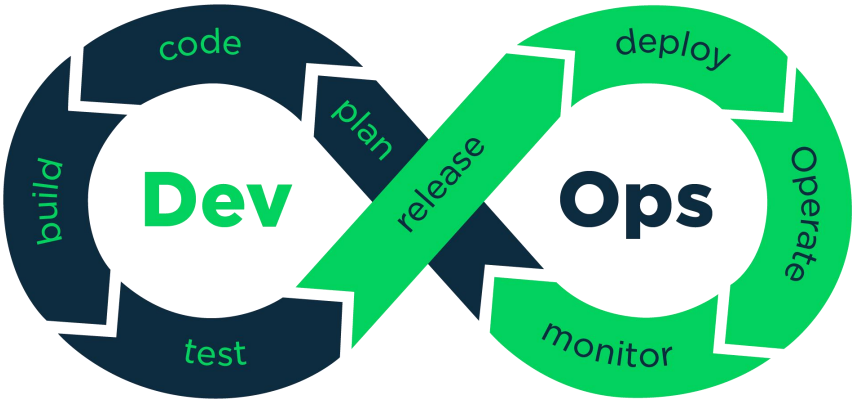
DevOps: A beginners guide



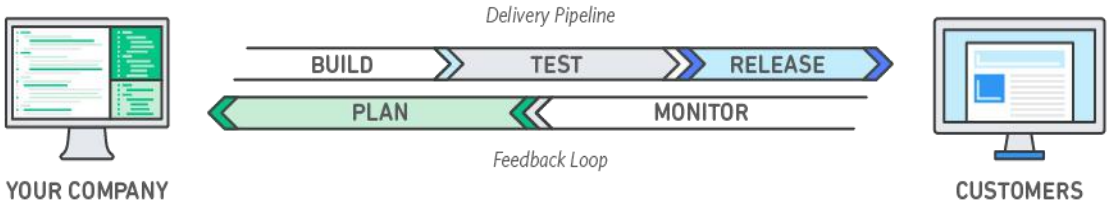
By: Rohit Sah

CONTENTS:

DevOps with AWS

First of all what is DevOps

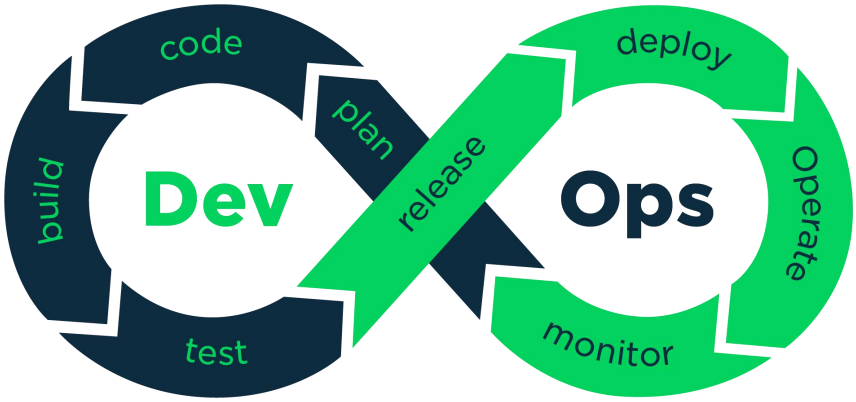
DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



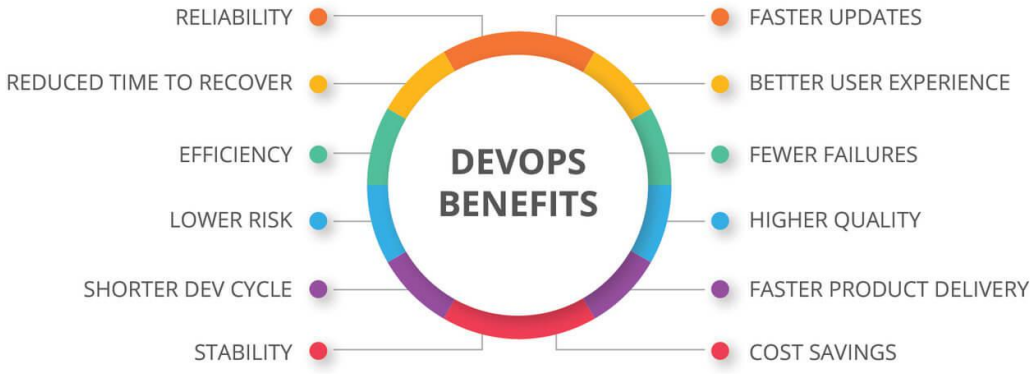
Under a DevOps model, development and operations teams are no longer "siloed." Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team's velocity.



Benefits of DevOps



Now we have seen what is DevOps and its benefits now the question arises why DevOps

Software and the Internet have transformed the world and its industries, from shopping to entertainment to banking. Software no longer merely supports a business; rather it becomes an integral component of every part of a business. Companies interact with their customers through software delivered as online services or applications and on all sorts of devices. They also use software to increase operational efficiencies by transforming every part of the value chain, such as logistics, communications, and operations. In a similar way that physical goods companies transformed how they design, build, and deliver products using industrial automation throughout the 20th century, companies in today's world must transform how they build and deliver software.

To adapt DevOps there are following practices that can be made

- Continuous Integration
- Continuous Delivery
- Microservices
- Infrastructure as Code
- Monitoring and Logging
- Communication and Collaboration

Continuous Integration:

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery:

Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Microservices:

The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

Infrastructure as Code:

Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud's API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

Monitoring and Logging:

Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.

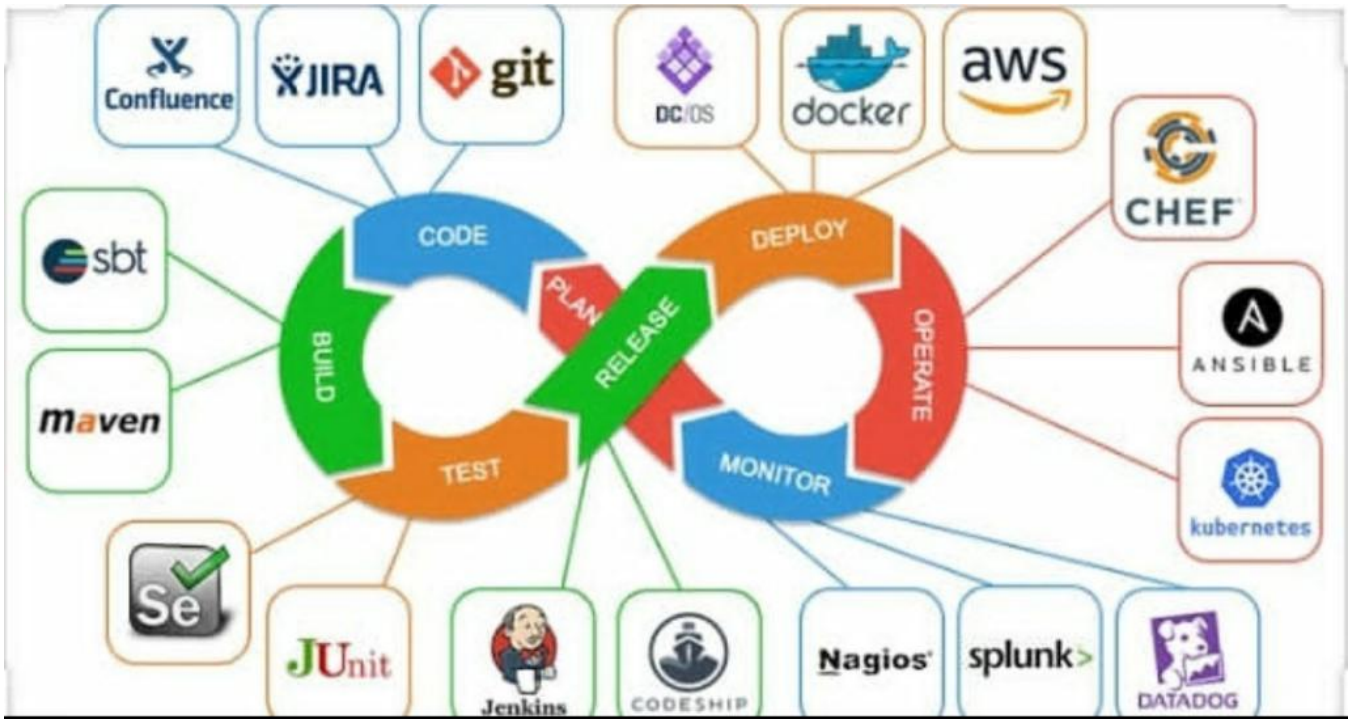
Communication and Collaboration:

Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

DevOps Tools:

The DevOps model relies on effective tooling to help teams rapidly and reliably deploy and innovate for their customers. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps.

Some of the widely used tools for each section are:



LINUX 

The Pathway to DevOps goes through Linux as Linux offers the DevOps team the flexibility and scalability needed to create a dynamic development process. You can set it up any way that suits your needs. Rather than letting the operating system dictate how you work, you can configure it to work for you.

Why Linux

> As per the latest report from Top 500, Linux now runs on all of the fastest 500 supercomputers in the world. The previous number was 498 as remaining two supercomputers ran Unix.
>
> Top500 is an independent project that was launched in 1993 to benchmark supercomputers. It publishes the details about the top 500 fastest supercomputers known to them, twice a year. You can go the website and filter out the list based on various criteria such as country, OS type, vendors etc.
>
> Looking deeper, Linux's importance to the Web is even more extreme. By W3Cook's analysis of Alexa's data, 96.3 percent of the top 1 million web servers are running Linux. The remainder is split between Windows, 1.9 percent, and FreeBSD, 1.8 percent.

No, I didn't use a misleading blog title. Smartphones powered by Linux are in fact dominating the smartphone market. A few of you may be scratching your heads at this point (stop that, you'll go bald) while others are filled with that *Sound of Music – "The Hills are Alive!"* kind of Linux pride! Read on and I'll provide some pudding, filled with proof that ~~81%~~ 86% of all Smartphones are powered by Linux.

Working with the Shell

Command and Arguments:

```
rohit_ubuntu@camouflage:~$ echo "hey there"
hey there
```

```
rohit_ubuntu@camouflage:~$ uptime
 22:33:14 up 33 min,  0 users,  load average: 0.02, 0.04, 0.00
```

```
command <options> <arguments>
echo = command
option = -n
Hello = argument
```

Command Types:

Internal or Built-in Commands
echo, cd, pwd, set e.t.c

```
[~]$ type echo
echo is a shell built-in
[~]$
```

External Commands
mv, date, uptime, cp, uptime e.t.c

```
[~]$ type mv
mv is hashed (/bin/mv)
[~]$
```

pwd:

```
rohit_ubuntu@camouflage:~$ pwd
/home/rohit_ubuntu
```

cd:

```
rohit_ubuntu@camouflage:~$ cd test
rohit_ubuntu@camouflage:~/test$
```

Ls:

```
rohit_ubuntu@camouflage:~/test$ ls
test_file_1.txt  test_file_2.txt  test_file_3.txt
```

Cat:

```
rohit_ubuntu@camouflage:~/test$ cat test_file_1.txt
Hello There this is from test_file_1.txt
rohit_ubuntu@camouflage:~/test$
```

mv :

```
rohit_ubuntu@camouflage:~$ mv /home/rohit_ubuntu/test/test_file_1.txt /home/rohit_ubuntu/test_2/
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ cd test_2
rohit_ubuntu@camouflage:~/test_2$ ls
test_file_1.txt
rohit_ubuntu@camouflage:~/test_2$
```

cp:

```
rohit_ubuntu@camouflage:~/test$ cp /home/rohit_ubuntu/test/test_file_2.txt /home/rohit_ubuntu/test_2/test_2_file_1.txt
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cd ..
rohit_ubuntu@camouflage:~$ cd test_2
rohit_ubuntu@camouflage:~/test_2$ ls
test_2_file_1.txt  test_file_1.txt
rohit_ubuntu@camouflage:~/test_2$
rohit_ubuntu@camouflage:~/test_2$ cat test_2_file_1.txt
Hello there this is being copied from test_file_2.txt
rohit_ubuntu@camouflage:~/test_2$
```

grep:

```
[~]$ grep second sample.txt
Followed by the second line.
```

```
[~]$ grep capital sample.txt
```

grep -i

```
[~]$ grep -i capital sample.txt
The fourth line has CAPITAL LETTERS
```

grep -r:

```
[~]$ grep -r "third line" /home/michael
./sample.txt:And then the third line.
```

grep -v:

```
[~]$ grep -v "printed" sample.txt
This is the first line.
Followed by the second line.
And then the third line.
The fourth line has CAPITAL LETTERS
```

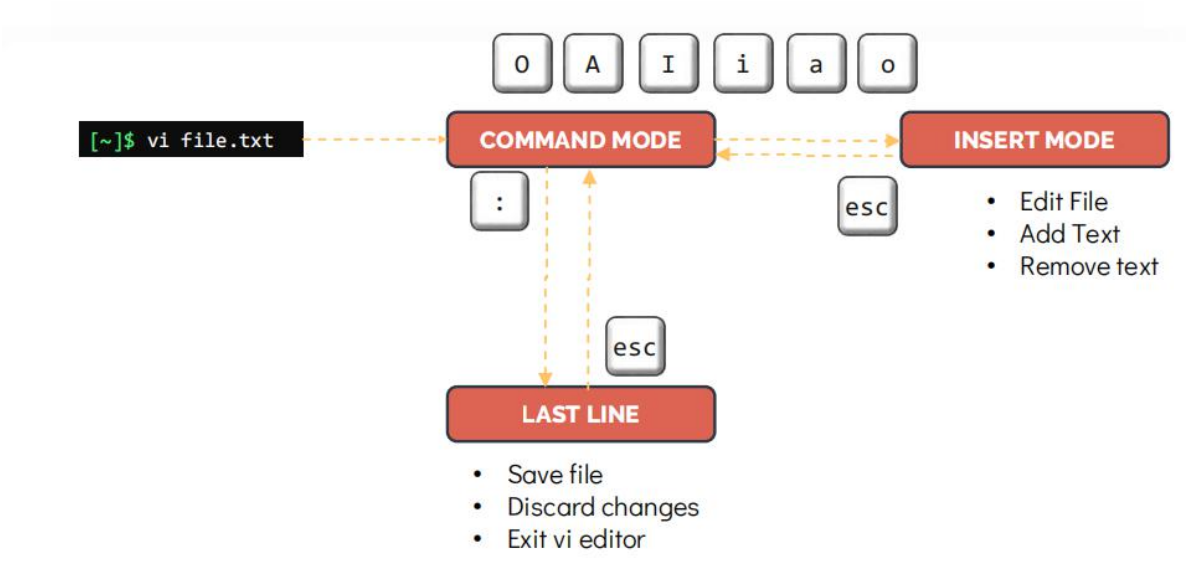
grep -A and grep -B :

```
[~]$ grep -A1 -B1 Chelsea premier-league-table.txt

2 Liverpool
3 Chelsea
4 Manchester City
```

EDITOR:

VI:

```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"sample.txt" 5L, 139C                                  1,1           All
```



Move Around

^ < v > K H J L

Copy a Line

y y

Paste

p

Delete a letter

x

```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"sample.txt" 5L, 139C                    1,1           All
```

Delete a line

d d

Delete 3 lines

d 3 d

Undo

u

Redo

r

```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"sample.txt" 5L, 139C                    1,1           All
```

Find

/line ?line

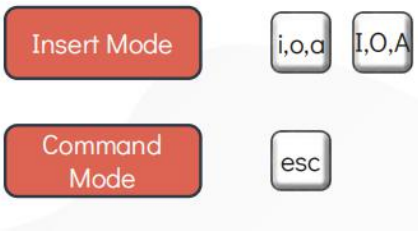
Find Next

n

Find Previous

N

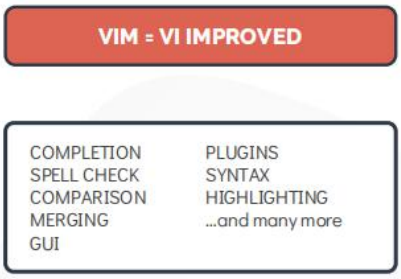
```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
/line
```




```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
~
~
~
~
~
~
```


```
This is the first line.
Followed by the second line.
Third line is very long compared to the previous two lines.
Hello there!
hello there!
I made some changes to this file.
~
~
~
~
~
~
~
~
~
~
~
~
~
```

VIM EDITOR:




```
[~]$ which vi
/usr/bin/vi

[~]$ ls -ltr /usr/bin/vi
lrwxrwxrwx 1 root root 20 Apr 10 08:31 /usr/bin/vi -> /etc/alternatives/vi

[~]$ ls -ltr /etc/alternatives/vi
lrwxrwxrwx 1 root root 18 Apr 24 02:06 /etc/alternatives/vi -> /usr/bin/vim.basic
```

```
Differences between Vim and Vi                          vi-differences

1. Simulated command                    simulated-command
2. Missing options                      missing-options
3. Limits                               limits
4. The most interesting additions       vim-additions
5. Other vim features                   other-features
6. Supported Vi features                vi-features
7. Command-line arguments               cmdline-arguments
8. POSIX compliance                     posix-compliance

===============================================================================
1. Simulated command                                    simulated-command
vi_diff.txt [Help][RO]
7,35-57         0%
This is the first line.
sample.txt                                                            1,1
Top
"vi_diff.txt" [readonly] 1370L, 57621C
```

Ifconfig:

```
[node1] (local) root@192.168.0.13 ~
$ ifconfig
docker0   Link encap:Ethernet  HWaddr 02:42:65:ED:7C:96
          inet addr:172.17.0.1  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0      Link encap:Ethernet  HWaddr 6E:AC:EF:94:68:26
          inet addr:192.168.0.13  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:1296 (1.2 KiB)  TX bytes:0 (0.0 B)

eth1      Link encap:Ethernet  HWaddr 02:42:AC:12:00:07
          inet addr:172.18.0.7  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6467 (6.3 KiB)  TX bytes:23948 (23.3 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

ping:

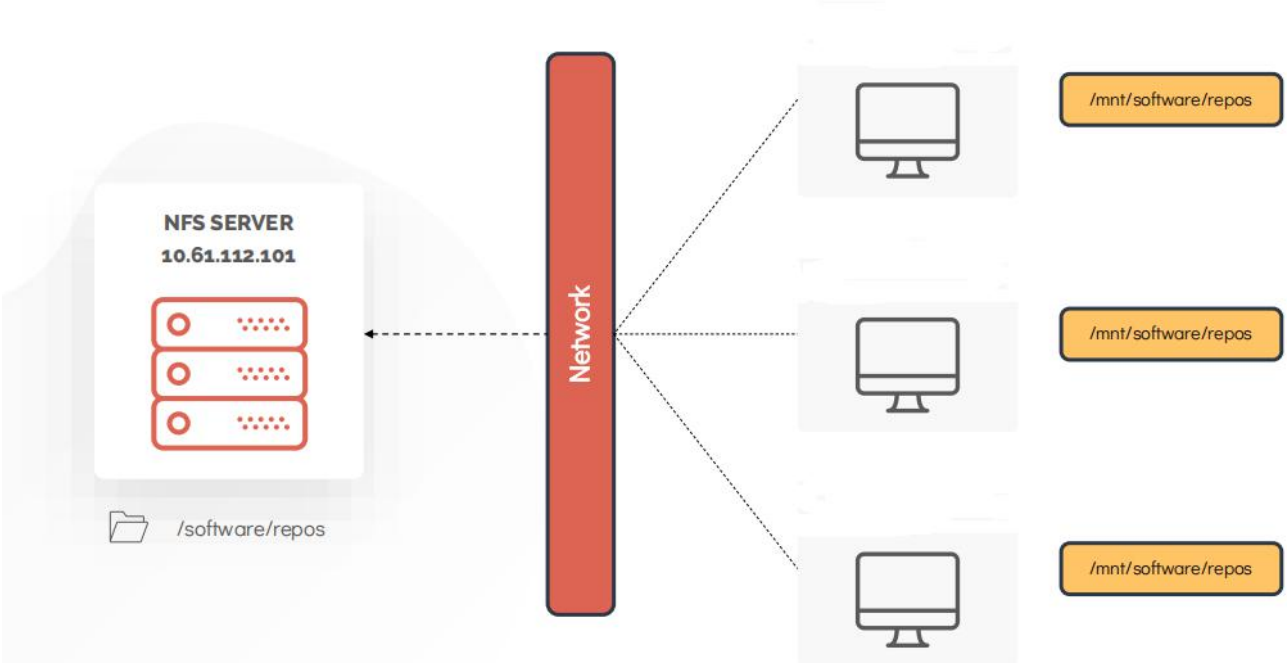
```
[node1] (local) root@192.168.0.13 ~
$ ping google.com
PING google.com (142.250.65.78): 56 data bytes
64 bytes from 142.250.65.78: seq=0 ttl=112 time=1.505 ms
64 bytes from 142.250.65.78: seq=1 ttl=112 time=1.418 ms

