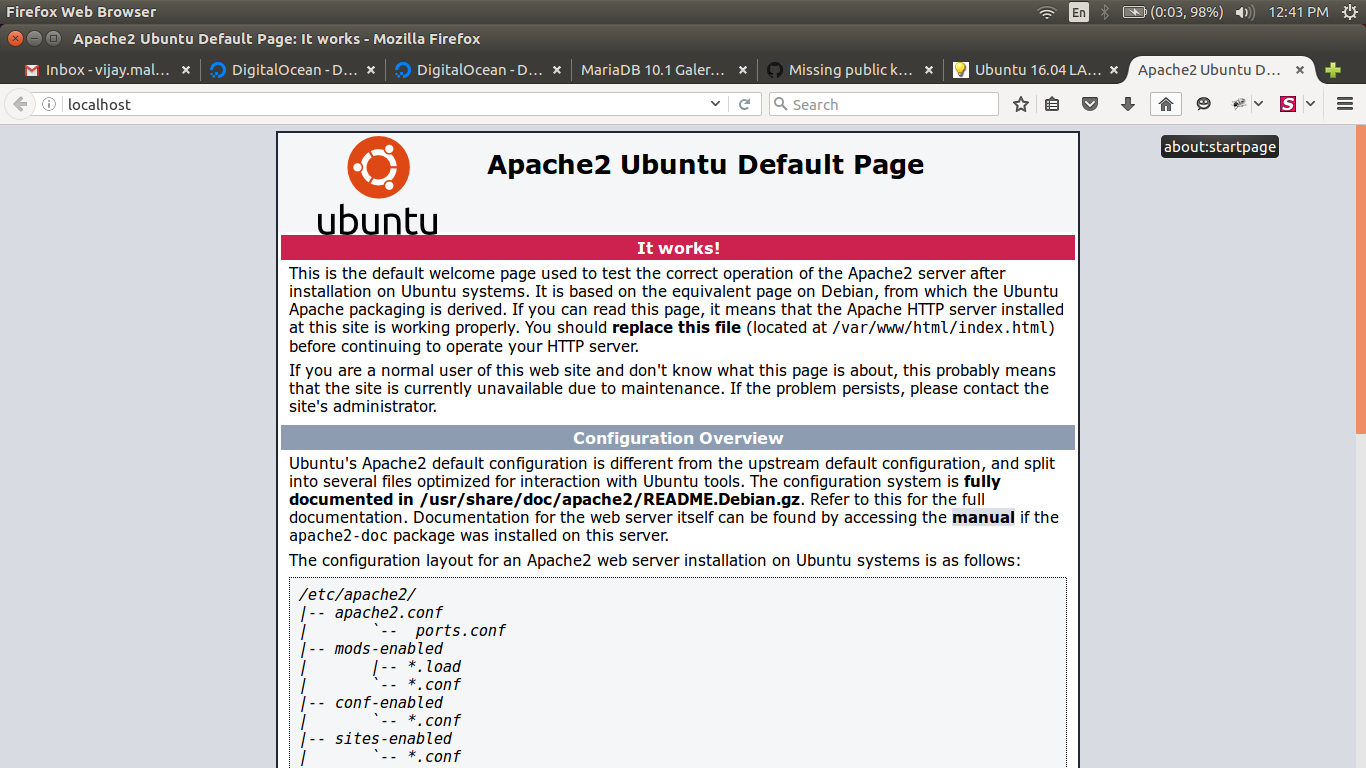
**To leave the MariaDB shell, enter the command "quit" and press enter.**

## Install Apache 2.4

Apache 2 is available as an Ubuntu package, therefore we can install it like this:

**$ sudo apt-get -y install apache2**

Now direct your browser to http://your\_ip\_address, and you should see the Apache2 default page (It works!):



The document root of the apache default vhost is /var/www/html on Ubuntu and the main configuration file is /etc/apache2/apache2.conf. The configuration system is fully documented in /usr/share/doc/apache2/README.Debian.gz.

## Install PHP 7

We can install PHP 7 and the Apache PHP module as follows:

**$ sudo apt-get -y install php7.0 libapache2-mod-php7.0**

Then restart Apache:

**$ sudo systemctl restart apache2** or **sudo service apache2 restart**

## Test PHP and get details about your PHP installation

The document root of the default web site is /var/www/html. We will now create a small PHP file (info.php) in that directory and call it in a browser. The file will display lots of useful details about our PHP installation, such as the installed PHP version.