64 bytes from 142.250.65.78: seq=2 ttl=112 time=1.417 ms
64 bytes from 142.250.65.78: seq=3 ttl=112 time=1.613 ms
64 bytes from 142.250.65.78: seq=4 ttl=112 time=2.057 ms
```

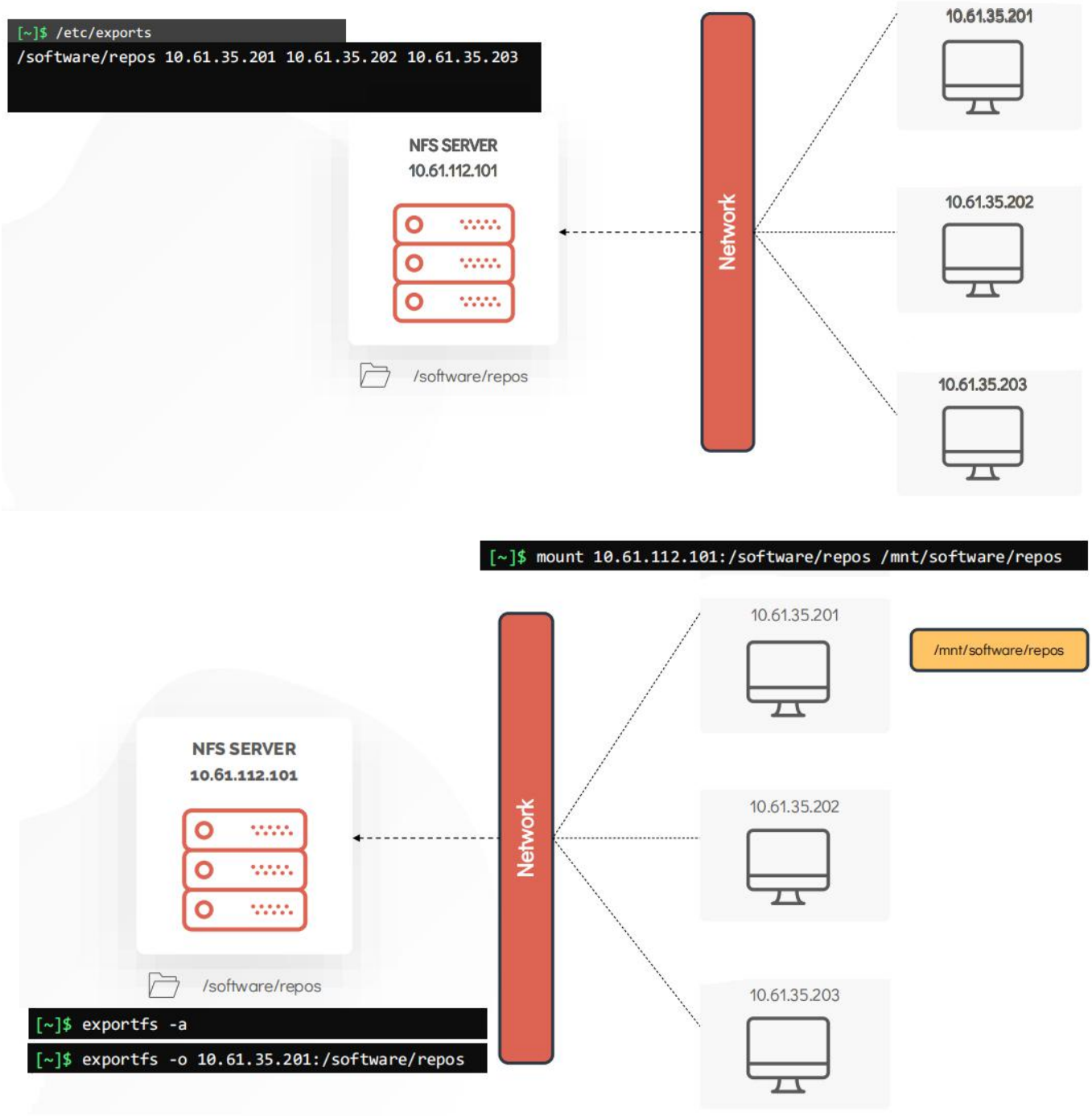
netstat:

```
rohit_ubuntu@camouflage:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  3      [ ]         STREAM     CONNECTED     17735    /run/guest-services/wsl2-expose-ports.sock
unix  3      [ ]         STREAM     CONNECTED     22612    /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     17399
unix  3      [ ]         STREAM     CONNECTED     20685
unix  3      [ ]         STREAM     CONNECTED     758      /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     18011
unix  3      [ ]         STREAM     CONNECTED     18578    /run/guest-services/memlogdq.sock
unix  3      [ ]         STREAM     CONNECTED     20798    /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     19903
unix  3      [ ]         STREAM     CONNECTED     1965
unix  3      [ ]         STREAM     CONNECTED     21861    /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     20023
unix  3      [ ]         STREAM     CONNECTED     15971
unix  3      [ ]         STREAM     CONNECTED     22611    /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     22613    /run/guest-services/docker.sock
unix  3      [ ]         STREAM     CONNECTED     20766
```

NFS(Network File system)

Network File Sharing (NFS) is a protocol that allows you to share directories and files with other Linux clients over a network. Shared directories are typically created on a file server, running the NFS server component. Users add files to them, which are then shared with other users who have access to the folder.





Advantages of a distributed file system:

1. Allows easy sharing of data among clients.
2. Provides centralized administration.
3. Provides security, i.e. one must only secure the servers to secure data

Shell Scripting:

A shell script is a computer program designed to be run by the Unix shell, a command-line interpreter. The various dialects of shell scripts are considered to be scripting languages. Typical operations performed by shell scripts include file manipulation, program execution, and printing text. Steve Bourne wrote the Bourne shell which appeared in the Seventh Edition Bell Labs Research version of Unix. Many other shells have been written; this particular tutorial concentrates on the Bourne and the Bourne Again shells. Other shells include the Korn Shell (ksh), the C Shell (csh), and variations such as tcsh.

Why Shell Scripts?

- Automate Daily Backups
- Automate Installation and Patching of software on multiple servers
- Monitor system periodically
- Raise alarms and send notifications
- Troubleshooting and Audits

Writing our first shell script:

Creating a fle with .sh extension and then adding some texts and command in it to be executed when we execute the file then changing the file properties to be an executable file using (chmod) command to 755 then we execute the file

```
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ vi first.sh
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ cat first.sh
#!/bin/sh
# This is a comment!
echo Hello World        # This is a comment, too!
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ chmod 755 first.sh
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ ./first.sh
Hello World
rohit_ubuntu@camouflage:~$
```

variables:

```
rohit_ubuntu@camouflage:~/test$ vi variable_2.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat variable_2.sh
#!/bin/sh
MY_MESSAGE="Hello World"
echo $MY_MESSAGE
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 variable_2.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./variable_2.sh
Hello World
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$
```

```
rohit_ubuntu@camouflage:~/test$ vi variable_2_2.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat variable_2_2.sh
#!/bin/sh
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I will create you a file called ${USER_NAME}_file"
touch "${USER_NAME}_file"

rohit_ubuntu@camouflage:~/test$ chmod 755 variable_2_2.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./variable_2_2.sh
What is your name?
Rohit
Hello Rohit
I will create you a file called Rohit_file
rohit_ubuntu@camouflage:~/test$ ls
Rohit_file  first.sh  variable_2.sh  variable_2_2.sh
rohit_ubuntu@camouflage:~/test$
```

```
rohit_ubuntu@camouflage:~/test$ vi escape_character_3.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat escape_character_3.sh
echo "This is \" a quote and this is \\ a backslash"
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 escape_character_3.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./escape_character_3.sh
This is " a quote and this is \ a backslash
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$
```

For Loops:

```
rohit_ubuntu@camouflage:~/test$ vi for_loops_4.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat for_loops_4.sh
#!/bin/sh
for i in hello 1 * 2 goodbye
do
          echo "Looping ... i is set to $i"
done
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 for_loops_4.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./for_loops_4.sh
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to Rohit_file
Looping ... i is set to escape_character_3.sh
Looping ... i is set to first.sh
Looping ... i is set to for_loops_4.sh
Looping ... i is set to variable_2.sh
Looping ... i is set to variable_2_2.sh
Looping ... i is set to 2
Looping ... i is set to goodbye
rohit_ubuntu@camouflage:~/test$
```

While loops:

```
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ vi while_loops_5.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat while_loops_5.sh
#!/bin/sh
while :
do
          echo "Please type something in (^C to quit)"
          read INPUT_STRING
          echo "You typed: $INPUT_STRING"
done
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 while_loops_5.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./while_loops_5.sh
Please type something in (^C to quit)
Hey there,this is from the user input to the while loop
You typed: Hey there,this is from the user input to the while loop
Please type something in (^C to quit)
^C
rohit_ubuntu@camouflage:~/test$
```

test:

```
rohit_ubuntu@camouflage:~/test$ vi test_6.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat test_6.sh
echo -en "Please guess the magic number: "
read X
echo $X | grep "[^0-9]" > /dev/null 2>&1
if [ "$?" -eq "0" ]; then
          # If the grep found something other than 0-9
          # then it's not an integer.
          echo "Sorry, wanted a number"
      else
          # The grep found only 0-9, so it's an integer.
          # We can safely do a test on it.
          if [ "$X" -eq "7" ]; then
                echo "You entered the magic number!"
          fi
fi
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 test_6.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./test_6.sh
Please guess the magic number: 5
rohit_ubuntu@camouflage:~/test$ ./test_6.sh
Please guess the magic number: 7
You entered the magic number!
rohit_ubuntu@camouflage:~/test$
```

case:

```
rohit_ubuntu@camouflage:~/test$ vi case_7.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat case_7.sh
#!/bin/sh

echo "Please talk to me ..."
while :
do
        read INPUT_STRING
        case $INPUT_STRING in
                hello)
                        echo "Hello yourself!"
                        ;;
                bye)
                        echo "See you again!"
                        break
                        ;;
                *)echo "Sorry, I don't understand";;
        esac
done
echo
echo "That's all folks!"
```

```
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 case_7.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./case_7.sh
Please talk to me ...
hello
Hello yourself!

Sorry, I don't understand

Sorry, I don't understand
bye'
Sorry, I don't understand

Sorry, I don't understand
bye
See you again!

That's all folks!
```

functions:

```
rohit_ubuntu@camouflage:~/test$ vi functions_8.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat functions_8.sh
#!/bin/sh

myfunc()
{
        echo "\$1 is $1"
        echo "\$2 is $2"
        # cannot change $1 - we'd have to say:
        # 1="Goodbye Cruel"
        # which is not a valid syntax. However, we can
        # change $a:
        a="Goodbye Cruel"
              }

### Main script starts here

a=Hello
b=World
myfunc $a $b
echo "a is $a"
echo "b is $b"
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 functions_8.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./functions_8.sh
$1 is Hello
$2 is World
a is Goodbye Cruel
b is World
rohit_ubuntu@camouflage:~/test$
```

functions_recursion:

```
rohit_ubuntu@camouflage:~/test$ vi functions_8_recurssion.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ cat functions_8_recurssion.sh
#!/bin/sh

factorial()
{
        if [ "$1" -gt "1" ]; then
                i=`expr $1 - 1`
                j=`factorial $i`
                k=`expr $1 \* $j`
                echo $k
        else
                echo 1
        fi
}

while :
do
        echo "Enter a number:"
        read x
        factorial $x
done
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ chmod 755 functions_8_recurssion.sh
rohit_ubuntu@camouflage:~/test$
rohit_ubuntu@camouflage:~/test$ ./functions_8_recurssion.sh
Enter a number:
3
6
Enter a number:
5
120
Enter a number:
0
1
Enter a number:
^C
rohit_ubuntu@camouflage:~/test$
```

When to use Functions?

- Break up large script that performs many different tasks:
- Installing packages
- Adding users
- Configuring firewalls
- Perform Mathematical calculations

ANSIBLE

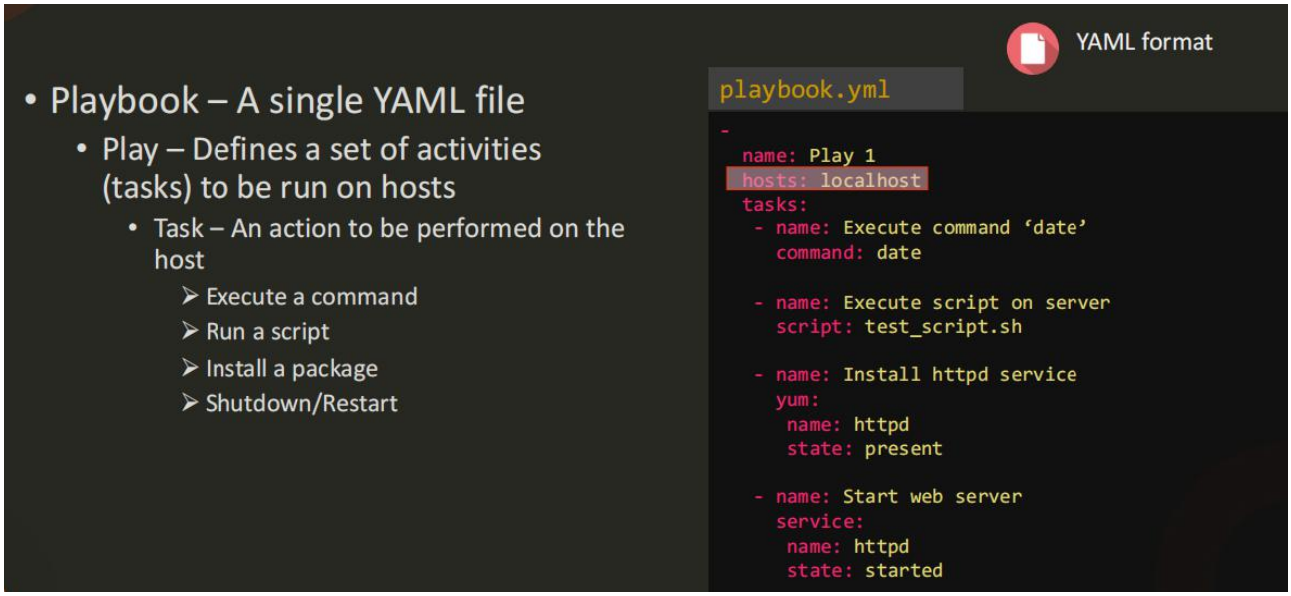
What is Ansible? Ansible is an open-source IT automation engine, which can remove drudgery from your work life, and will also dramatically improve the scalability, consistency, and reliability of your IT environment.

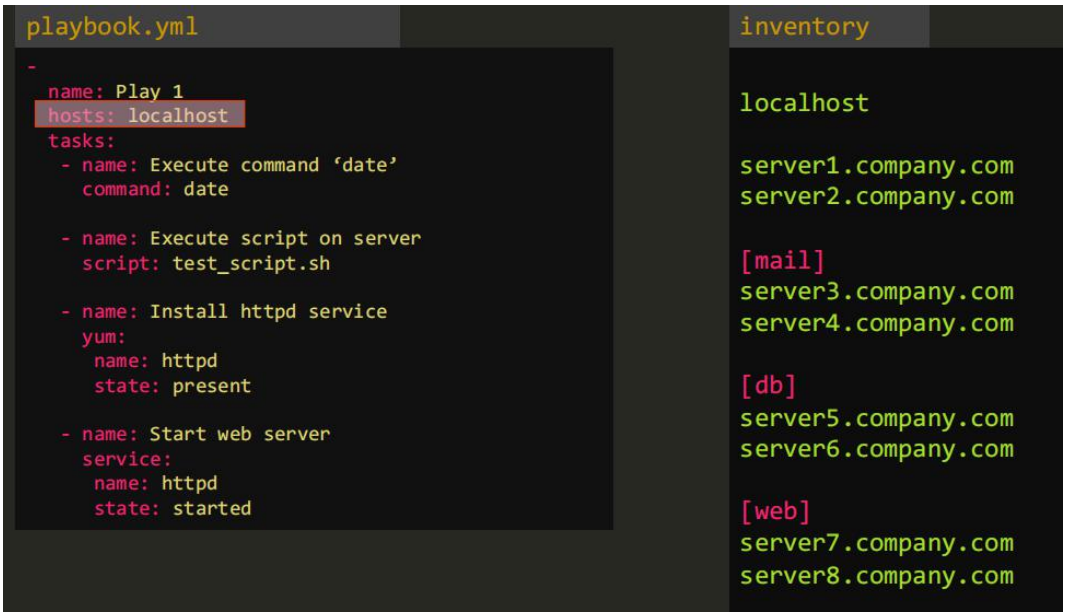
Why Ansible

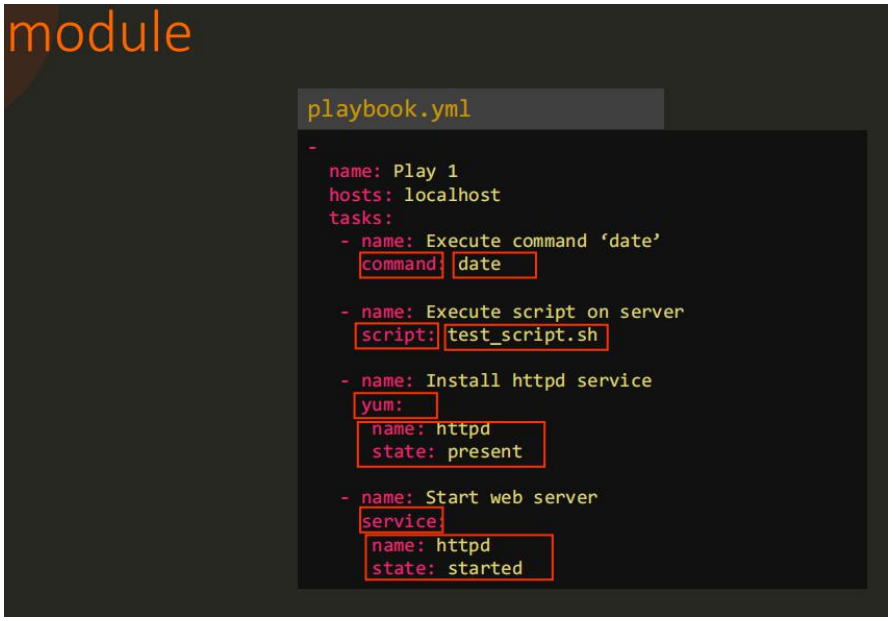
Ansible automates and simplifies repetitive, complex, and tedious operations. Everybody likes it because it brings huge time savings when we install packages or configure large numbers of servers. Its architecture is simple and effective. It works by connecting to your nodes and pushing small programs to them.

Installing Ansible:

Playbook:


YAML format
• Playbook – A single YAML file
• Play – Defines a set of activities (tasks) to be run on hosts
• Task – An action to be performed on the host
➢ Execute a command
➢ Run a script
➢ Install a package
➢ Shutdown/Restart
playbook.yml
- name: Play 1
hosts: localhost
tasks:
- name: Execute command 'date'
command: date
- name: Execute script on server
script: test_script.sh
- name: Install httpd service
yum:
name: httpd
state: present
- name: Start web server
service:
name: httpd
state: started


playbook.yml
- name: Play 1
hosts: localhost
tasks:
- name: Execute command 'date'
command: date
- name: Execute script on server
script: test_script.sh
- name: Install httpd service
yum:
name: httpd
state: present
- name: Start web server
service:
name: httpd
state: started
inventory
localhost
server1.company.com
server2.company.com
[mail]
server3.company.com
server4.company.com
[db]
server5.company.com
server6.company.com
[web]
server7.company.com
server8.company.com


module
playbook.yml
- name: Play 1
hosts: localhost
tasks:
- name: Execute command 'date'
command: date
- name: Execute script on server
script: test_script.sh
- name: Install httpd service
yum:
name: httpd
state: present
- name: Start web server
service:
name: httpd
state: started

Run:

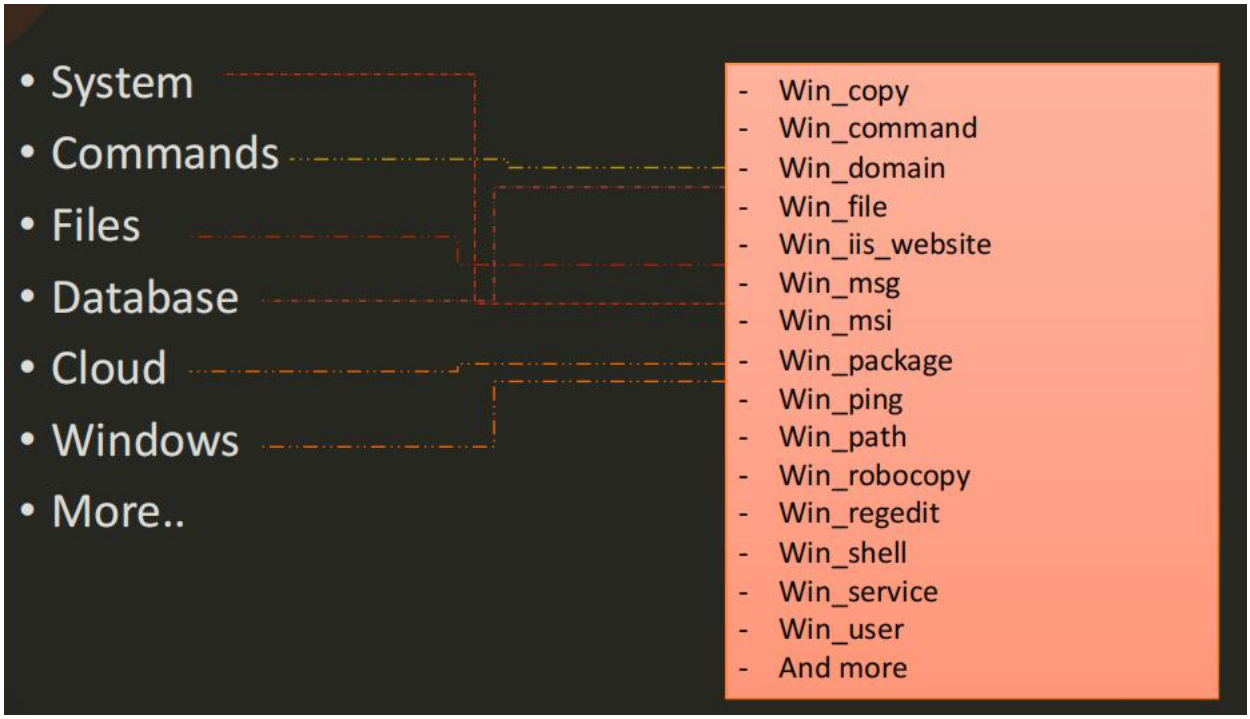
Execute Ansible Playbook

- Syntax:

  ansible-playbook <playbook file name>

  Ex: ansible-playbook playbook.yml

Modules:



Command:

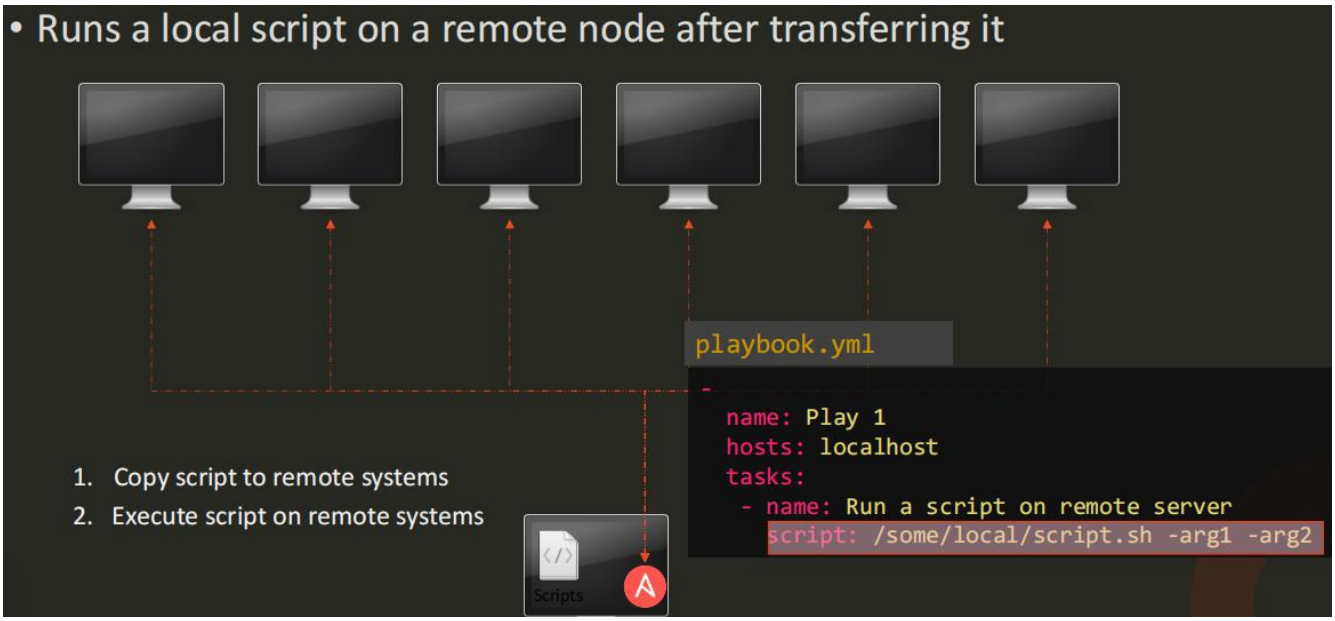
Executes a command on a remote node

parameter	comments
chdir	cd into this directory before running the command
creates	a filename or (since 2.0) glob pattern, when it already exists, this step will **not** be run.
executable	change the shell used to execute the command. Should be an absolute path to the executable.
free_form	the command module takes a free form command to run. There is no parameter actually named 'free form'. See the examples!
removes	a filename or (since 2.0) glob pattern, when it does not exist, this step will **not** be run.
warn (added in 1.8)	if command warnings are on in ansible.cfg, do not warn about this particular line if set to no/false.

playbook.yml

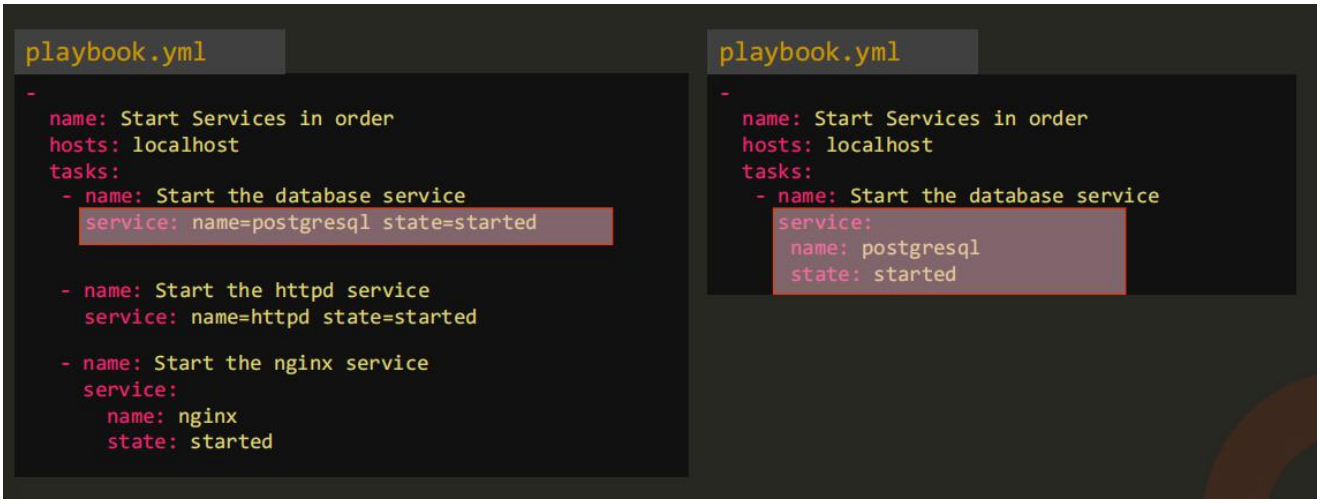
```
-
  name: Play 1
  hosts: localhost
  tasks:
   - name: Execute command 'date'
     command: date

   - name: Display resolv.conf contents
     command: cat /etc/resolv.conf

   - name: Display resolv.conf contents
     command: cat resolv.conf chdir=/etc

   - name: Display resolv.conf contents
     command: mkdir /folder creates=/folder
```

Script:



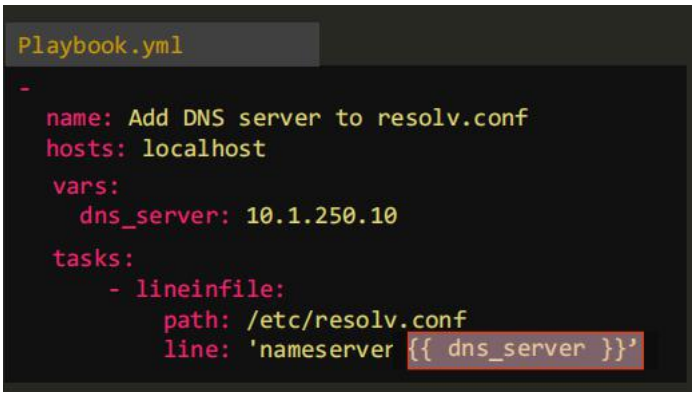
Services:

Manage services: Start, Stop and Restart



Variables:

Stores information that varies with each host



Loops:

```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name= '{{ item.name }}' state=present  uid='{{ item.uid }}'
     loop:
       - name: joe      -   { name: joe, uid: 1010 }
         uid: 1010
       - name: george   -   { name: george, uid: 1011 }
         uid: 1011
       - name: ravi     -   { name: ravi, uid: 1012 }
         uid: 1012
       - name: mani     -   { name: mani, uid: 1013 }
         uid: 1013
       - name: kiran    -   { name: kiran, uid: 1014 }
         uid: 1014
       - name: jazlan   -   { name: jazlan, uid: 1015 }
         uid: 1015
       - name: emaan    -   { name: emaan, uid: 1016 }
         uid: 1016
       - name: mazin    -   { name: mazin, uid: 1017 }
         uid: 1017
       - name: izaan    -   { name: izaan, uid: 1018 }
         uid: 1018
       - name: mike     -   { name: mike, uid: 1019 }
```

```
-
  name: Create users
  hosts: localhost
  tasks:
  -  var:
      item:
        name: joe
        uid: 1010
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: george
        uid: 1011
     user: name='{{ item.name }}' state=present uid='{{ item.uid

  -  var:
      item:
        name: ravi
        uid: 1012
     user: name='{{ item.name }}' state=present uid='{{ item.uid
  -  var:
      item:
        name: mani
        uid: 1013
     user: name='{{ item.name }}' state=present uid='{{ item.uid
```

With_*:

```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'    state=present
     loop:
       - joe
       - george
       - ravi
       - mani
```

```
-
  name: Create users
  hosts: localhost
  tasks:
   - user: name='{{ item }}'    state=present
     with_items:
       - joe
       - george
       - ravi
       - mani
```

Conditionals - when:

```
---
- name: Install NGINX
  hosts: all
  tasks:
  - name: Install NGINX on Debian
    apt:
      name: nginx
      state: present
    when:  ansible_os_family == "Debian"

  - name: Install NGINX on Redhat
    yum:
      name: nginx
      state: present
    when:  ansible_os_family == "RedHat"
```

Operator - or :

```
---
- name: Install NGINX
  hosts: all
  tasks:
  - name: Install NGINX on Debian
    apt:
      name: nginx
      state: present
    when:  ansible_os_family == "Debian"

  - name: Install NGINX on Redhat
    yum:
      name: nginx
      state: present
    when:  ansible_os_family == "RedHat"  or
           ansible_os_family == "SUSE"
```

Operator - and :

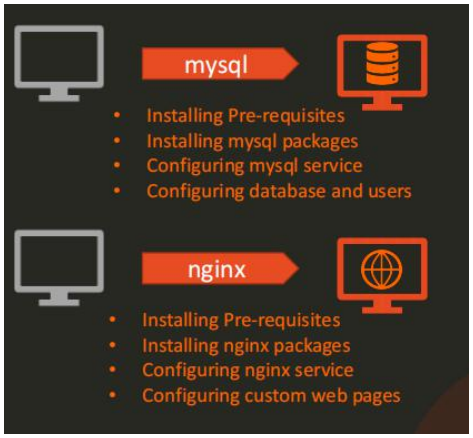
```
---
- name: Install NGINX
  hosts: all
  tasks:
  - name: Install NGINX on Debian
    apt:
      name: nginx
      state: present
    when:  ansible_os_family == "Debian"    and
           ansible_distribution_version == "16.04"

  - name: Install NGINX on Redhat
    yum:
      name: nginx
      state: present
    when:  ansible_os_family == "RedHat"  or
           ansible_os_family == "SUSE"
```

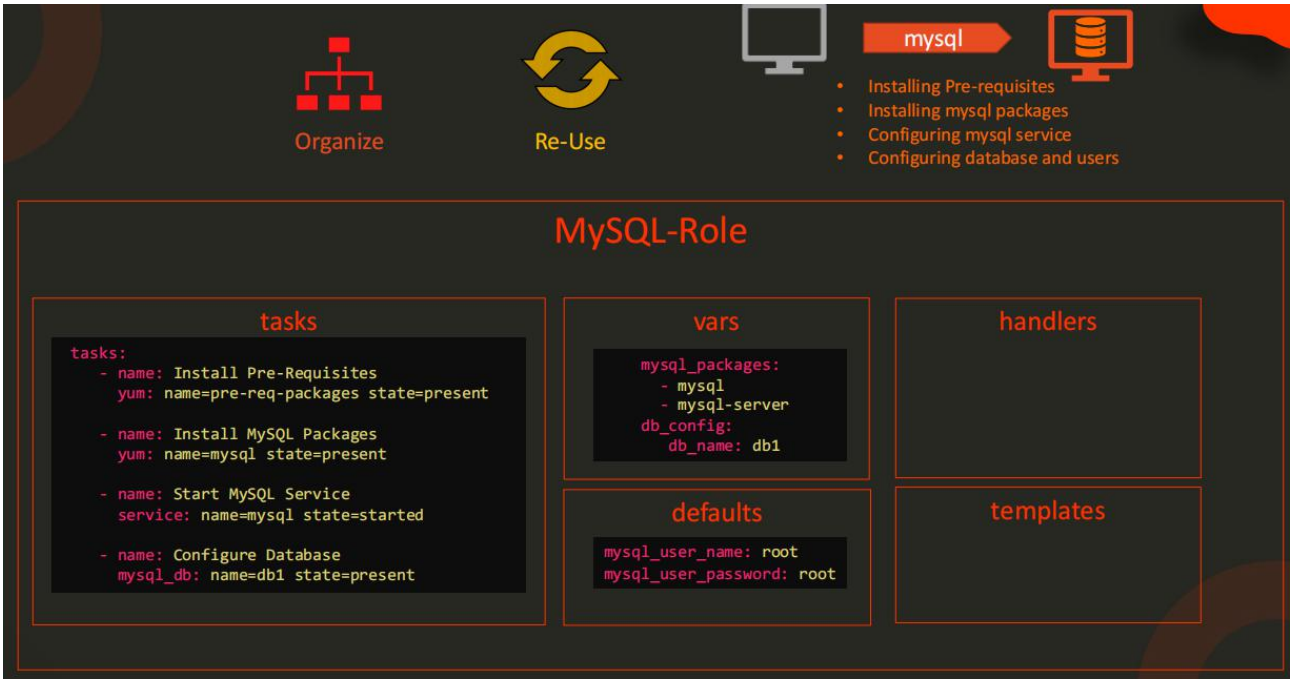
Conditionals in Loops:

```
---
- name: Install Softwares
  hosts: all
  vars:
     packages:
        - name: nginx
          required: True
        - name: mysql
          required : True
        - name: apache
          required : False
  tasks:
  - name: Install  "{{ item.name }}" on Debian
    apt:
      name: "{{ item.name }}"
      state: present

    when:   item.required == True

    loop: "{{ packages }}"
```

Roles:

```
- name: Install and Configure MySQL
  hosts: db-server
  tasks:
     - name: Install Pre-Requisites
       yum: name=pre-req-packages state=present

     - name: Install MySQL Packages
       yum: name=mysql state=present

     - name: Start MySQL Service
       service: name=mysql state=started

     - name: Configure Database
       mysql_db: name=db1 state=present
```





Use Roles:

```
$ ansible-galaxy install geerlingguy.mysql

- downloading role 'mysql', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-mysql/archive/2.9.5.tar.gz
- extracting geerlingguy.mysql to /etc/ansible/roles/geerlingguy.mysql
- geerlingguy.mysql (2.9.5) was installed successfully
```

playbook.yml

```
-
  name: Install and Configure MySQL
  hosts: db-server
  roles:
    - geerlingguy.mysql
```

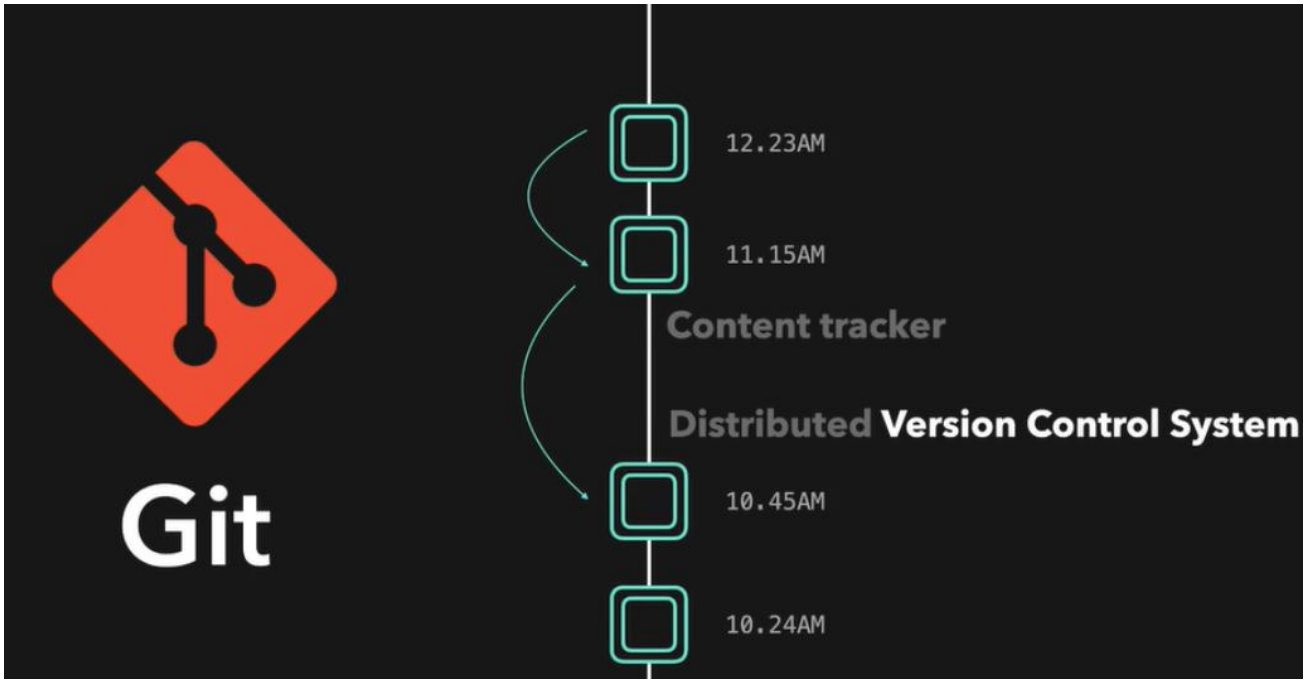
```
-
  name: Install and Configure MySQL
  hosts: db-server
  roles:
    - role: geerlingguy.mysql
      become: yes
      vars:
        mysql_user_name: db-user
```

List Roles:

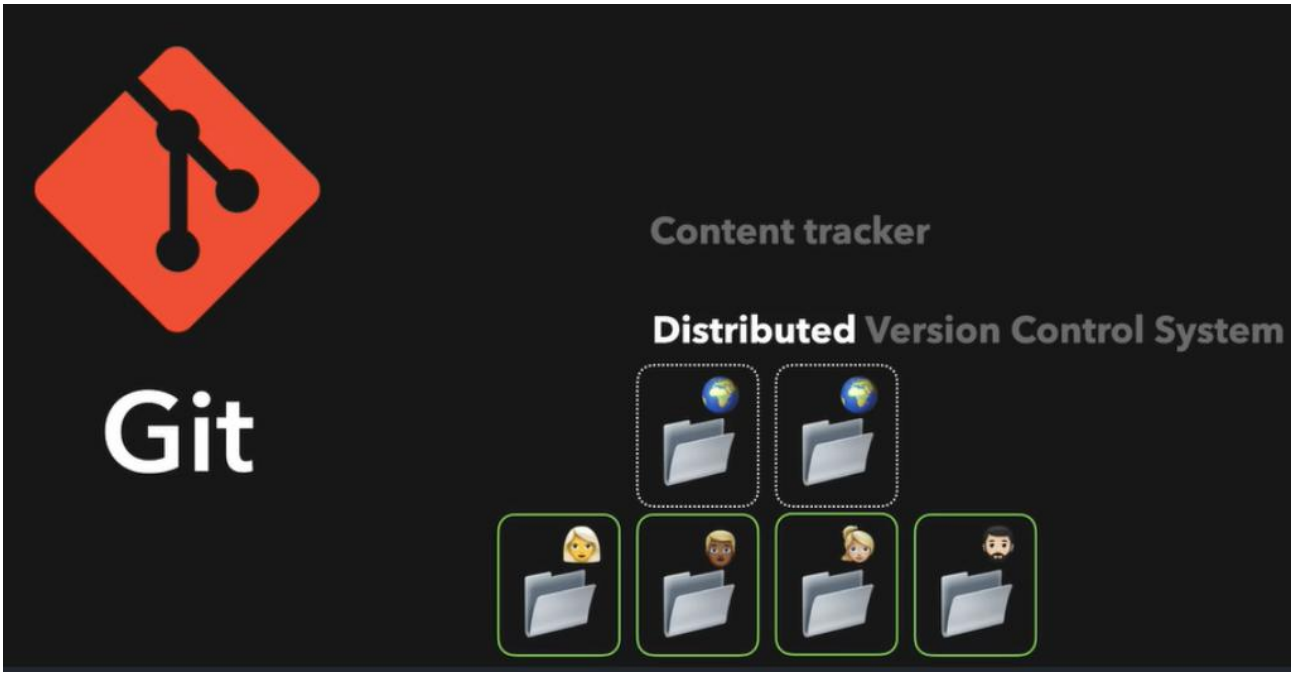
```
$ ansible-galaxy list

- geerlingguy.mysql
- kodekloud1.mysql

$ ansible-config dump | grep ROLE

EFAULT_PRIVATE_ROLE_VARS(default) = False
DEFAULT_ROLES_PATH(default) = [u'/root/.ansible/roles', u'/usr/share/ansible/roles', u'/etc/ansible/roles']
GALAXY_ROLE_SKELETON(default) = None
GALAXY_ROLE_SKELETON_IGNORE(default) = ['^.git$', '^.*/.git_keep$']