**$ php**

<?php

phpinfo();

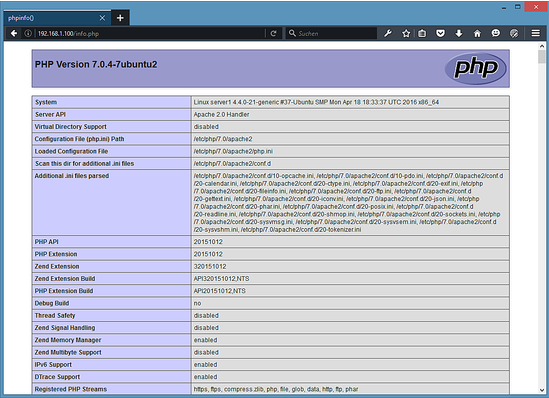
?>

**Clt+x, y ,enter**

Then change the owner of the info.php file to the www-data user and group.

**$ chown www-data:www-data /var/www/html/info.php**

Now we call that file in a browser (e.g. [http://ip\_address/info.php](http://192.168.1.100/info.php)):



As you see, PHP 7.0 is working, and it's working through the Apache 2.0 Handler, as shown in the Server API line. If you scroll further down, you will see all modules that are already enabled in PHP5. MySQL is not listed there which means we don't have MySQL / MariaDB support in PHP yet.

## Get MySQL / MariaDB support in PHP

To get MySQL support in PHP, we can install the php7.0-mysql package. It's a good idea to install some other PHP modules as well as you might need them for your applications. You can search for available PHP modules like this:

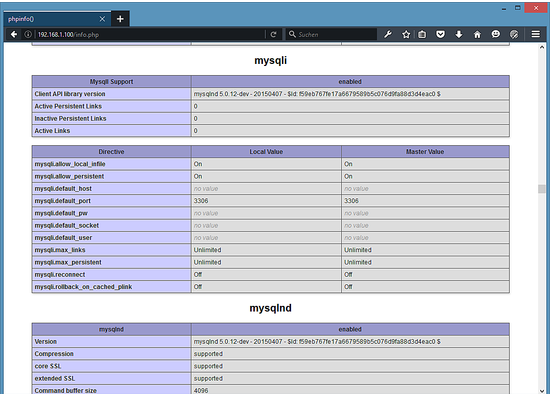
**$ apt-cache search php7.0**

Pick the ones you need and install them like this:

**$ sudo apt-get -y install php7.0-mysql php7.0-curl php7.0-gd php7.0-intl php-pear php-imagick php7.0-imap php7.0-mcrypt php-memcache php7.0-pspell php7.0-recode php7.0-sqlite3 php7.0-tidy php7.0-xmlrpc php7.0-xsl php7.0-mbstring php-gettext**

Now restart Apache2:

**$ systemctl restart apache2**



PHP 7 has now MySQL / MariaDB support as shown in phpinfo() above.

## Install the APCu PHP cache to speed up PHP

APCu is a free PHP opcode cacher for caching and optimizing PHP intermediate code. It is strongly recommended to have an Opcache installed to speed up your PHP page.

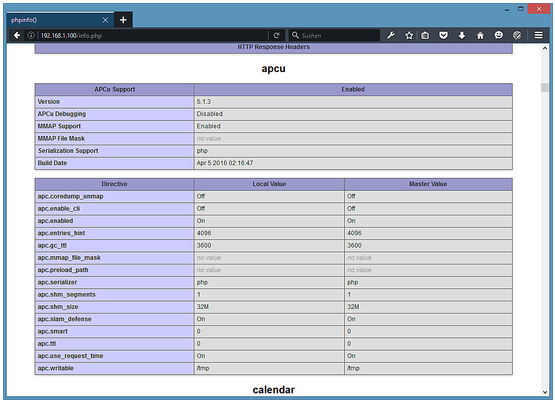
APCu can be installed as follows:

**$ sudo apt-get -y install php-apcu**

Now restart Apache:

**$ systemctl restart apache2**

Now reload http://your\_ip\_address/info.php in your browser and scroll down to the modules section again. You should now find lots of new modules there:



Please don't forget to delete the info.php file when you don't need it anymore as it provides sensitive details of your server. Run the following command to delete the file.