$ ansible-galaxy install geerlingguy.mysql -p ./roles
```

GIT

Git is a Distributed Version Control System. it is a content tracker, it stores all our code changes.



Distributed means it has a remote repository which is stored in a server and a local repository which is stored on the computer of every developer working on a project. Every developer has a full copy of the code base.



In this section, we will take a look at Installing Git:

Download the Git binaries for your specific machine
https://git-scm.com/downloads

Installing git for Linux
https://git-scm.com/download/linux

Installing git for Mac
https://git-scm.com/download/mac

Installing git for windows
https://git-scm.com/download/win

Once, installed to check the version of the git run the below command
$ git --version

Let's initialize a local git repository:

Go into a project folder and initialize git

$ cd /Users//Desktop/myproject
$ git init
We have initizalized an empty git repository in a .git folder

To list all the contents of the folder including the hidden folder such as .git folder
$ ls -a

Let's add a file to a project
Create a file with a basic sentence
$ touch story1.txt
$ echo "This is a beautiful story" >> story1.txt

To see the status of git:
$ git status

git status:

- Default branch will be master branch
- Untracked files are files that are not added to the git

git add:

To add a file to the staging area

$ git add story1.txt

git commit:

To commit the changes

$ git commit -m "Added first story"

git Log:

To know about other commits information such as commit hash, the author name and the date. To easily show commit details as one line

Git Branches:

In this section, we will take a look at git branches

Branches:

Keep project versioned using branch

A branch is basically a pointer to the last commit

If you are working on a feature, you might work on a feature branch (eg. feature/signup), once the features are tested, you can merge to master branch

To create a new branch

$ git checkout Rohit

To create a new branch and switch to it

$ git checkout -b Rohit

To list of all of our branches

$ git branch

Switch to exisiting branch

$ git checkout Rohit

Delete a branch

$ git branch -d max

Git Merging Branches:

In this section, we will take a look at git merging braches

We can merge a branch with 'git merge' command

$ git checkout master

$ git merge feature/signup

Two types of merge that git can perform

- **Fast forward merge**
- **No-Fast Forward merge**

Initialize Remote Repositories:

In this section, we will take a look at initializing remote repositories
We can push code to the remote repositories that is hosted at somewhere else and get this code on our local machines by pulling this information.
There are several platforms where we can host our remote repositories. Most commonly used one are the below

- Github
- Gitlab
- Bitbucket

Once we initialize a repository on those platform, we will get access to something called connection string.
A connection string is the URL that we can use in order to let git know where the remote repository is located.

To add a remote repository to a local project

$ git add remote origin <connection URL>

To list all remote repositories

$ git remote -v

Pushing to remote repositories
In this section, we will take a look at pushing to remote repositories
In order to keep our local and remote repo in sync, we have to push the data from local repo to remote repo

To push data from local to remote repo

$ git push origin master

Cloning Remote Repositories:
In this section, we will take a look at cloning remote repositories

We can clone the remote repository on our local machines

To clone remote repo with ssh link

$ git clone <ssh-link-goes-here>

$ git clone git@github.com:account/remote-repo.git

To check the history of the project

$ cd remote-repo

$ git log

Pull Requests:
In this section, we will take a look at Pull requests

To push the latest changes to github Rohit branch

$ git push origin Rohit

To push changes from Rohit to master we have to open something called Pull Request

Fetching and Pulling:
In this section, we will take a look at fetching and pulling

To update remote repository in our local repo

$ git fetch origin master

To update local master branch to point to the latest changes made on remote branch(origin/master). To merge origin/master into local master branch

$ git merge origin/master

To pull the remote branch

$ git pull origin/master

Merge Conflicts:
In this section, we will take a look at merge conflicts

Remove the lines that we don't wanna keep to resolve the conficts and save the file.
Add the changes to git again

$ git add fourth_story.txt

Fork:
How do you create a pull request if you are not part of a git project? One way to contribute such projects is to fork the main project.
After forking the project you can add your changes to a branch on the fork copy and send a pull request to the original project to merge your changes

Git Rebasing:
In this section, we will take a look at git rebasing.

Git merge:
We can merge the master branch into our own branch by git merge command.
This creates a new merge commit, to merge these changes into the our own branch. Another way to achieving this, by rebasing branches.

$ git merge master

Git rebase:
Basically in the rebasing, we are putting branch into top of the another branch one.
After performing this, current branch is containing all the changes that were made in the master branch.

$ git rebase master

In the GIT, each commit has the unique identifier. This unique identifier is a hash that contains the information about the commit.
When its merge, this unique identifer won't be modified. In the other words, we are not modifying the history of the GIT commits. When we are merging.
When we are rebasing, actually we are copying commits from one branch to the another branch. Each time a hash value will update.

Git Interactive Rebasing:
In this section, we will take a look at git interactive-rebasing.

- When all commits are looking same and should be added into the single commit.
- We can change the history of the git branch within the interactive rebase. To access this we use -i flag with git rebase command.
- We have to specify which commits we want to update.
- Suppose we wants to modify first four commits. We are telling to git to rebase last 4 commits.

  $ git rebase -i HEAD~4

It will open the file into the editor, we need to change "pick" to "squash" command. After changing the commands save the file and exit. After this, it will combined the commits. There are so many other options when you are interactively rebasing. It's very powerful tool to changes to your branch and commits**.**

Git Cherry Picking:
In this section, we will take a look at git cherry-pick command.

- One branch has certain changes and you would like to apply into the other branch. But you don't want to all the changes that you made in that branch. In the certain things, we use cherry-pick command to copy of that commits onto own branch.
- We use hash of the commit, that we want to use for a cherry-pick. After running this command, it will create a copy of the commit into the specified branch.

  $ git cherry-pick aaba5

Git Resetting and Reverting:
In this section, we will take a took at **git reset** and **git revert** command.

Everyone makes a mistake, in some situation we don't want to commit certain things but do. So there is a several options to undo that commit.
One of the option is **git revert** command. Second one is **git reset** command.

Git revert:
git revert command creates a new commit, which literally reverse the only changes that we made on the commit that we specified. A **git revert** command is useful if you want to undo the changes and keep those changes in your GIT history.

$ git revert <commit-id>

Git reset:
In the **git reset** command, there is two ways to reset the commit. Either with the **--soft** flag which we wants to keep the changes that we made or over the **--hard** flag in order to loose all the changes that we made on that commit.

 # soft reset
$ git reset --soft HEAD~1

 # hard reset
$ git reset --hard HEAD~1

When we reset the commit with **--soft** flag, we still have the access to changes that we made by that commit. We can see that status by **git status** command. We can easily create an another commit this way.

$ git reset --soft HEAD~1
$ git status
On branch Rohit
Changes to be committed:
added: third_story.txt # file shown with git status command

In the **--hard** flag, it will reset the commit without saving all those changes.

$ git reset --hard HEAD~1
$ git status
On branch Rohit
Nothing to commit

Git Stashing:
In this section, we will take a took at git stash command.

git stash:
If we don't want to commit now, we can stash all changes in the working area with git stash command. Modification in the working area, all get added to the stash.

$ git stash

With git stash pop command, to get back changes in our working area.

$ git stash pop

To see the list of stash files, we can run the following command:

$ git stash list

To see the content of the specific stash file, we can run the following command:

$ git stash show stash@{1}

To pop the specific stash, we can run the following command:

$ git stash pop stash@{2}

Git Reflog:
In this section, we will take a took at **git reflog** command.

If commit is not necessary at all and wants to remove then we can run the **git reset --hard** command: -

$ git reset --hard HEAD~1
After running this command, that data will be gone forever. But we don't need to be panic at all, **git reflog** command shows the all actions that have been taken to the repository. This includes resets, reverts and merges.

$ git reflog

You can easily undo the mistakes you made in the repository that information reflog command gives us.
You can get the hash value from the **git reflog** command as we already see in the previous process.

$ git reset --hard 8ad5

After this repository has been set into the previous state.

You can see that **git reflog** status also changed.

$ git reflog

git log and git reflog may look smiliar. git log will show you only information about the commits not the status about repository that git reflog does.

$ git log

$ git reflog

Conclusion:

In this session, we will take a quick recap of all the concepts that we learned.

git install:

Git installing with the software package manager brew.

$ brew install git

git init:

After initialize it git-beginner directory with git init command. Git uses to store data.

$ git init

Initialized empty Git repository in /root/git-beginner/.git/

git remote:

We also covered, how we can initialize our remote repository to save our data in the remote server.

$ git remote add origin https://.../.../[name].git

git push:

We also learned about how we can push our data into the remote repository.

$ git push origin master

git clone:

We saw that how we can clone remote repository data into our local machine.

$ git clone https://.../.../[name].git

git branch:

We can work on our own version of the project by creating a new branch.

$ git branch checkout -b

git commit:

To commit the changes.

$ git commit -m 'First commit'

git merge:
To merge the commit changes into the other branch.

$ git merge feature

git rebase:
git rebase command creates a different GIT history compare to merging.

$ git rebase master

We also learned interactive rebasing, to get more control over the commits that we are rebasing. It will change the commits before rebasing them.

$ git rebase -i HEAD~4

git revert:
We also saw how we could do undo changes. We have to write a couple of characters of commit hash id.

$ git revert 8ad58

git reset:
git reset is a bit harsh. If we reset with --soft flag changes still available in the working area and if we do with --hard flag, it will remove the changes forever.

$ git reset --soft HEAD~1

$ git reset --hard HEAD~1

We also learned how we could open the pull request in the Github platform, where's your team members can review your work.
Git is an extremely powerful tool. I hope this course has given you a Git introduction of the main concepts.

JENKINS

Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

Installing Jenkins:

The procedures in this chapter are for new installations of Jenkins.
Jenkins is typically run as a standalone application in its own process with the built-in Java servlet container/application server (Jetty).
Jenkins can also be run as a servlet in different Java servlet containers such as Apache Tomcat or GlassFish. However, instructions for setting up these types of installations are beyond the scope of this chapter.

Jenkins installers are available for several Linux distributions.

- Debian/Ubuntu
- Fedora
- Red Hat / CentOS

Red Hat / CentOS:
You can install Jenkins through yum on Red Hat Enterprise Linux, CentOS, and other Red Hat based distributions. You need to choose either the Jenkins Long Term Support release or the Jenkins weekly release.

Long Term Support release
A LTS (Long-Term Support) release is chosen every 12 weeks from the stream of regular releases as the stable release for that time period. It can be installed from the redhat-stable yum repository.

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
   https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key
sudo yum upgrade
sudo yum install epel-release java-11-openjdk-devel
sudo yum install jenkins
sudo systemctl daemon-reload
```

Start Jenkins:

You can start the Jenkins service with the command:

sudo systemctl start jenkins

You can check the status of the Jenkins service using the command:

sudo systemctl status jenkins

If everything has been set up correctly, you should see an output like this:

Loaded: loaded (/etc/rc.d/init.d/jenkins; generated)
Active: active (running) since Tue 2021-09-15 16:19:01 +03; 3min 17s ago
…

Post-installation setup wizard:

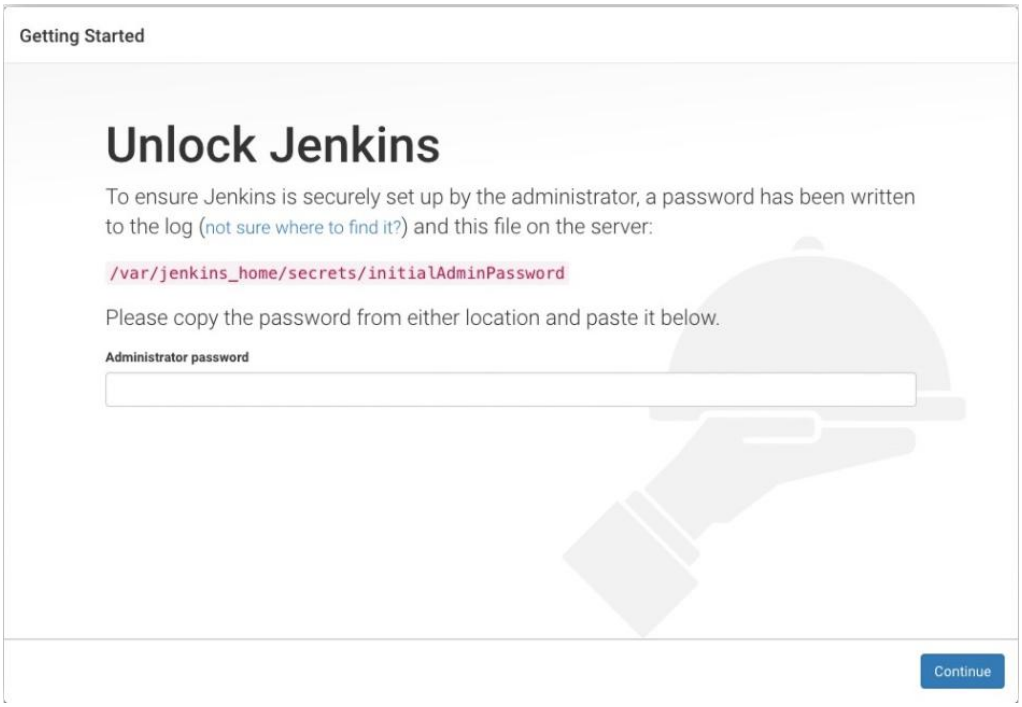
After downloading, installing and running Jenkins using one of the procedures above, the post-installation setup wizard begins.

This setup wizard takes you through a few quick "one-off" steps to unlock Jenkins, customize it with plugins and create the first administrator user through which you can continue accessing Jenkins.

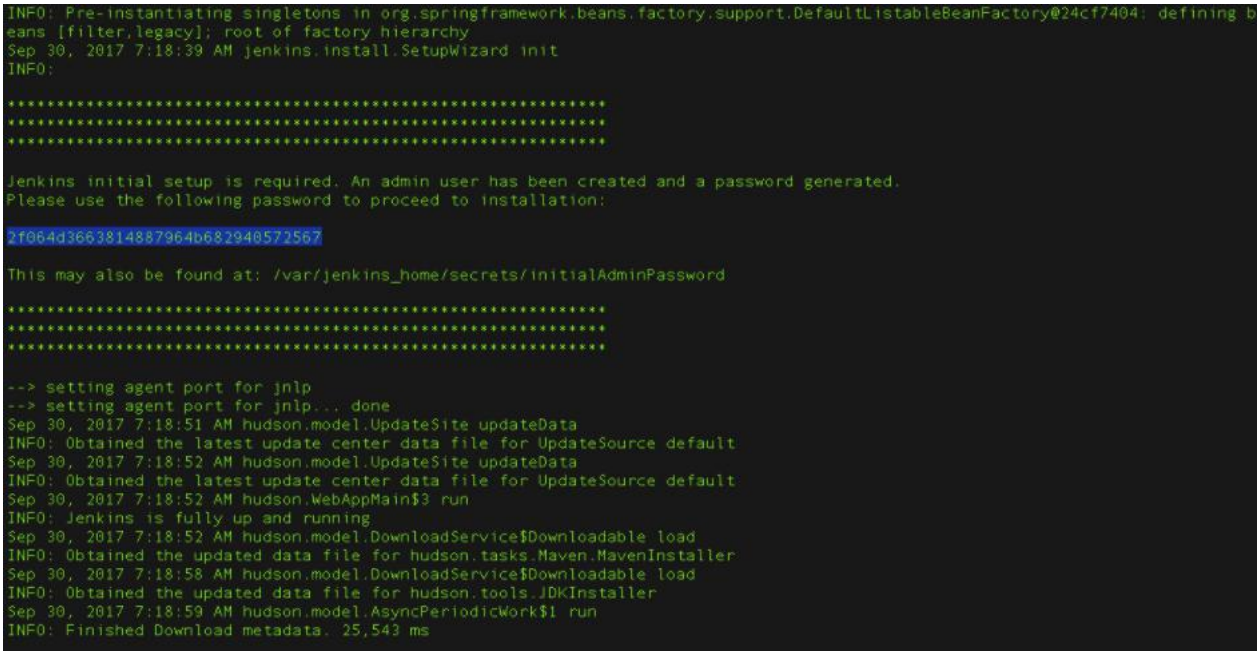
Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

- Browse to http://localhost:8080 (or whichever port you configured for Jenkins when installing it) and wait until the Unlock Jenkins page appears.



- From the Jenkins console log output, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).



Note:

- The command: **sudo cat /var/lib/jenkins/secrets/initialAdminPassword** will print the password at console.
- If you are running Jenkins in Docker using the official **jenkins/jenkins** image you can use **sudo docker exec ${CONTAINER_ID or CONTAINER_NAME}cat/var/jenkins_home/secrets/initialAdminPassword** to print the password in the console without having to exec into the container.

On the Unlock Jenkins page, paste this password into the **Administrator password** field and click **Continue.Notes:**

- You can always access the Jenkins console log from the Docker logs (above).
- The Jenkins console log indicates the location (in the Jenkins home directory) where this password can also be obtained. This password must be entered in the setup wizard on new Jenkins installations before you can access Jenkins's main UI. This password also serves as the default admininstrator account's password (with username "admin") if you happen to skip the subsequent user-creation step in the setup wizard.

Customizing Jenkins with plugins:

After unlocking Jenkins, the **Customize Jenkins** page appears. Here you can install any number of useful plugins as part of your initial setup.

Click one of the two options shown:

- **Install suggested plugins -** to install the recommended set of plugins, which are based on most common use cases.

- **Select plugins to install -** to choose which set of plugins to initially install. When you first access the plugin selection page, the suggested plugins are selected by default.

Creating the first administrator user:

Finally, after customizing Jenkins with plugins, Jenkins asks you to create your first administrator user.

- When the Create First Admin User page appears, specify the details for your administrator user in the respective fields and click Save and Finish.
- When the Jenkins is ready page appears, click Start using Jenkins.

Notes:

  - This page may indicate Jenkins is almost ready! instead and if so, click Restart.
  - If the page does not automatically refresh after a minute, use your web browser to refresh the page manually.

- If required, log in to Jenkins with the credentials of the user you just created and you are ready to start using Jenkins!

Using Jenkins :

This chapter contains topics for typical Jenkins users (of all skill levels) about Jenkins usage which is outside the scope of the core Jenkins features: Pipeline and Blue Ocean.
If you want to create and configure a Pipeline project through a Jenkinsfile or through Blue Ocean, or you wish to find out more about these core Jenkins features, refer to the relevant topics within the respective Pipeline and Blue Ocean chapters.
If you are a Jenkins administrator and want to know more about managing Jenkins nodes and instances, see Managing Jenkins.If you are a system administrator and want learn how to back-up, restore, maintain as Jenkins servers and nodes, see Jenkins System Administration**.**

- Using credentials
- Search Box
- Referencing another project by name
- Aborting a build
- Fingerprints
- Using local language
- Change time zone
- Remote Access API
- Executor Starvation
- Using Jenkins agents
- Using JMeter with Jenkins

Pipeline:

This chapter covers all recommended aspects of Jenkins Pipeline functionality, including how to:

- get started with Pipeline - covers how to define a Jenkins Pipeline (i.e. your Pipeline) through Blue Ocean, through the classic UI or in SCM,
- create and use a Jenkinsfile - covers use-case scenarios on how to craft and construct your Jenkinsfile,
- work with branches and pull requests,
- use Docker with Pipeline - covers how Jenkins can invoke Docker containers on agents/nodes (from a Jenkinsfile) to build your Pipeline projects,
- extend Pipeline with shared libraries,
- use different development tools to facilitate the creation of your Pipeline, and
- work with Pipeline syntax - this page is a comprehensive reference of all Declarative Pipeline syntax.

For an overview of content in the Jenkins User Handbook, see User Handbook overview.

What is Jenkins Pipeline?

Jenkins Pipeline (or simply "Pipeline" with a capital "P") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. [1]

The definition of a Jenkins Pipeline is written into a text file (called a Jenkinsfile) which in turn can be committed to a project's source control repository. [2] This is the foundation of "Pipeline-as-code"; treating the CD pipeline a part of the application to be versioned and reviewed like any other code.

Creating a Jenkinsfile and committing it to source control provides a number of immediate benefits:

- Automatically creates a Pipeline build process for all branches and pull requests.
- Code review/iteration on the Pipeline (along with the remaining source code).
- Audit trail for the Pipeline.
- Single source of truth [3] for the Pipeline, which can be viewed and edited by multiple members of the project.

While the syntax for defining a Pipeline, either in the web UI or with a Jenkinsfile is the same, it is generally considered best practice to define the Pipeline in a Jenkinsfile and check that in to source control.

Declarative versus Scripted Pipeline syntax:

A Jenkinsfile can be written using two types of syntax - Declarative and Scripted. Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which:

- provides richer syntactical features over Scripted Pipeline syntax, and
- is designed to make writing and reading Pipeline code easier.

Many of the individual syntactical components (or "steps") written into a Jenkinsfile, however, are common to both Declarative and Scripted Pipeline. Read more about how these two types of syntax differ in Pipeline concepts and Pipeline syntax overview below.

Why Pipeline?

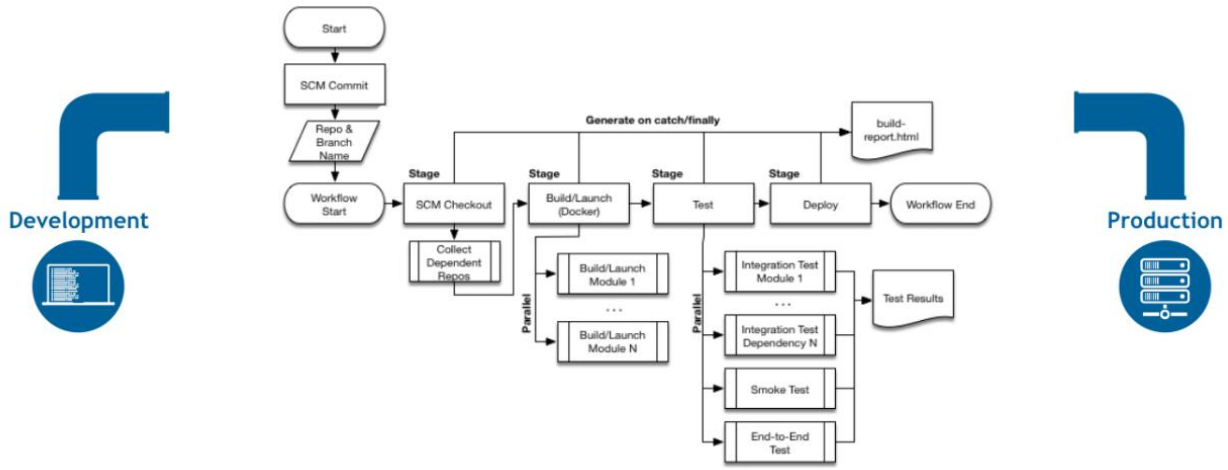
Jenkins is, fundamentally, an automation engine which supports a number of automation patterns. Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines. By modeling a series of related tasks, users can take advantage of the many features of Pipeline:

- **Code**: Pipelines are implemented in code and typically checked into source control, giving teams the ability to edit, review, and iterate upon their delivery pipeline.
- **Durable**: Pipelines can survive both planned and unplanned restarts of the Jenkins controller.
- **Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.
- **Versatile**: Pipelines support complex real-world CD requirements, including the ability to fork/join, loop, and perform work in parallel.
- **Extensible**: The Pipeline plugin supports custom extensions to its DSL [1] and multiple options for integration with other plugins.

While Jenkins has always allowed rudimentary forms of chaining Freestyle Jobs together to perform sequential tasks, [4] Pipeline makes this concept a first-class citizen in Jenkins.

Building on the core Jenkins value of extensibility, Pipeline is also extensible both by users with Pipeline Shared Libraries and by plugin developers.

The flowchart below is an example of one CD scenario easily modeled in Jenkins Pipeline:



Pipeline concepts:

- **Pipeline**

A Pipeline is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.
Also, a pipeline block is a key part of Declarative Pipeline syntax.

- **Node**

A node is a machine which is part of the Jenkins environment and is capable of executing a Pipeline.
Also, a node block is a key part of Scripted Pipeline syntax.

- **Stage**

A stage block defines a conceptually distinct subset of tasks performed through the entire Pipeline (e.g. "Build", "Test" and "Deploy" stages), which is used by many plugins to visualize or present Jenkins Pipeline status/progress. [6]

- **Step**

A single task. Fundamentally, a step tells Jenkins what to do at a particular point in time (or "step" in the process). For example, to execute the shell command make use the sh step: sh 'make'. When a plugin extends the Pipeline DSL, [1] that typically means the plugin has implemented a new step.

Pipeline syntax overview

The following Pipeline code skeletons illustrate the fundamental differences between Declarative Pipeline syntax and Scripted Pipeline syntax.

Be aware that both stages and steps (above) are common elements of both Declarative and Scripted Pipeline syntax.

Declarative Pipeline fundamentals
In Declarative Pipeline syntax, the pipeline block defines all the work done throughout your entire Pipeline.

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```

Using a Jenkinsfile:

This section builds on the information covered in Getting started with Pipeline and introduces more useful steps, common patterns, and demonstrates some non-trivial Jenkinsfile examples.
Creating a Jenkinsfile, which is checked into source control , provides a number of immediate benefits:

- Code review/iteration on the Pipeline
- Audit trail for the Pipeline
- Single source of truthfor the Pipeline, which can be viewed and edited by multiple members of the project.

Pipeline supports two syntaxes, Declarative and Scripted Pipeline. Both of which support building continuous delivery pipelines. Both may be used to define a Pipeline in either the web UI or with a Jenkinsfile, though it's generally considered a best practice to create a Jenkinsfile and check the file into the source control repository

Creating a Jenkinsfile:

As discussed in the Defining a Pipeline in SCM, a Jenkinsfile is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. Consider the following Pipeline which implements a basic three-stage continuous delivery pipeline.

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Build:

For many projects the beginning of "work" in the Pipeline would be the "build" stage. Typically this stage of the Pipeline will be where source code is assembled, compiled, or packaged. The Jenkinsfile is not a replacement for an existing build tool such as GNU/Make, Maven, Gradle, etc, but rather can be viewed as a glue layer to bind the multiple phases of a project's development lifecycle (build, test, deploy, etc) together.

Jenkins has a number of plugins for invoking practically any build tool in general use, but this example will simply invoke make from a shell step (sh). The sh step assumes the system is Unix/Linux-based, for Windows-based systems the bat could be used instead.

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'make'
                archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
            }
        }
    }
}
```

Test:

Running automated tests is a crucial component of any successful continuous delivery process. As such, Jenkins has a number of test recording, reporting, and visualization facilities provided by a number of plugins. At a fundamental level, when there are test failures, it is useful to have Jenkins record the failures for reporting and visualization in the web UI. The example below uses the junit step, provided by the JUnit plugin.

In the example below, if tests fail, the Pipeline is marked "unstable", as denoted by a yellow ball in the web UI. Based on the recorded test reports, Jenkins can also provide historical trend analysis and visualization.

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Test') {
            steps {
                /* `make check` returns non-zero on test failures,
                * using `true` to allow the Pipeline to continue nonetheless
                */
                sh 'make check || true'
                junit '**/target/*.xml'
            }
        }
    }
}
```

Deploy:

Deployment can imply a variety of steps, depending on the project or organization requirements, and may be anything from publishing built artifacts to an Artifactory server, to pushing code to a production system.

At this stage of the example Pipeline, both the "Build" and "Test" stages have successfully executed. In essence, the "Deploy" stage will only execute assuming previous stages completed successfully, otherwise the Pipeline would have exited early.

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any

    stages {
        stage('Deploy') {
            when {
              expression {
                currentBuild.result == null || currentBuild.result == 'SUCCESS'
              }
            }
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Working with your Jenkinsfile

The following sections provide details about handling:
specific Pipeline syntax in your Jenkinsfile and
features and functionality of Pipeline syntax which are essential in building your application or Pipeline project.

Using environment variables:

Jenkins Pipeline exposes environment variables via the global variable env, which is available from anywhere within a Jenkinsfile. The full list of environment variables accessible from within Jenkins Pipeline is documented at ${YOUR_JENKINS_URL}/pipeline-syntax/globals#env and includes:

BUILD_ID

The current build ID, identical to BUILD_NUMBER for builds created in Jenkins versions 1.597+

BUILD_NUMBER
The current build number, such as "153"

BUILD_TAG
String of jenkins-${JOB_NAME}-${BUILD_NUMBER}. Convenient to put into a resource file, a jar file, etc for easier identification

BUILD_URL
The URL where the results of this build can be found (for example http://buildserver/jenkins/job/MyJobName/17/)

EXECUTOR_NUMBER
The unique number that identifies the current executor (among executors of the same machine) performing this build. This is the number you see in the "build executor status", except that the number starts from 0, not 1

JAVA_HOME
If your job is configured to use a specific JDK, this variable is set to the JAVA_HOME of the specified JDK. When this variable is set, PATH is also updated to include the bin subdirectory of JAVA_HOME

JENKINS_URL
Full URL of Jenkins, such as https://example.com:port/jenkins/ (NOTE: only available if Jenkins URL set in "System Configuration")

JOB_NAME
Name of the project of this build, such as "foo" or "foo/bar".

NODE_NAME
The name of the node the current build is running on. Set to 'master' for the Jenkins controller.

WORKSPACE
The absolute path of the workspace

DOCKER

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

What can I use Docker for?
Fast, consistent delivery of your applications

Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.

Consider the following example scenario:

- Your developers write code locally and share their work with their colleagues using Docker containers.
- They use Docker to push their applications into a test environment and execute automated and manual tests.
- When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
- When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Responsive deployment and scaling

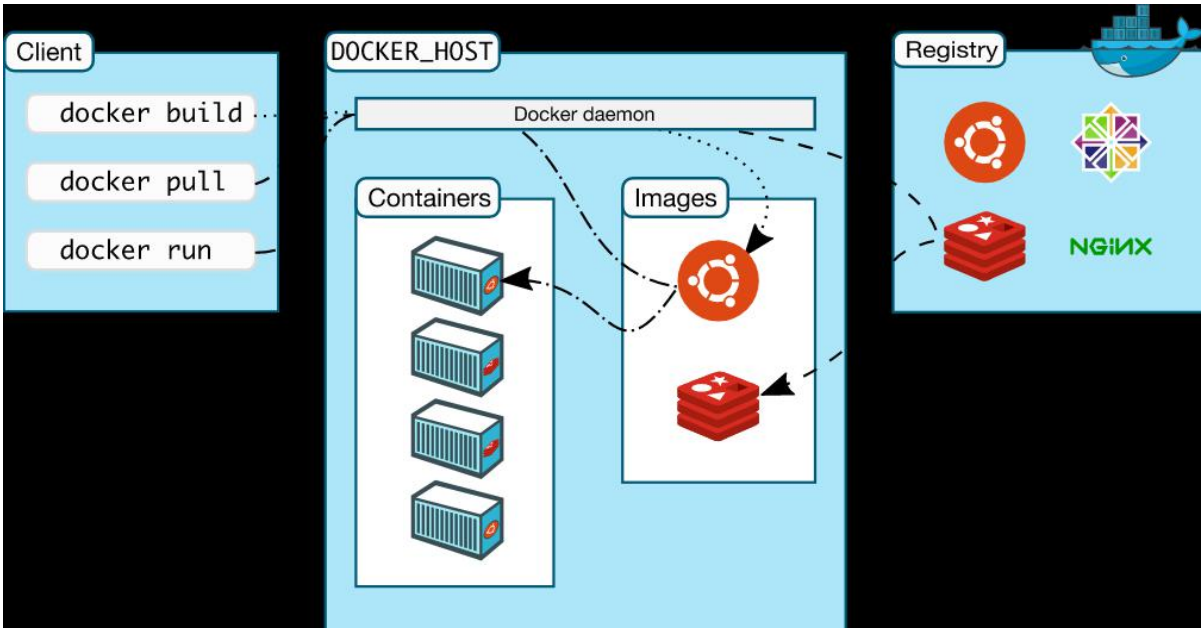
Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.

Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.

Running more workloads on the same hardware

Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your compute capacity to achieve your

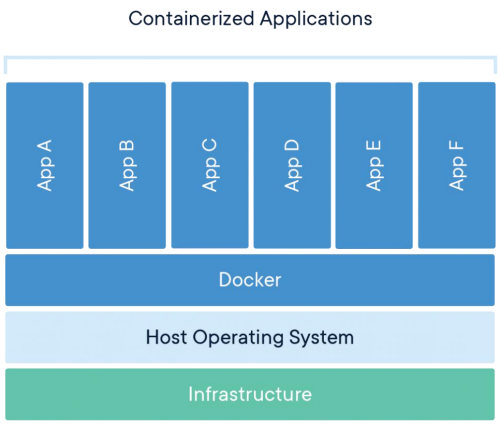
business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.



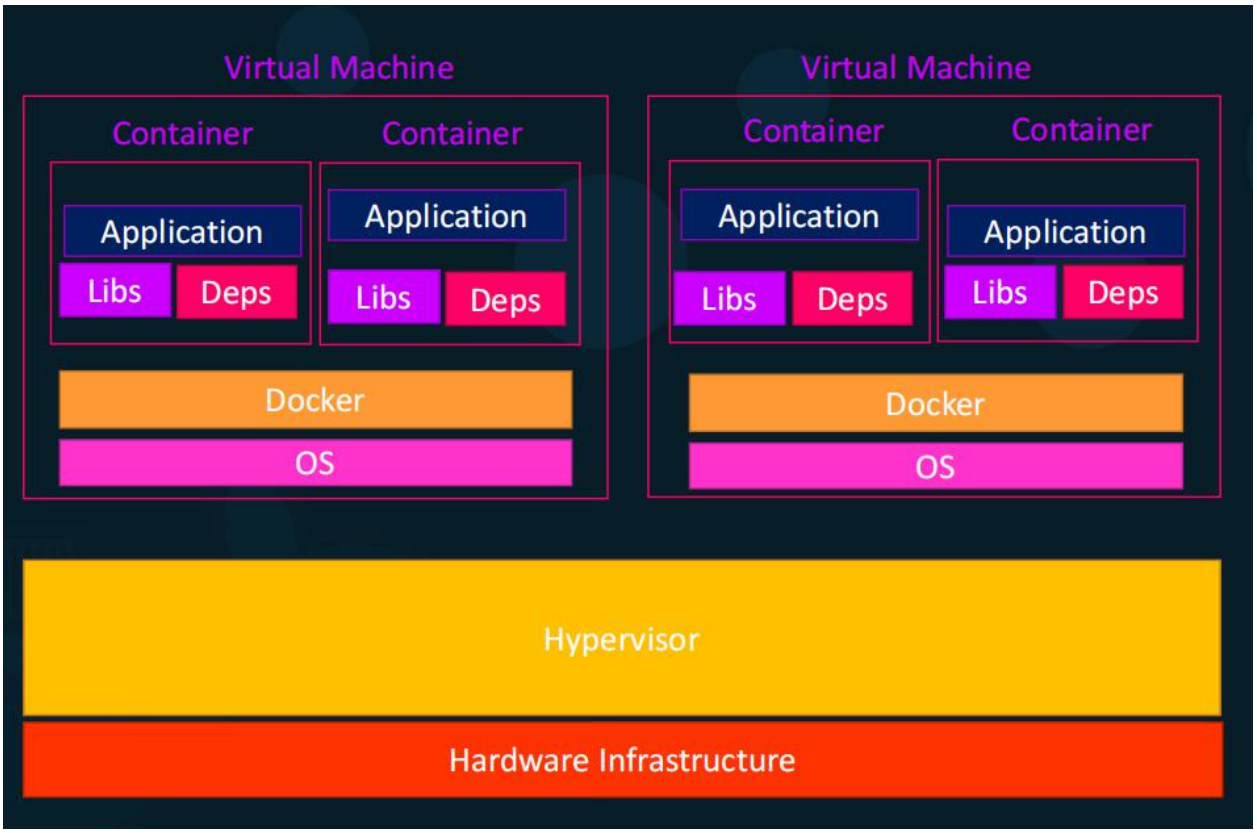
What are containers:

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will always run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.



Containers VS virtual machines:



What is an Image:

A Docker image is a file used to execute code in a Docker container. Docker images act as a set of instructions to build a Docker container, like a template. Docker images also act as the starting point when using Docker. An image is comparable to a snapshot in virtual machine (VM) environments.

A Docker image has everything needed to run a containerized application, including code, config files, environment variables, libraries and runtimes. When the image is deployed to a Docker environment, it can be executed as a Docker container. The docker run command creates a container from a specific image.

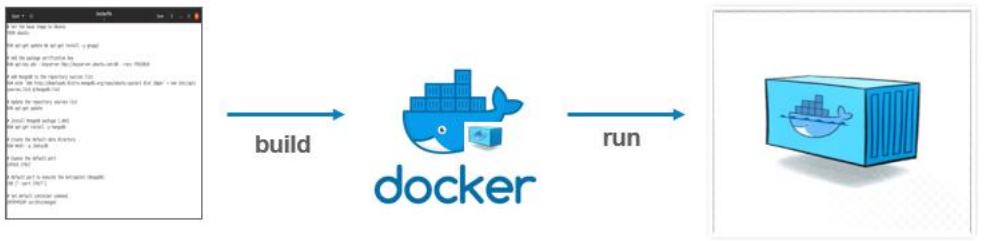
Docker images are a reusable asset -- deployable on any host. Developers can take the static image layers from one project and use them in another. This saves the user time, because they do not have to recreate an image from scratch.

Docker container vs. Docker image:

A Docker container is a virtualized runtime environment used in application development. It is used to create, run and deploy applications that are isolated from the underlying hardware. A Docker container can use one machine, share its kernel and virtualize the OS to run more isolated processes. As a result, Docker containers are lightweight.

A Docker image is like a snapshot in other types of VM environments. It is a record of a Docker container at a specific point in time. Docker images are also immutable. While they can't be changed, they can be duplicated, shared or deleted. The feature is useful for testing new software or configurations because whatever happens, the image remains unchanged.

Containers need a runnable image to exist. Containers are dependent on images, because they are used to construct runtime environments and are needed to run an application



Anatomy of a Docker image:

A Docker image has many layers, and each image includes everything needed to configure a container environment -- system libraries, tools, dependencies and other files. Some of the parts of an image include:

- **Base image:** The user can build this first layer entirely from scratch with the build command.
- **Parent image**: As an alternative to a base image, a parent image can be the first layer in a Docker image. It is a reused image that serves as a foundation for all other layers.
- **Layers:** Layers are added to the base image, using code that will enable it to run in a container. Each layer of a Docker image is viewable under /var/lib/docker/aufs/diff, or via the Docker history command in the command-line interface (CLI). Docker's default status is to show all top-layer images, including repository, tags and file sizes. Intermediate layers are cached, making top layers easier to view. Docker has storage drives that handle the management of image layer contents.
- **Container layer:** A Docker image not only creates a new container, but also a writable or container layer. This layer hosts changes made to the running container and stores newly written and deleted files, as well as changes to existing files. This layer is also used to customize containers.
- **Docker manifest**: This part of the Docker image is an additional file. It uses JSON format to describe the image, using information such as image tags and digital signature.

Docker image repositories

Docker images get stored in private or public repositories, such as those in the Docker Hub cloud registry service, from which users can deploy containers and test and share images. Docker Hub's Docker Trusted Registry also provides image management and access control capabilities.

Official images are ones Docker produces, while community images are images Docker users create. CoScale agent is an official Docker image that monitors Docker applications. Datadog/docker-dd-agent, a Docker container for agents in the Datadog log management program, is an example of a community Docker image.

Users can also create new images from existing ones and use the docker push command to upload custom images to the Docker Hub.

Docker commands:

Running a container:

```
docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Already exists
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

List containers:

Running:

```
docker ps
CONTAINER ID  IMAGE   COMMAND                  CREATED         STATUS         PORTS     NAMES
796856ac413d  nginx   "nginx -g 'daemon of…"   7 seconds ago   Up 6 seconds   80/tcp    silly_sammet
```

All:

```
docker ps -a
CONTAINER ID   IMAGE   COMMAND                  CREATED          STATUS                       NAMES
796856ac413d   nginx   "nginx -g 'daemon of…"   7 seconds ago    Up 6 seconds                 silly_sammet
cff8ac918a2f   redis   "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago     relaxed_aryabhata
```

Stop containers:

```
docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
796856ac413d   nginx     "nginx -g 'daemon of…"   7 seconds ago    Up 6 seconds    80/tcp    silly_sammet
```

```
docker stop silly_sammet
silly_sammet
```

```
docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      NAMES
796856ac413d   nginx     "nginx -g 'daemon of…"   7 seconds ago    Exited (0) 3 seconds ago    silly_sammet
cff8ac918a2f   redis     "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago    relaxed_aryabhata
```

Remove containers:

```
docker rm silly_sammet
silly_sammet
```

```
docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      NAMES
cff8ac918a2f   redis     "docker-entrypoint.s…"   6 seconds ago    Exited (0) 3 seconds ago    relaxed_aryabhata
```

List Images:

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx               latest              f68d6e55e065        4 days ago          109MB
redis               latest              4760dc956b2d        15 months ago       107MB
ubuntu              latest              f975c5035748        16 months ago       112MB
alpine              latest              3fd9065eaf02        18 months ago       4.14MB
```

Remove images:

```
docker rmi nginx
Untagged: nginx:latest
Untagged: nginx@sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Deleted: sha256:f68d6e55e06520f152403e6d96d0de5c9790a89b4cfc99f4626f68146fa1dbdc
Deleted: sha256:1b0c768769e2bb66e74a205317ba531473781a78b77feef8ea6fd7be7f4044e1
Deleted: sha256:34138fb60020a180e512485fb96fd42e286fb0d86cf1fa2506b11ff6b945b03f
Deleted: sha256:cf5b3c6798f77b1f78bf4e297b27cfa5b6caa982f04caeb5de7d13c255fd7a1e
```

NOTE: but before deleting the images delete all dependent containers to remove image

Pull an image:

```
docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Already exists
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

```
docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
fc7181108d40: Pull complete
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

Append a command:

```
docker run ubuntu sleep 5
```

Exec a command:

```
docker exec distracted_mcclintock cat /etc/hosts
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.2      538d037f94a7
```

Run : attach and detach:

```
rohit_ubuntu@camouflage:~$ docker run nginx:1.13
^Crohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ docker run -d nginx:1.13
79f17bdf052df1a8fc1abe319ab97493013db32d2cd2ebacf53f5aa7ed67a25d
rohit_ubuntu@camouflage:~$
rohit_ubuntu@camouflage:~$ docker attach 79f17
```

Run - tag:

```
docker run redis:4.0        TAG
Unable to find image 'redis:4.0' locally
4.0: Pulling from library/redis
e44f086c03a2: Pull complete
Status: Downloaded newer image for redis:4.0

1:C 31 Jul 09:02:56.527 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 31 Jul 09:02:56.527 # Redis version=4.0.14, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 09:02:56.530 # Server initialized
```

Port-mapping:

```
$ docker run -p 127.0.0.1:80:8080/tcp ubuntu bash
```

Volume mapping:

```
docker run -v /opt/datadir:/var/lib/mysql mysql
```

Inspect container:

```
docker inspect blissful_hopper
[
    {
        "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
        "Name": "/blissful_hopper",
        "Path": "python",
        "Args": [
            "app.py"
        ],
        "State": {
            "Status": "running",
            "Running": true,
        },

        "Mounts": [],
        "Config": {
           "Entrypoint": [
                "python",
                "app.py"
            ],
        },
        "NetworkSettings": {..}
      }
]
```

Container logs:

```
docker logs blissful_hopper
 This is a sample web application that displays a colored background.
 A color can be specified in two ways.

 1. As a command line argument with --color as the argument. Accepts one of
red,green,blue,blue2,pink,darkblue
 2. As an Environment variable APP_COLOR. Accepts one of
red,green,blue,blue2,pink,darkblue
 3. If none of the above then a random color is picked from the above list.
 Note: Command line argument precedes over environment variable.


No command line argument or environment variable. Picking a Random Color =blue
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

Environment variables:

```python
import os
from flask import Flask

app = Flask(__name__)

…
…

color = "red"

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)


if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

```
docker run -e APP_COLOR=blue     simple-webapp-color
```


```
docker run  -e APP_COLOR=green simple-webapp-color
```


```
docker run  -e APP_COLOR=pink simple-webapp-color
```

Inspecting environment variables:

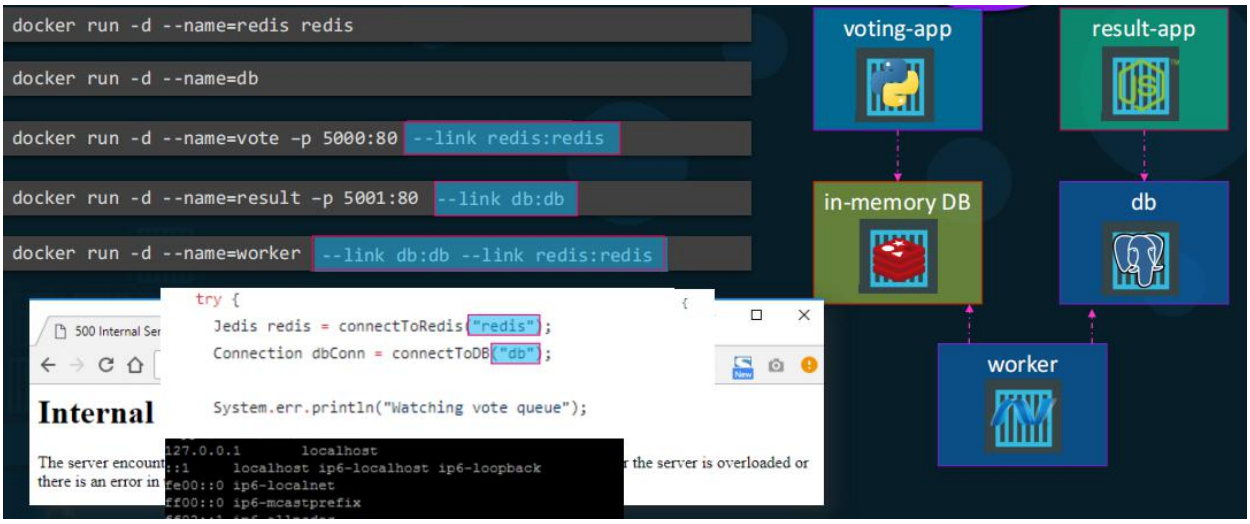
```
docker inspect blissful_hopper
```


```
[
    {
        "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
        "State": {
            "Status": "running",
            "Running": true,
        },

        "Mounts": [],
        "Config": {
            "Env": [
                "APP_COLOR=blue",
                "LANG=C.UTF-8",
                "GPG_KEY=0D96DF4D4110E5C43FBFB17F2D347EA6AA65421D",
                "PYTHON_VERSION=3.6.6",
                "PYTHON_PIP_VERSION=18.1"
            ],
            "Entrypoint": [
                "python",
                "app.py"
            ],
      }
]
```

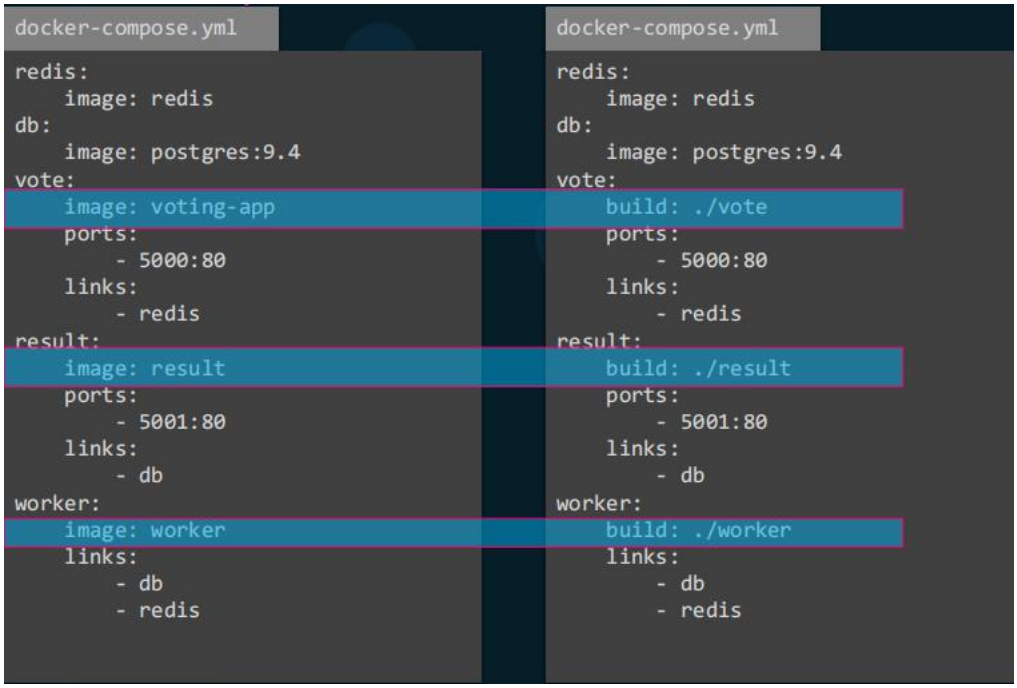

Docker compose:

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

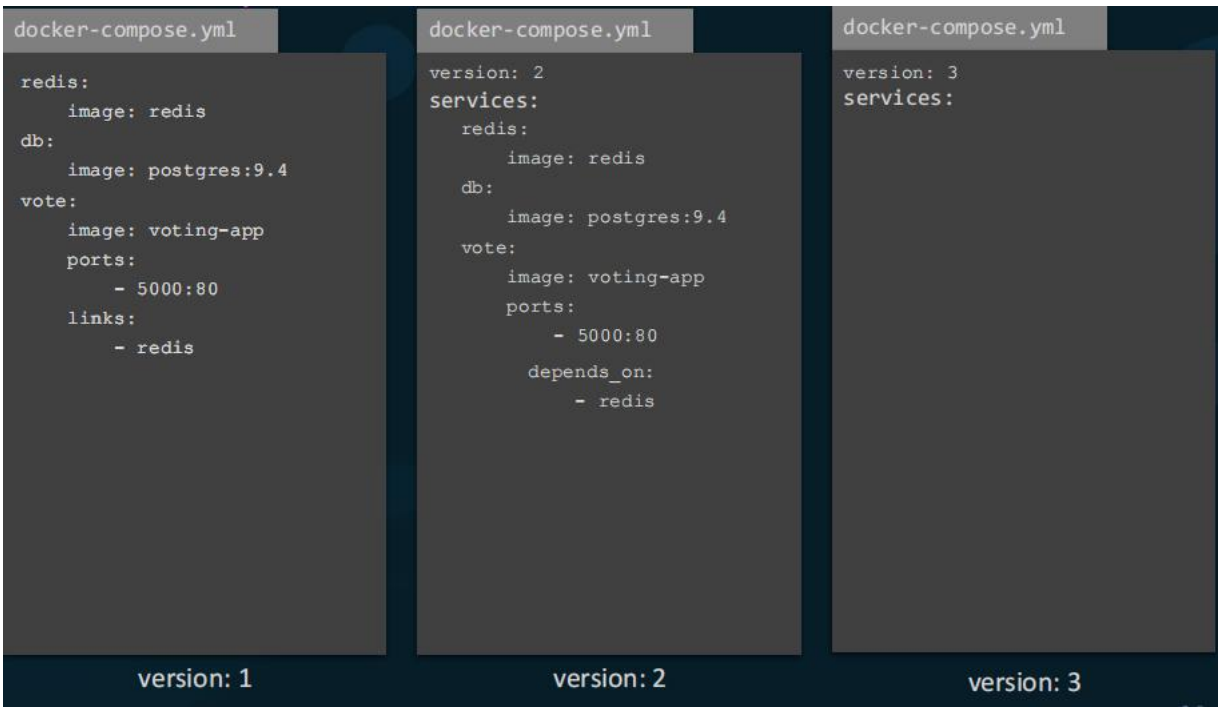
Understanding docker compose by deploying a voting application:



Creating a docker compose file:



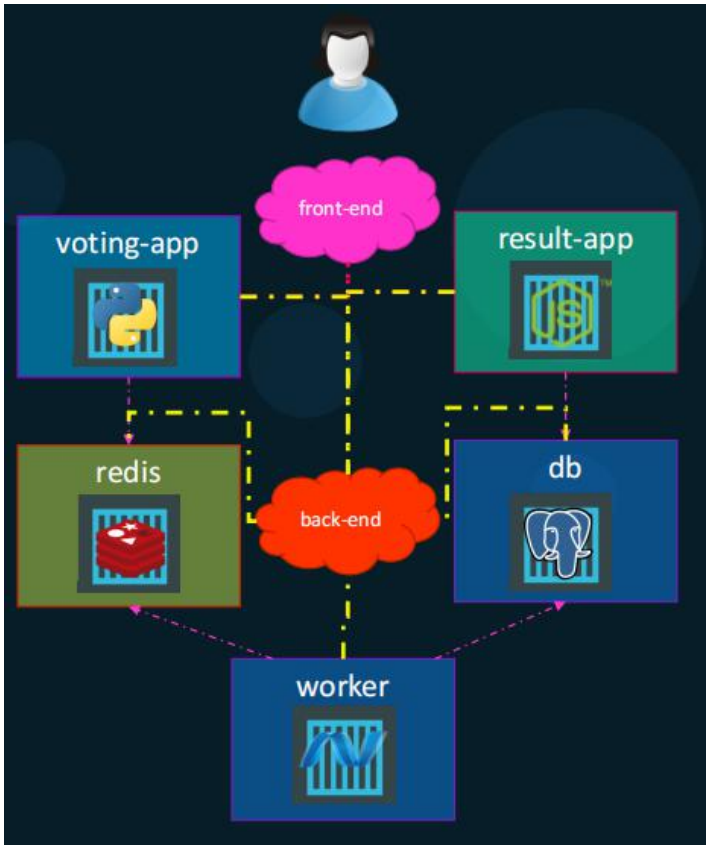
Versions of docker compose:



Now linking the elements according to the applications requirement:

docker-compose.yml

```
version: 2
services:
     redis:
         image: redis

          networks:
               - back-end
     db:
         image: postgres:9.4
         networks:
              - back-end
     vote:
         image: voting-app
         networks:
              - front-end
              - back-end
     result:
         image: result
         networks:
              - front-end
              - back-end

networks:
    front-end:
    back-end:
```



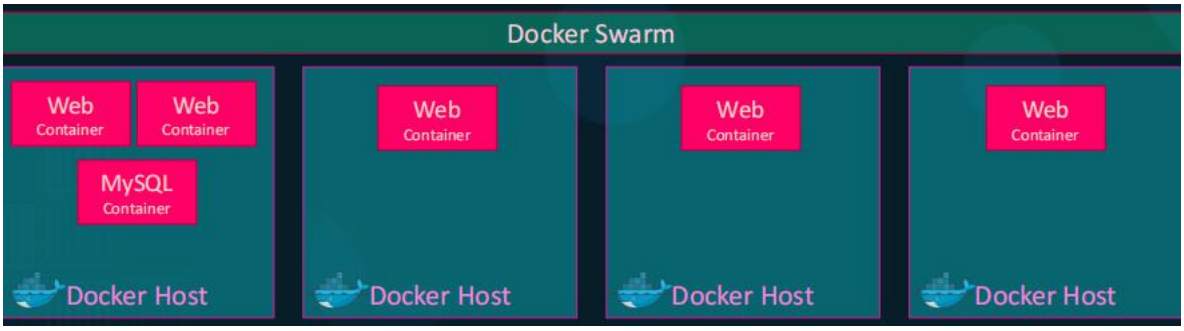
Container Orchestration and Docker SWARM:

Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more

Some of the container orchestration tools:

Docker SWARM:

Docker swarm is a container orchestration tool, meaning that it allows the user to manage multiple containers deployed across multiple host machines. One of the key benefits associated with the operation of a docker swarm is the high level of availability offered for applications.



What are Docker Swarm Nodes?

A docker swarm is comprised of a group of physical or virtual machines operating in a cluster. When a machine joins the cluster, it becomes a node in that swarm. The docker swarm function recognizes three different types of nodes, each with a different role within the docker swarm ecosystem:

Docker Swarm Manager Node:

The primary function of manager nodes is to assign tasks to worker nodes in the swarm. Manager nodes also help to carry out some of the managerial tasks needed to operate the swarm. Docker recommends a maximum of seven manager nodes for a swarm.

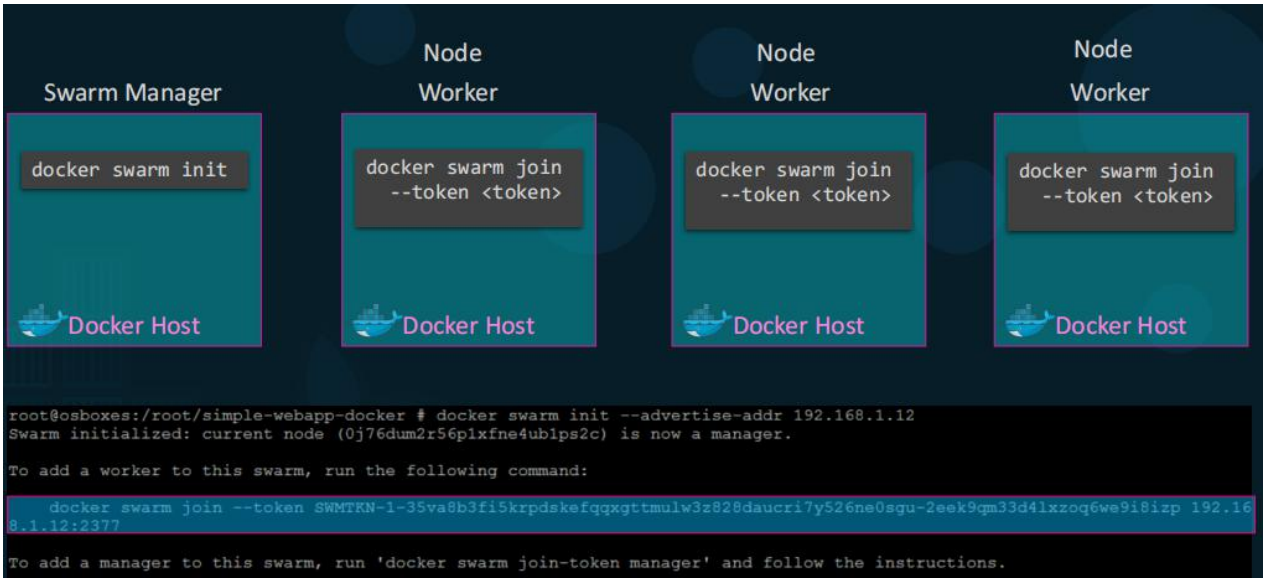
Docker Swarm Leader Node:

When a cluster is established, the Raft consensus algorithm is used to assign one of them as the "leader node". The leader node makes all of the swarm management and task orchestration decisions for the swarm. If the leader node becomes unavailable due to an outage or failure, a new leader node can be elected using the Raft consensus algorithm.

Docker Swarm Worker Node:

In a docker swarm with numerous hosts, each worker node functions by receiving and executing the tasks that are allocated to it by manager nodes. By default, all manager modes are also worker nodes and are capable of executing tasks when they have the resources available to do so.

Setting up docker SWARM:



Docker Swarm Benefits:

We're seeing an increasing number of developers adopt the Docker engine and use docker swarms to more efficiently produce, update and operate applications. Even software giants like Google are adopting container-based methodologies like docker swarm. Here are three simple reasons why Docker Swarms are becoming more popular:

- **Leverage the Power of Containers**

Developers love using docker swarm because it fully leverages the design advantages offered by containers. Containers allow developers to deploy applications or services in self-contained virtual environments, a task that was previous the domain of virtual machines. Containers are proving a more lightweight version of virtual machines, as their architecture allows them to make more efficient use of computing power.

- **Docker Swarm Helps Guarantee High Service Availability**

One of the main benefits of docker swarms is increasing application availability through redundancy. In order to function, a docker swarm must have a swarm manager that can assign tasks to worker nodes. By implementing multiple managers, developers ensure that the system can continue to function even if one of the manager nodes fails. Docker recommends a maximum of seven manager nodes for each cluster.

- **Automated Load-Balancing**

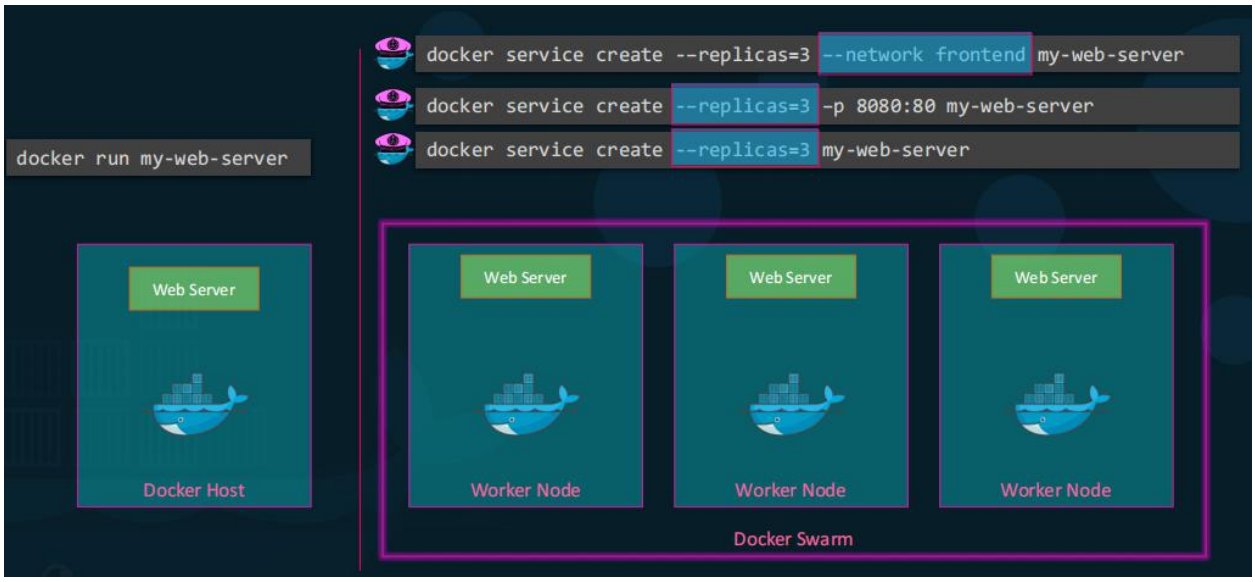
Docker swarm schedules tasks using a variety of methodologies to ensure that there are enough resources available for all of the containers. Through a process that can be described as automated load balancing, the swarm manager ensures that container workloads are assigned to run on the most appropriate host for optimal efficiency.

Docker service:

Docker service: Docker service will be the image for a microservice within the context of some larger application. Examples of services might include an HTTP server, a database, or any other type of executable program that you wish to run in a distributed environment.

When you create a service, you specify which container image to use and which commands to execute inside running containers. You also define options for the service including:

- The port where the swarm will make the service available outside the swarm
- An overlay network for the service to connect to other services in the swarm
- CPU and memory limits and reservations
- A rolling update policy
- The number of replicas of the image to run in the swarm



Cloud computing and AWS

What Is 'Cloud'? -And Why To Call It As "Cloud"?

"The cloud" refers to servers that are accessed over the Internet, and the software and databases that run on those servers. Cloud servers are located in data centers all over the world. By using cloud computing, users and companies don't have to manage physical servers themselves or run software applications on their own machines.

"The cloud" started off as a tech industry slang term. In the early days of the Internet, technical diagrams often represented the servers and networking infrastructure that make up the Internet as a cloud. As more computing processes moved to this servers-and-infrastructure part of the Internet, people began to talk about moving to "the cloud" as a shorthand way of expressing where the computing processes were taking place. Today, "the cloud" is a widely accepted term for this style of computing.

Understanding Cloud Computing:

• Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources
• Through a cloud services platform with pay-as-you-go pricing
• You can provision exactly the right type and size of computing resources you need
• You can access as many resources as you need, almost instantly
• Simple way to access servers, storage, databases and a set of application services

Gmail
- E-mail cloud service
- Pay for ONLY your emails stored (no infrastructure, etc.)

Dropbox
- Cloud Storage Service
- Originally built on AWS

Netflix
- Built on AWS
- Video on Demand

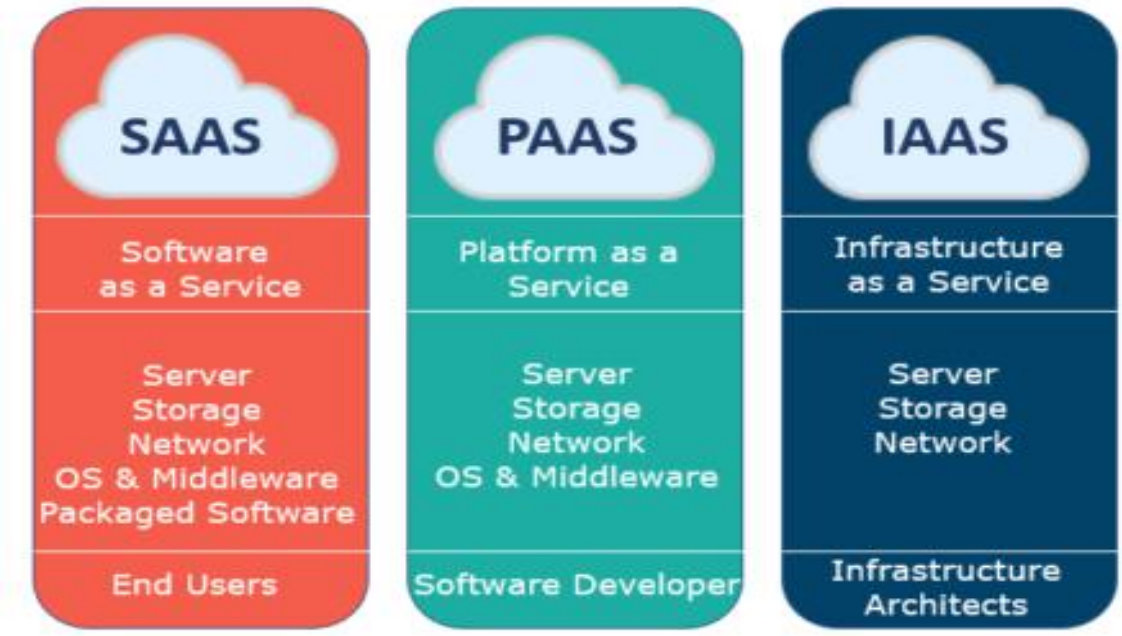
How does cloud computing work?

Cloud computing is possible because of a technology called virtualization. Virtualization allows for the creation of a simulated, digital-only "virtual" computer that behaves as if it were a physical computer with its own hardware. The technical term for such a computer is virtual machine. When properly implemented, virtual machines on the same host machine are sandboxed from one another, so they don't interact with each other at all, and the files and applications from one virtual machine aren't visible to the other virtual machines even though they're on the same physical machine.

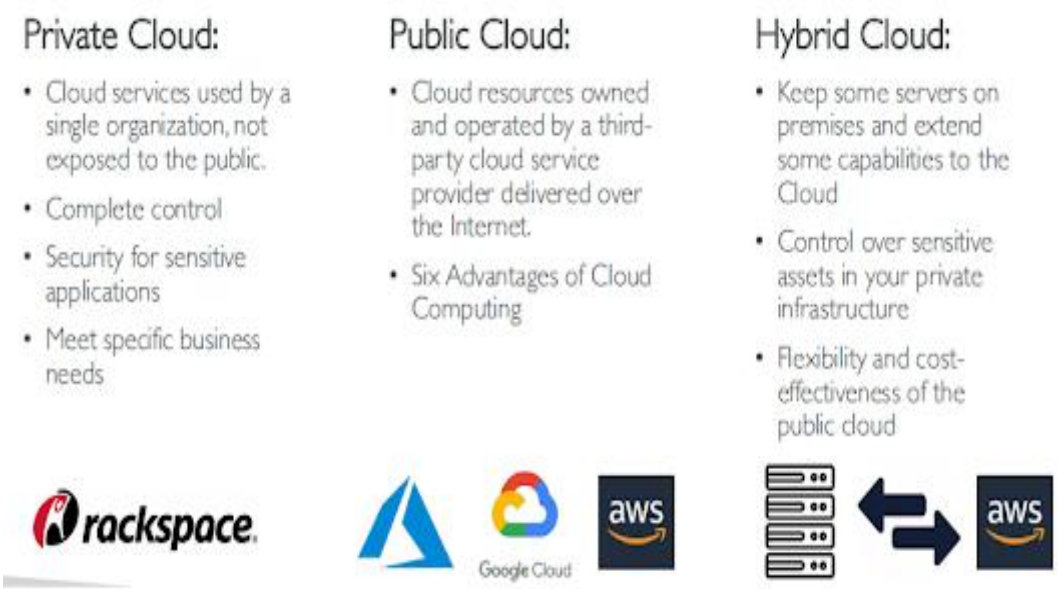
Virtual machines also make more efficient use of the hardware hosting them. By running many virtual machines at once, one server becomes many servers, and a data center becomes a whole host of data centers, able to serve many organizations. Thus, cloud providers can offer the use of their servers to far more customers at once than they would be able to otherwise, and they can do so at a low cost.

Even if individual servers go down, cloud servers in general should be always online and always available. Cloud vendors generally back up their services on multiple machines and across multiple regions. Users access cloud services either through a browser or through an app, connecting to the cloud over the Internet – that is, through many interconnected networks – regardless of what device they're using.

What are the main service models of cloud computing?