**$ rm -f /var/www/html/info.php**

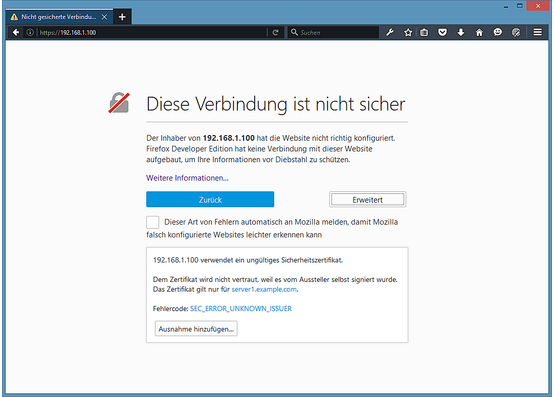
## **Enable the SSL website in apache**

SSL/ TLS is a security layer to encrypt the connection between the web browser and your server. Execute the following commands on your server to enable https:// support. Run:

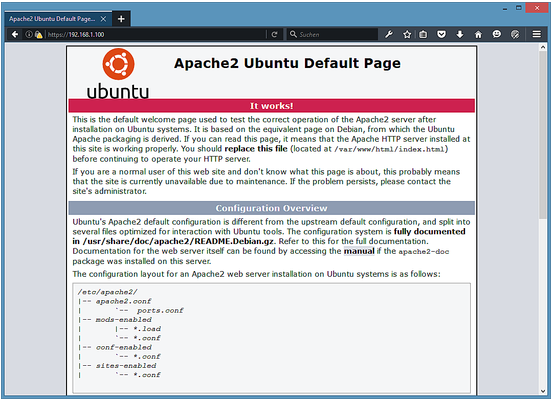
**$ a2enmod ssl  
$ a2ensite default-ssl**

which enables the ssl module and adds a symlink in the /etc/apache2/sites-enabled folder to the file /etc/apache2/sites-available/default-ssl.conf to include it into the active apache configuration. Then restart apache to enable the new configuration:

**$ systemctl restart apache2**

Now test the SSL connection by opening [https://your\_i](https://192.168.1.100/)p\_address in a web browser.

You will receive an SSL warning as the SSL certificate of the server is a "self-signed" SSL certificate, this means that the browser does not trust this certificate by default and you have to accept the security warning first. After accepting the warning, you will see the apache default page.



The closed "Green Lock" in front of the URL in the browser shows that the connection is encrypted. To get rid of the SSL warning, replace the self-signed SSL certificate /etc/ssl/certs/ssl-cert-snakeoil.pem with an officially signed SSL certificate from an SSL Authority.

## **Install phpMyAdmin**

[phpMyAdmin](http://www.phpmyadmin.net/) is a web interface through which you can manage your MySQL databases. It's a good idea to install it:

**$ sudo apt-get -y install phpmyadmin**

You will see the following questions:

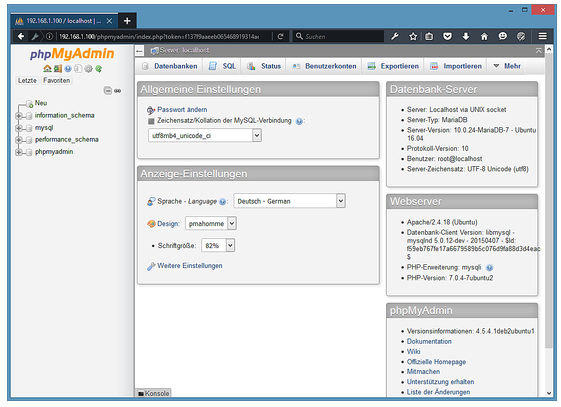
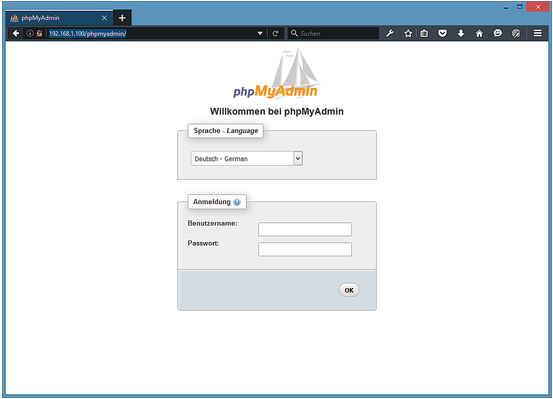
Web server to configure automatically: <-- **Select the option**: **apache2**  
Configure database for phpmyadmin with dbconfig-common? <-- **Yes**  
MySQL application password for phpmyadmin: <-- **Press enter, apt will create a random password automatically**.

MariaDB enables a plugin called "unix\_socket" for the root user by default, this plugin prevents that the root user can log in to PHPMyAdmin and that TCP connections to MySQL are working for the root user. Therefore, I'll deactivate that plugin with the following command:

**$ echo "update user set plugin='' where User='root'; flush privileges;" | mysql -u root -p mysql**

Enter the MariaDB root password, when requested by the mysql command.