The deployment models of cloud computing:



Six Advantages of Cloud Computing

- **Trade capital expense (CAPEX) for operational expense (OPEX)**
- Pay On-Demand: don't own hardware
- Reduced Total Cost of Ownership (TCO) & Operational Expense (OPEX)
- **Benefit from massive economies of scale**
- Prices are reduced as AWS is more efficient due to large scale
- **Stop guessing capacity**
- Scale based on actual measured usage
- **Increase speed and agility**
- **Stop spending money running and maintaining data centers**
- **Go global in minutes:** leverage the AWS global infrastructure

Problems solved by the Cloud

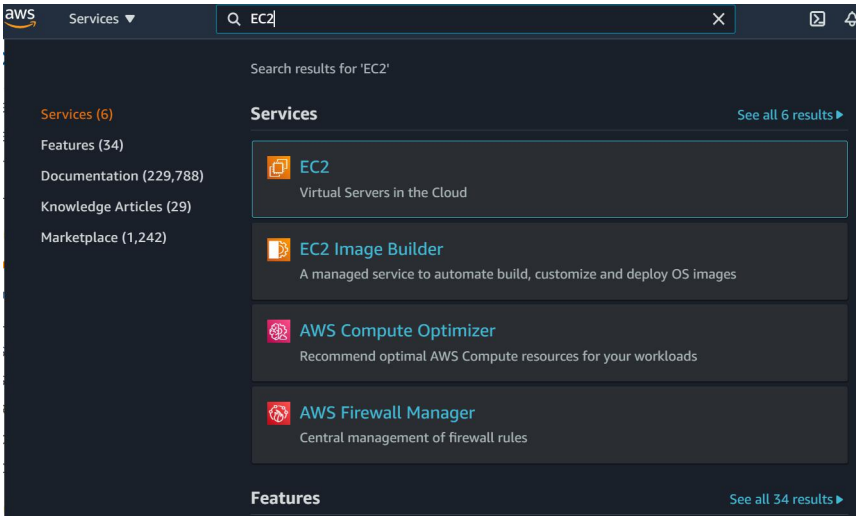
- **Flexibility:** change resource types when needed
- **Cost-Effectiveness:** pay as you go, for what you use
- **Scalability:** accommodate larger loads by making hardware stronger or adding additional nodes
- **Elasticity:** ability to scale out and scale-in when needed
- **High-availability and fault-tolerance:** build across data centers
- **Agility:** rapidly develop, test and launch software applications

AWS(Amazon web Services

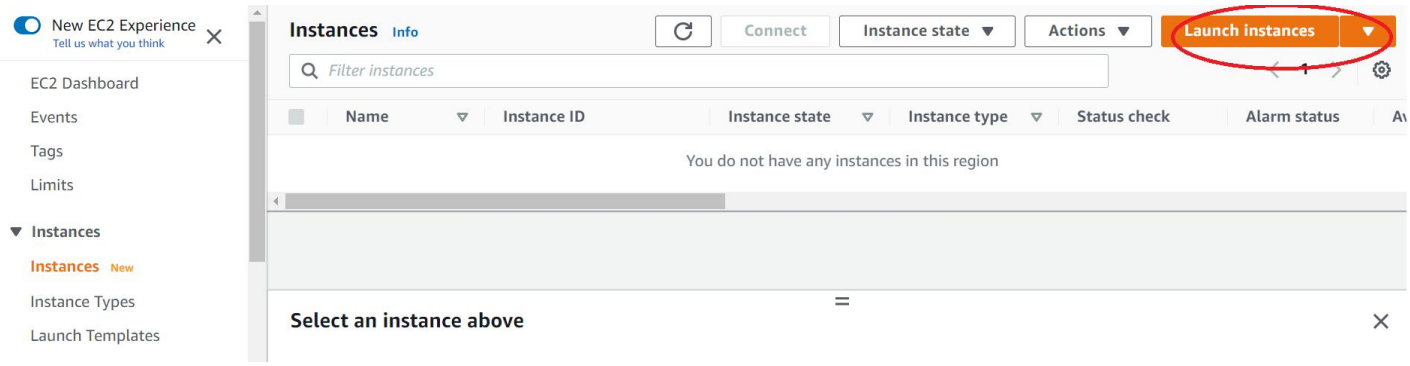
AWS-EC2

EC2 in action: Creating an EC2 instance with and without bootstrapping and attaching ENI's and playing with hibernate

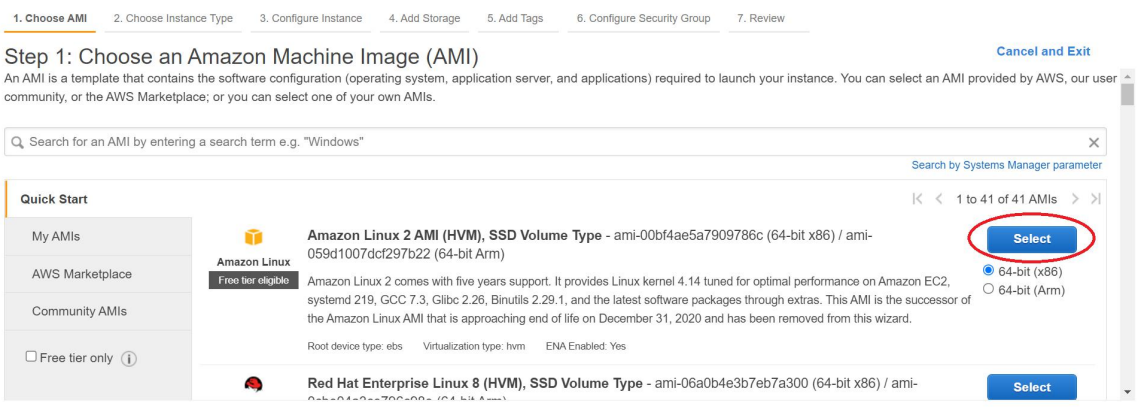
First go to the EC2 section



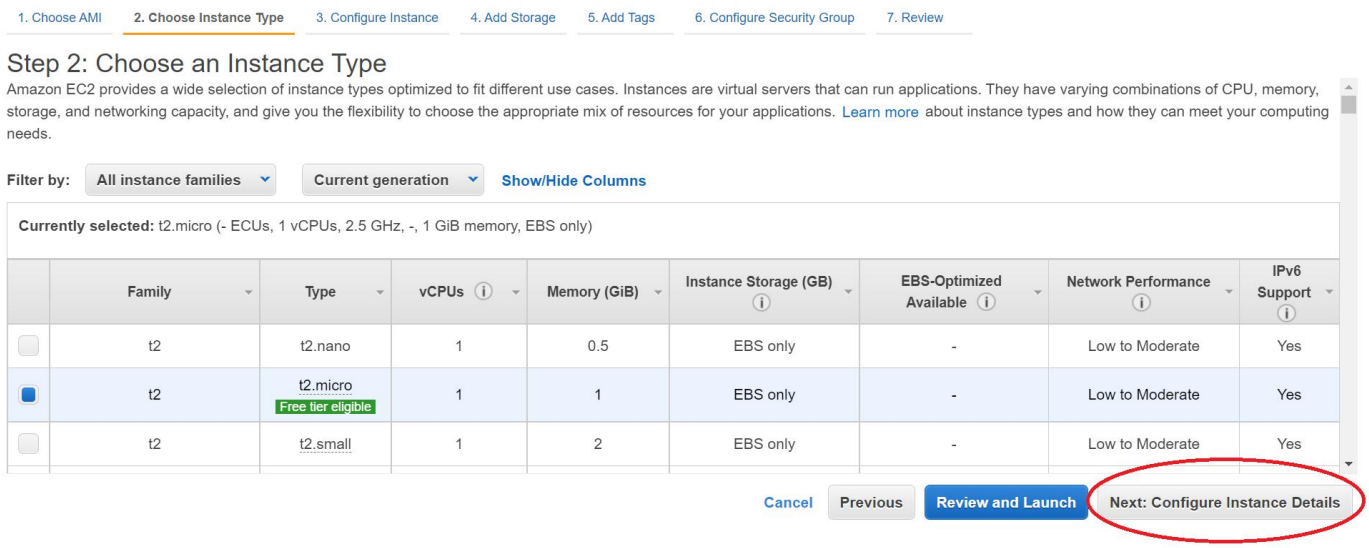
Then Click on Launch Instance



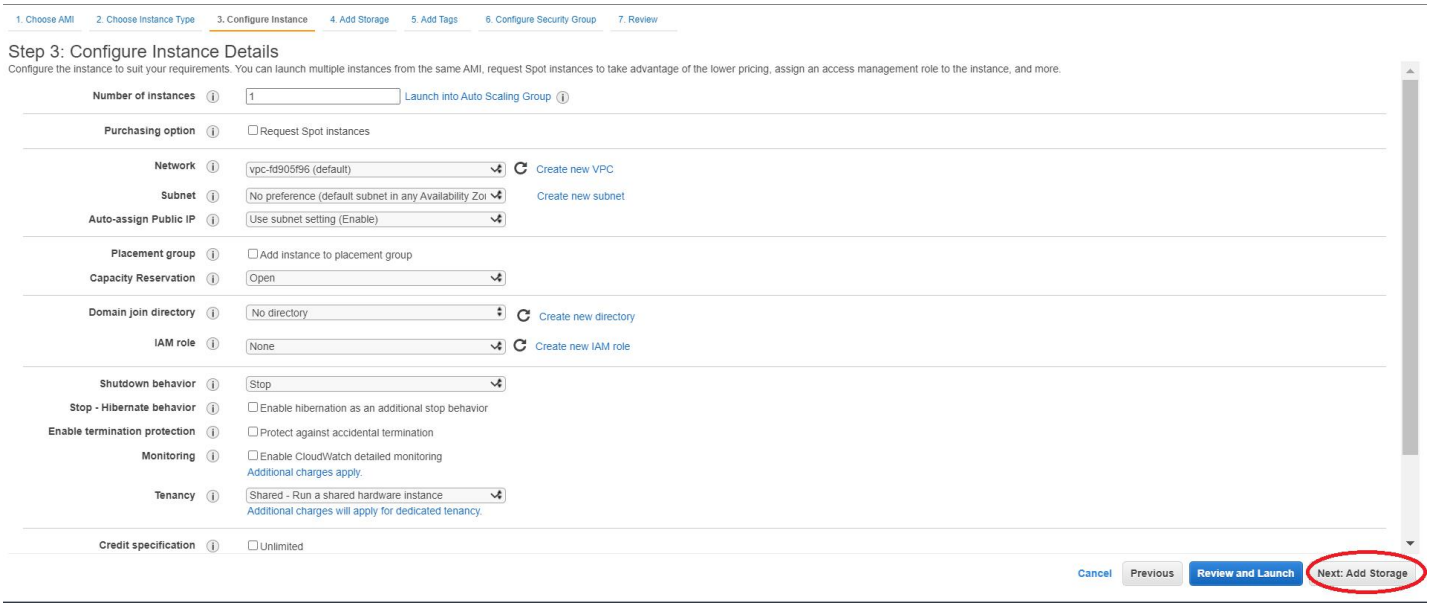
Select the AMI(Amazon Machine Image) that you want



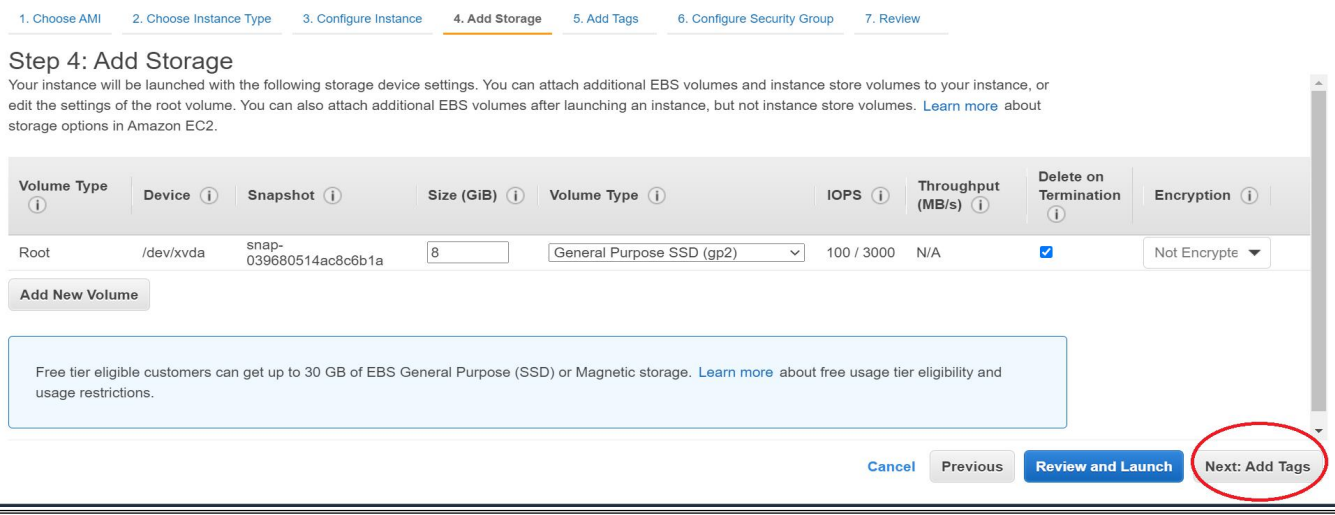
Select the Instance type or the machine that you want



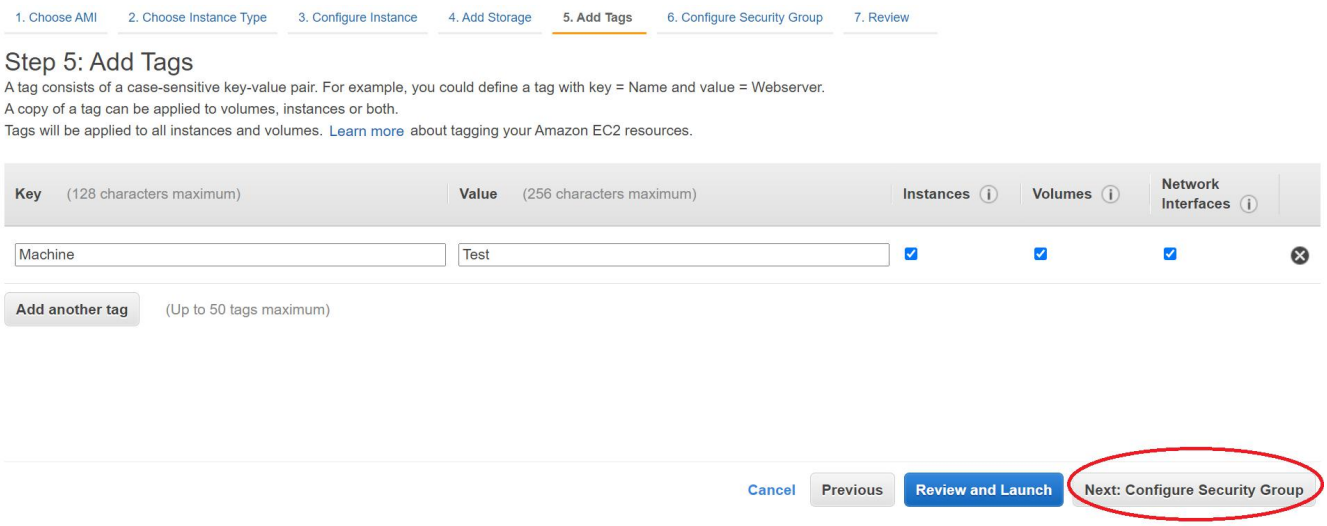
Then configure the instance details



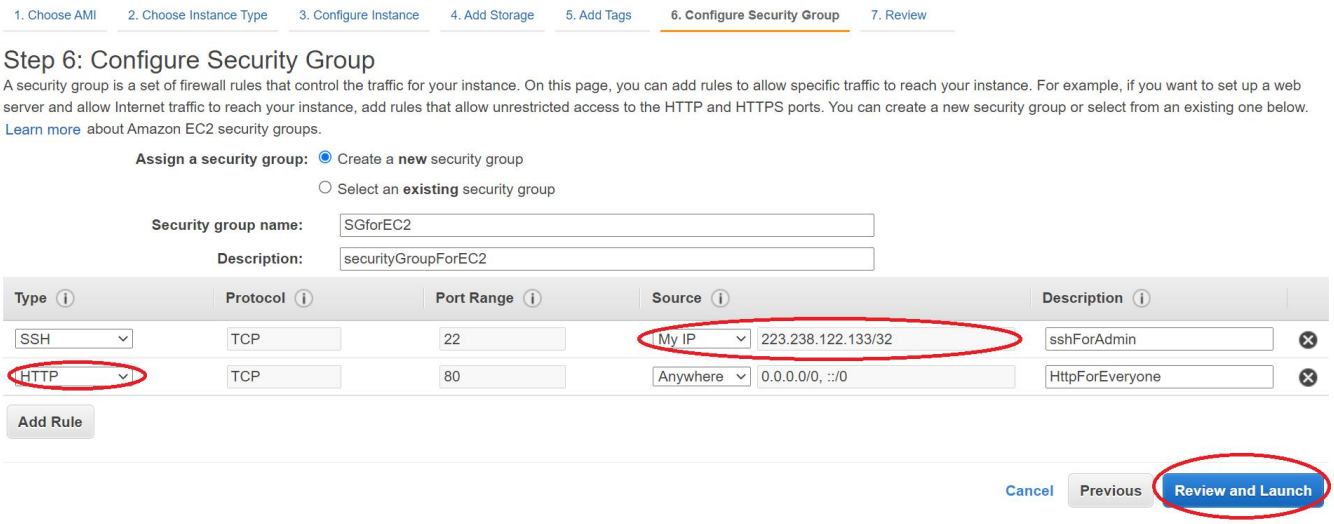
Select the storage details



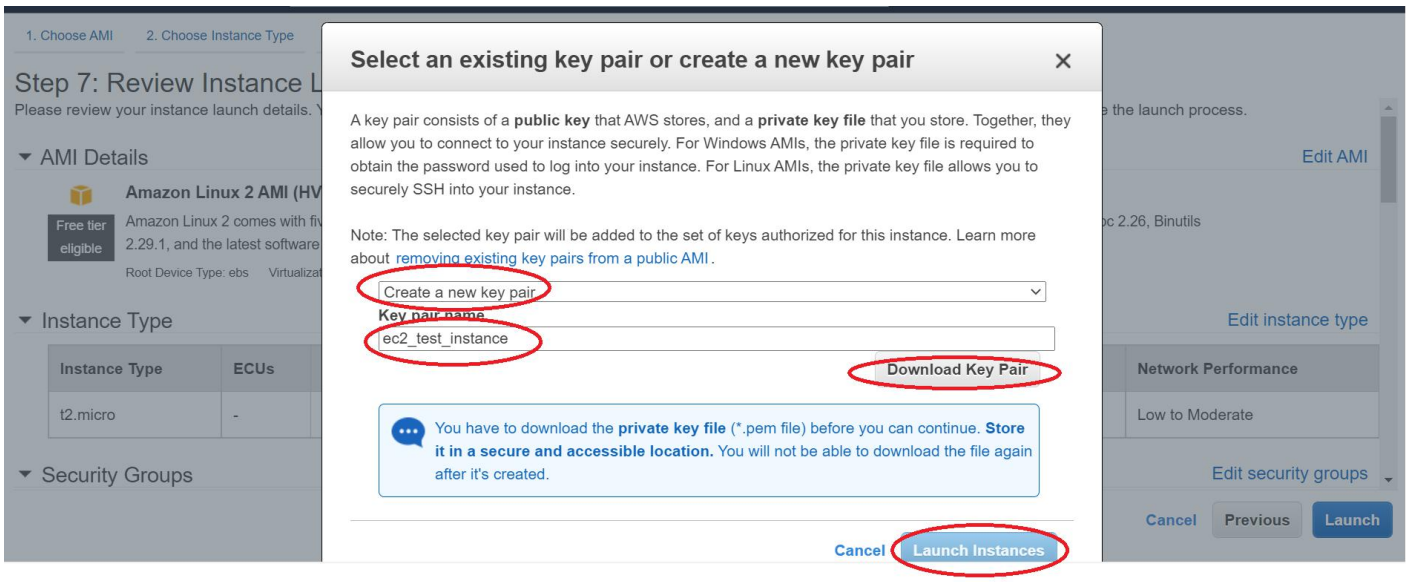
Add a tag to the EC2 machine but it is optional



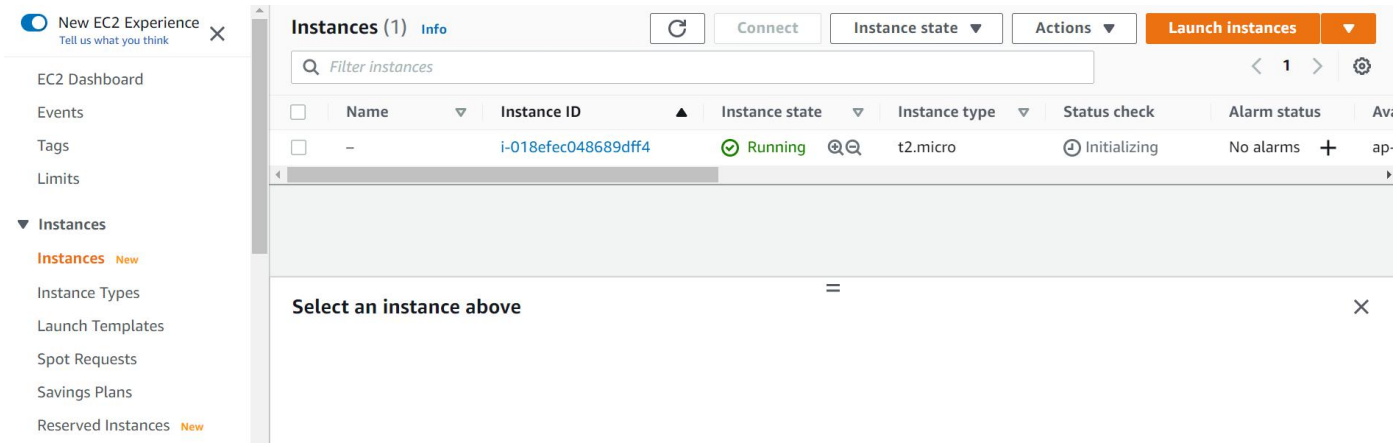
Now the most important part Is to configure the security groups,here we are setting an SSH from our own IP and HTTP from everywhere so everyone can access the machine through the IP but only we can SSH



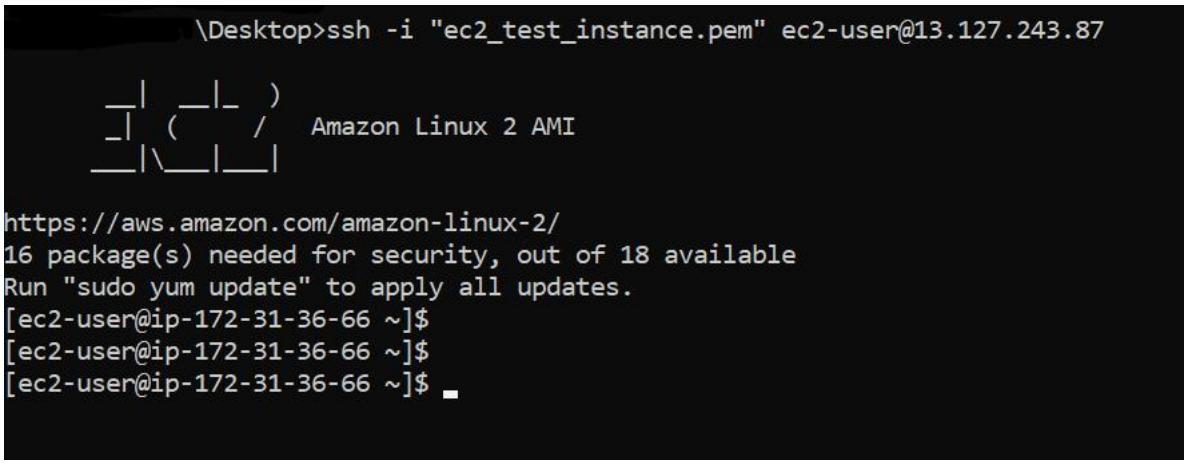
Now we set a key pair and download it for ssh into our instance from our terminal