Afterward, you can access phpMyAdmin under http://your\_ip\_address/phpmyadmin/:

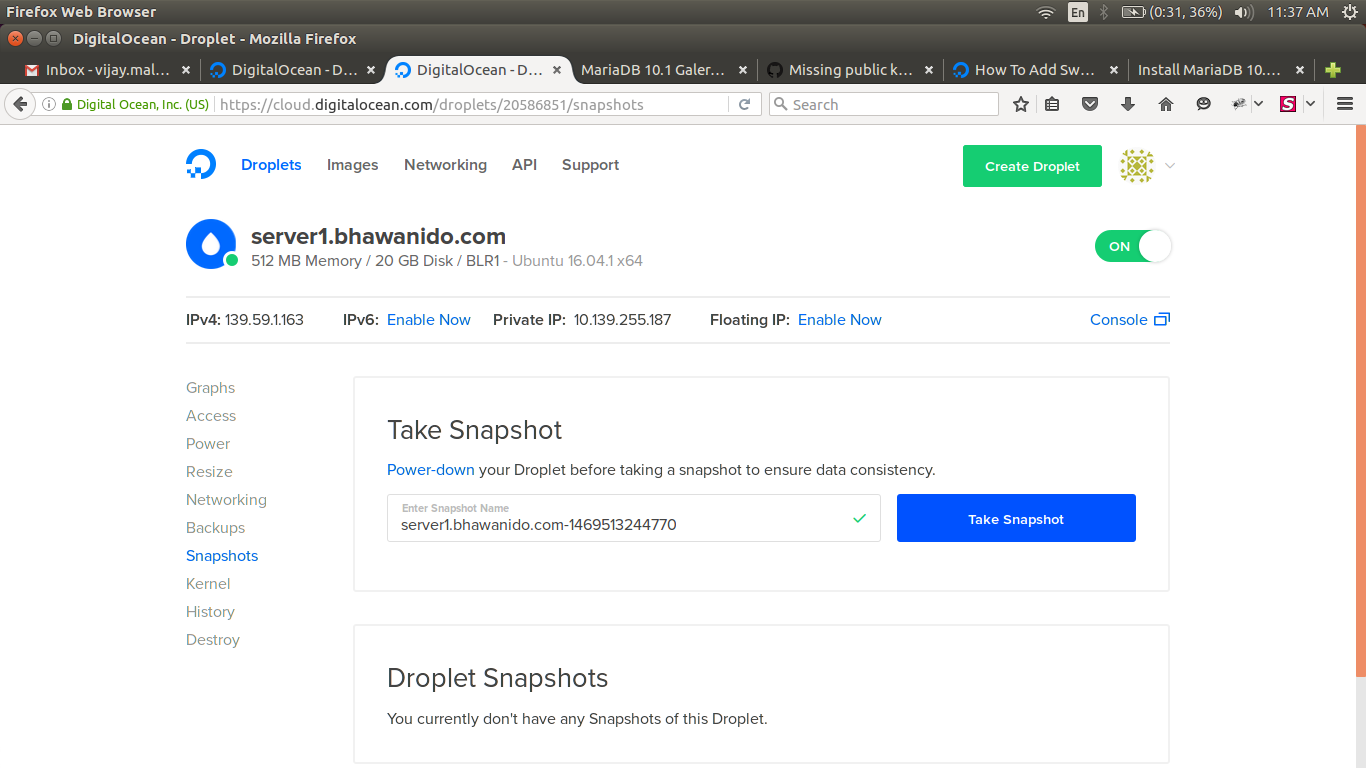


**Take a Snap Shot**

**Configuration server Setup**

Click on droplets to open your droplets panel

click to snapshot & take a snap shot



This is a Howto about installing MariaDB Galera Cluster

## What we need

In our setup we assume 3 nodes (server1,server2,server3) with one interface each. We assume following IP addresses: 139.59.1.163, 139.59.16.234, and 139.59.4.203.

## Configuring Galera

So we have to do some configuration next. There is a MariaDB configuration part and one part to configure Galera (starting with wsrep\_). As we do the most basic and simple installation in this Howto, it is sufficient you just change the IP’s (Remember: 139.59.1.163, 139.59.16.234,139.59.4.203) with your IP’s. In our example, we have set hostnames on each node (server1,server2,server3) so we do not need the IP addresses of the hosts.  
This will be needed to define the wsrep\_cluster\_address Variable (the list of nodes a starting mysqld contacts to join the cluster).

The following configuration file has to be distributed on all nodes. We use a separate configuration file (create a new file) /etc/mysql/conf.d/galera.cnf with the following settings:

**$ sudo /etc/mysql/conf.d**

**$ nano galera.cnf**

[mysqld]

#mysql settings

binlog\_format=ROW

default-storage-engine=innodb

innodb\_autoinc\_lock\_mode=2

innodb\_doublewrite=1

query\_cache\_size=0

query\_cache\_type=0

bind-address=0.0.0.0

#galera settings

wsrep\_on=ON

wsrep\_provider=/usr/lib/galera/libgalera\_smm.so

wsrep\_cluster\_name="test\_cluster"

wsrep\_cluster\_address=gcomm://server1,server2,server3

wsrep\_sst\_method=rsync

FYI: The shared library for wsrep\_provider is provided by the installed galera package.

We could also change the cluster name by changing the value of wserp\_cluster\_name to fit our style. This setting also works as a shared secret to control the access to the cluster.

With wsrep\_cluster\_address you see the hostnames of our setup. wsrep\_cluster\_address could also be gcomm://139.59.1.163, 139.59.16.234,139.59.4.203. Multiple IP’s or hostnames have to be comma seperated.

The wsrep\_sst\_method tells what method to use to synchronise the nodes. While there are also mysqldump and xtrabackup available, I prefer rsync because it is easy to configure (i.e. it does not need any credentials set on the nodes). If you are considering using the xtrabackup method, don’t forget to install xtrabackup.

Now stop mysqld on all nodes:

server1**# systemctl stop mysql**

server2**# systemctl stop mysql**

server3**# systemctl stop mysql**

Debian/Ubuntu uses a special user (‚debian-sys-maint’@’localhost‘) in their init script and the credentials for that user are stored in /etc/mysql/debian.cnf. This user is used to make some checks starting MySQL. Checks I don’t think belong into a service script anyway. Because of the unique password for the user on each system, mysqld will throw some errors while starting and stopping.  
We could simply ignore it, but the user is also used to shutdown mysqld. This is also not required, as a SIGTERM is sufficient to shutdown the mysqld :/  
So we’ve got to fix it, by copying /etc/mysql/debian.cnf from the first node (server1) to all other nodes. So the data and configuration files have the same data.

## Starting the Galera Cluster

The configuration file (galera.cnf) is already distributed to all nodes, so we next start the first mysqld on cluster01. This node initializes/starts the cluster (creates a GTID).

server1**# galera\_new\_cluster**

**cluster01# mysql -u root -p -e 'SELECT VARIABLE\_VALUE as "cluster size" FROM INFORMATION\_SCHEMA.GLOBAL\_STATUS WHERE VARIABLE\_NAME="wsrep\_cluster\_size"'**

**+--------------+**

| cluster size |

+--------------+

| 1 |

+--------------+

**If you see the above, great! That’s what we would expect. Now that the Cluster already exists, we let the next nodes just start and join the cluster.**

server2**# systemctl start mysql**

**Let’s pause here and do a quick check. As we are running a cluster it is not important if we execute the following on** server1 **or** server2**.**

**$ mysql -u root -p -e 'SELECT VARIABLE\_VALUE as "cluster size" FROM INFORMATION\_SCHEMA.GLOBAL\_STATUS WHERE VARIABLE\_NAME="wsrep\_cluster\_size"'**

**+--------------+**

| cluster size |

+--------------+

| 2 |

+--------------+

If you see the above, very nice! Now let’s start the third node:

server3**# systemctl start mysql**

**$ mysql -u root -p -e 'SELECT VARIABLE\_VALUE as "cluster size" FROM INFORMATION\_SCHEMA.GLOBAL\_STATUS WHERE VARIABLE\_NAME="wsrep\_cluster\_size"'**

**+--------------+**

| cluster size |

+--------------+

| 3 |

+--------------+

Ok we are finished. We have a running MariaDB Galera Cluster \o/

## Restarting the whole Galera Cluster

If the whole galera cluster has to be restarted (due to poweroutage for example) we have to complete the following steps by hand:

1. Identify the node with the most advanced node state ID.
2. Start the most advanced node as the first node of the cluster.
3. Start the rest of the node as usual.

#### Identify the node with the most advanced node state ID.

**$ cat /var/lib/mysql/grastate.dat**

The node with the highest seqno is your new first host.

#### Start the most advanced node as the first node of the cluster.

On the new first host, the one with the highes squence number, bootstrap the galera cluster again:

**$ galera\_new\_cluster**

#### Start the rest of the node as usual.

On each other node start mysqld as usual:

$ **systemctl start mysql**