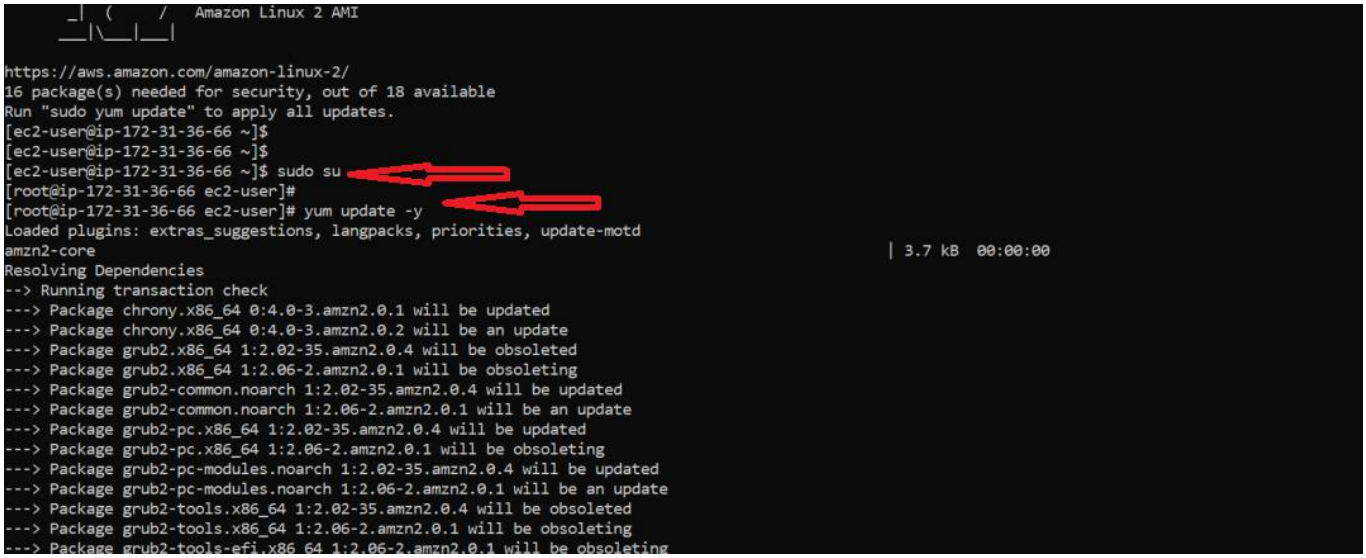
We can see the listed instance that we just created



Now to ssh from our local machine to the EC2 instance we got the terminal and enter the command as shown below with the user as ec2@user and at the public IP of the instance



Now we will launch a web server from our instance so first we install all the dependencies



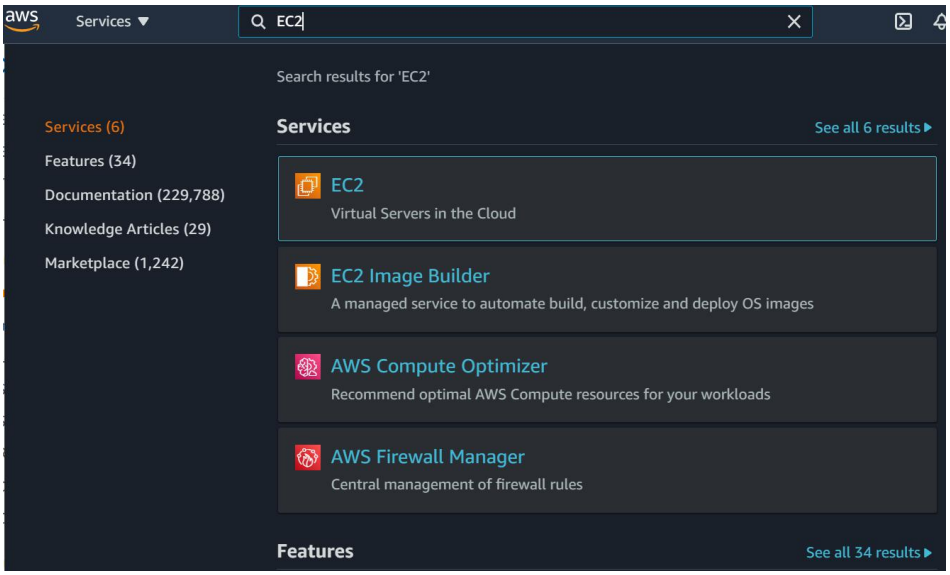
```
  python2-rpm.x86_64 0:4.11.3-40.amzn2.0.6          rpm.x86_64 0:4.11.3-40.amzn2.0.6                          rpm-build-libs.x86_64 0:4.11.3-4
  rpm-libs.x86_64 0:4.11.3-40.amzn2.0.6             rpm-plugin-systemd-inhibit.x86_64 0:4.11.3-40.amzn2.0.6

Replaced:
  grub2.x86_64 1:2.02-35.amzn2.0.4                                        grub2-tools.x86_64 1:2.02-35.amzn2.0.4

Complete!
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]# yum install -y httpd.x86_64
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Resolving Dependencies
--> Running transaction check
---> Package httpd.x86_64 0:2.4.48-2.amzn2 will be installed
--> Processing Dependency: httpd-tools = 2.4.48-2.amzn2 for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: httpd-filesystem = 2.4.48-2.amzn2 for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: system-logos-httpd for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: mod_http2 for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: httpd-filesystem for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: /etc/mime.types for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: libaprutil-1.so.0()(64bit) for package: httpd-2.4.48-2.amzn2.x86_64
--> Processing Dependency: libapr-1.so.0()(64bit) for package: httpd-2.4.48-2.amzn2.x86_64
--> Running transaction check
---> Package apr.x86_64 0:1.6.3-5.amzn2.0.2 will be installed
---> Package apr-util.x86_64 0:1.6.1-5.amzn2.0.2 will be installed
--> Processing Dependency: apr-util-bdb(x86-64) = 1.6.1-5.amzn2.0.2 for package: apr-util-1.6.1-5.amzn2.0.2.x86_64
---> Package generic-logos-httpd.noarch 0:18.0.0-4.amzn2 will be installed
---> Package httpd-filesystem.noarch 0:2.4.48-2.amzn2 will be installed
---> Package httpd-tools.x86_64 0:2.4.48-2.amzn2 will be installed
---> Package mailcap.noarch 0:2.1.41-2.amzn2 will be installed
---> Package mod_http2.x86_64 0:1.15.19-1.amzn2.0.1 will be installed
--> Running transaction check
---> Package apr-util-bdb.x86_64 0:1.6.1-5.amzn2.0.2 will be installed
--> Finished Dependency Resolution
```

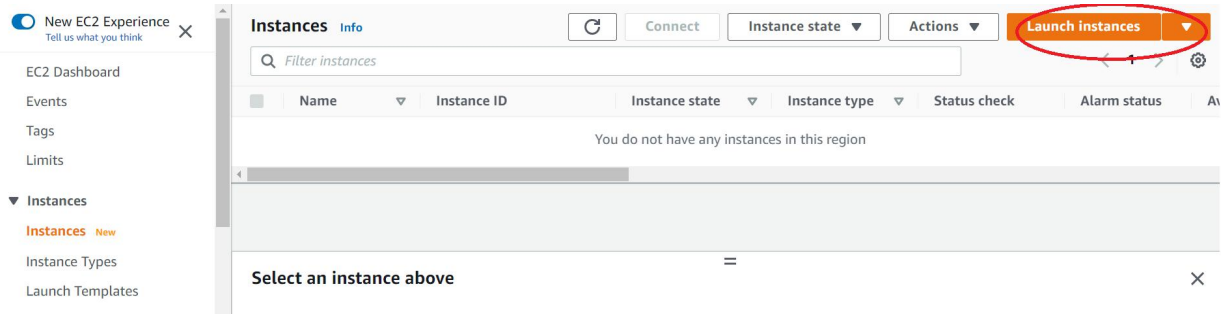
```
root@ip-172-31-36-66:/home/ec2-user
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]# systemctl start httpd.service
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]# systemctl enable httpd.service
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]# echo "Hello World from $(hostname -f)" > /var/www/html/index.html
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]#
[root@ip-172-31-36-66 ec2-user]#
```

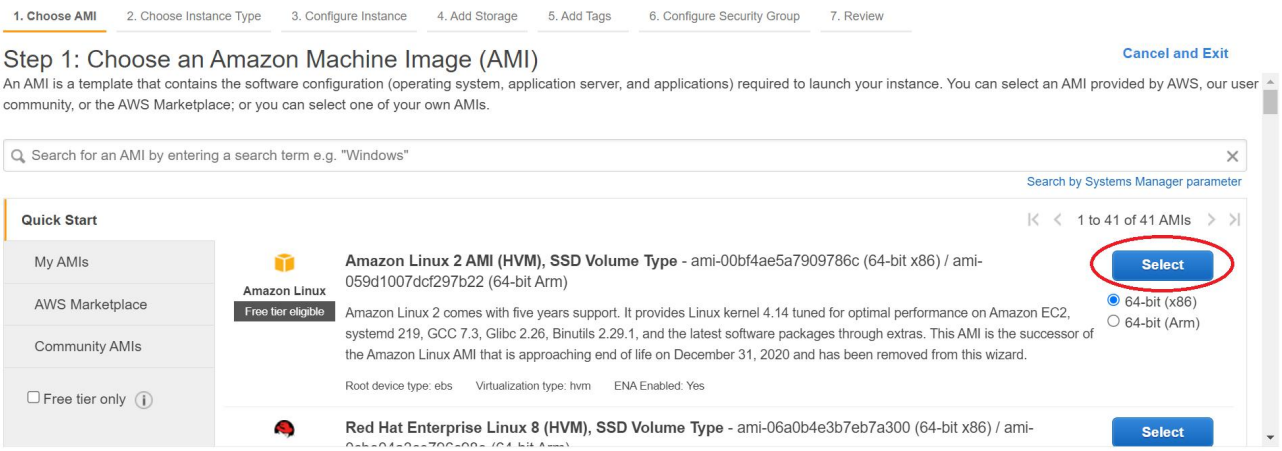
Now as we have installed and configured our web server we can view it on the HTTP rule from the browser

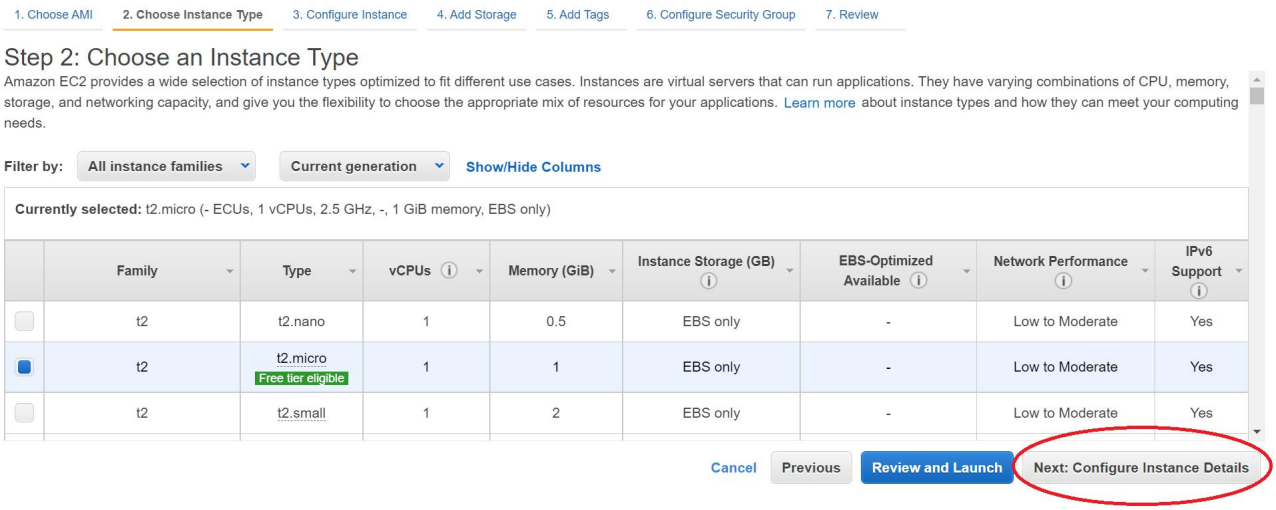
Hello World from ip-172-31-36-66.ap-south-1.compute.internal

Now accessing the same site using bootstrapping
Following the same steps from the previous topic:

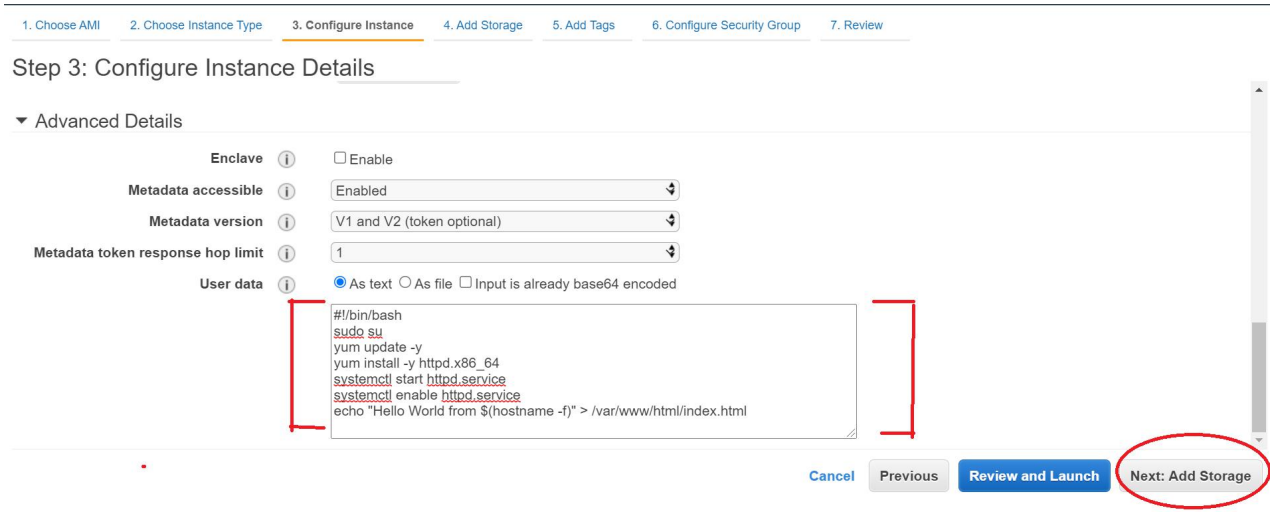




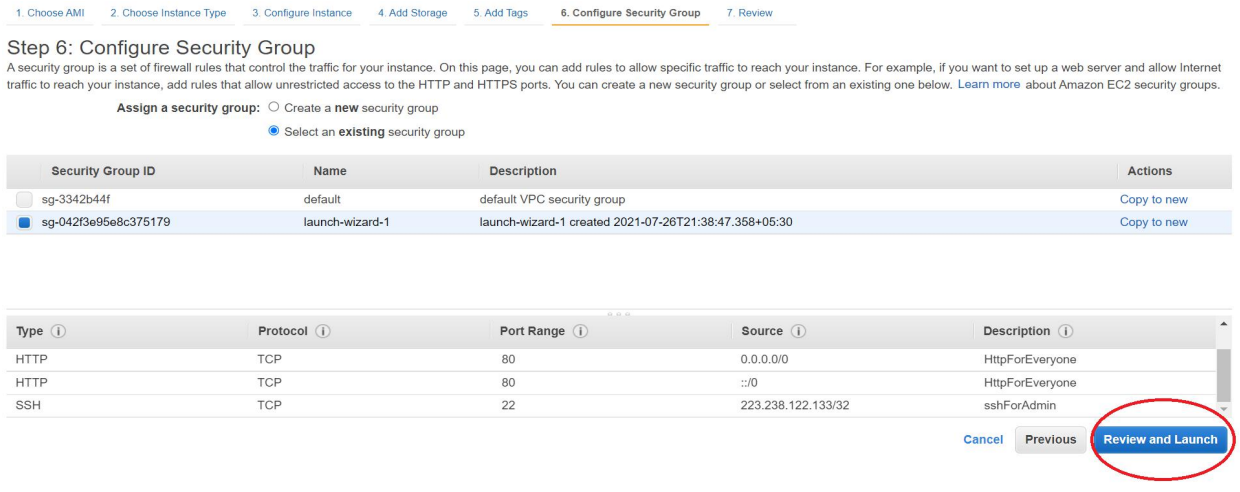




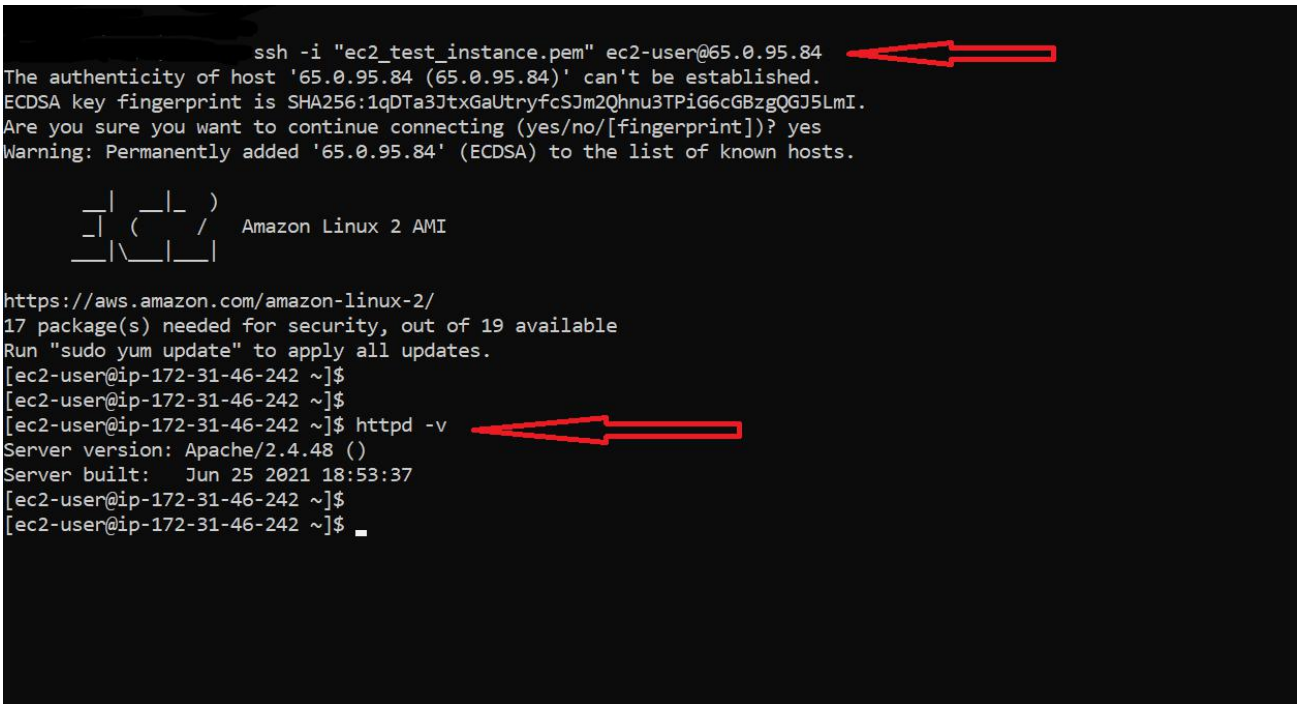
Now the main part where we are using the bootstrapping method or the user-data part here we will add all the commands that are to be executed when the machine boots p will all the commands in a new line



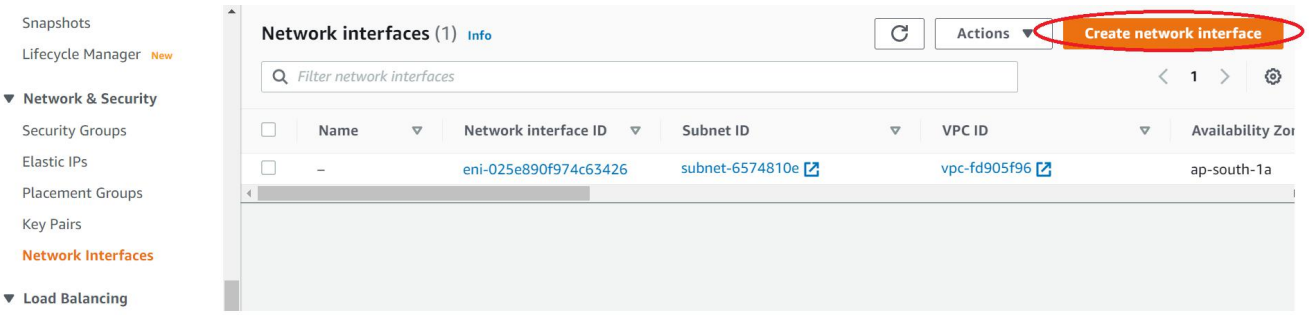
Using the same security group details as before



We can verify these when we ssh into our instance and hit **httpd -v** we can see that all our services have been installed already that was because of the bootstrapping that we used while launching the instance

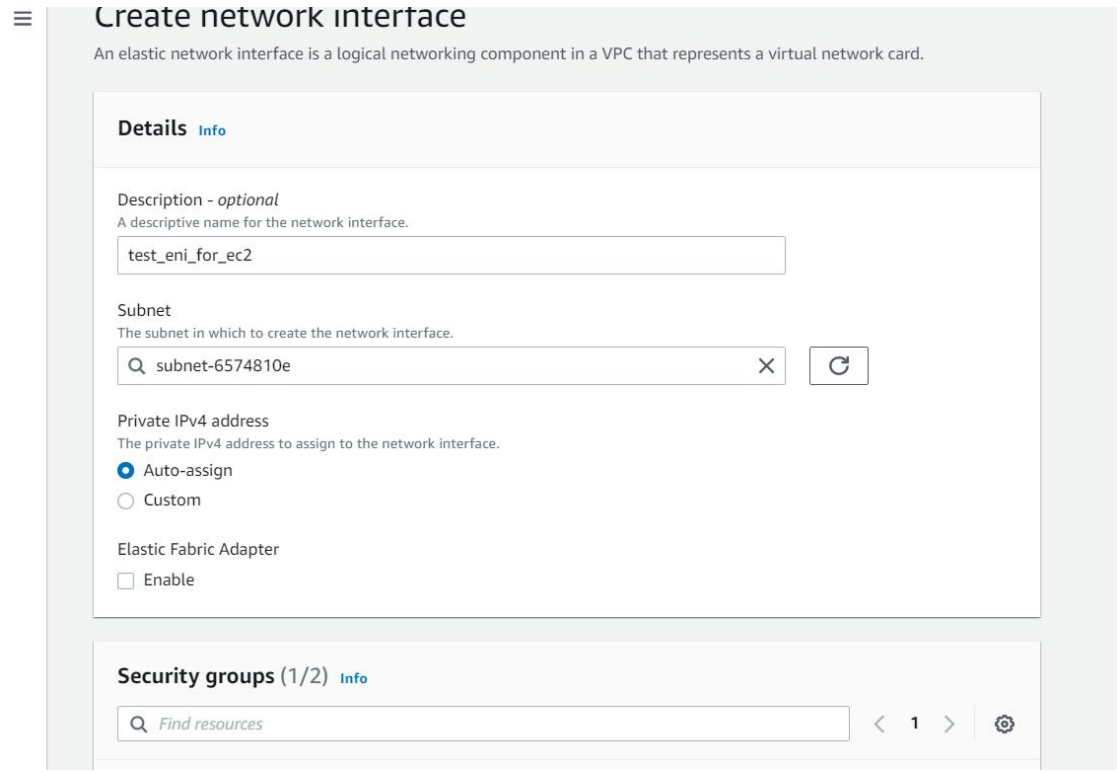


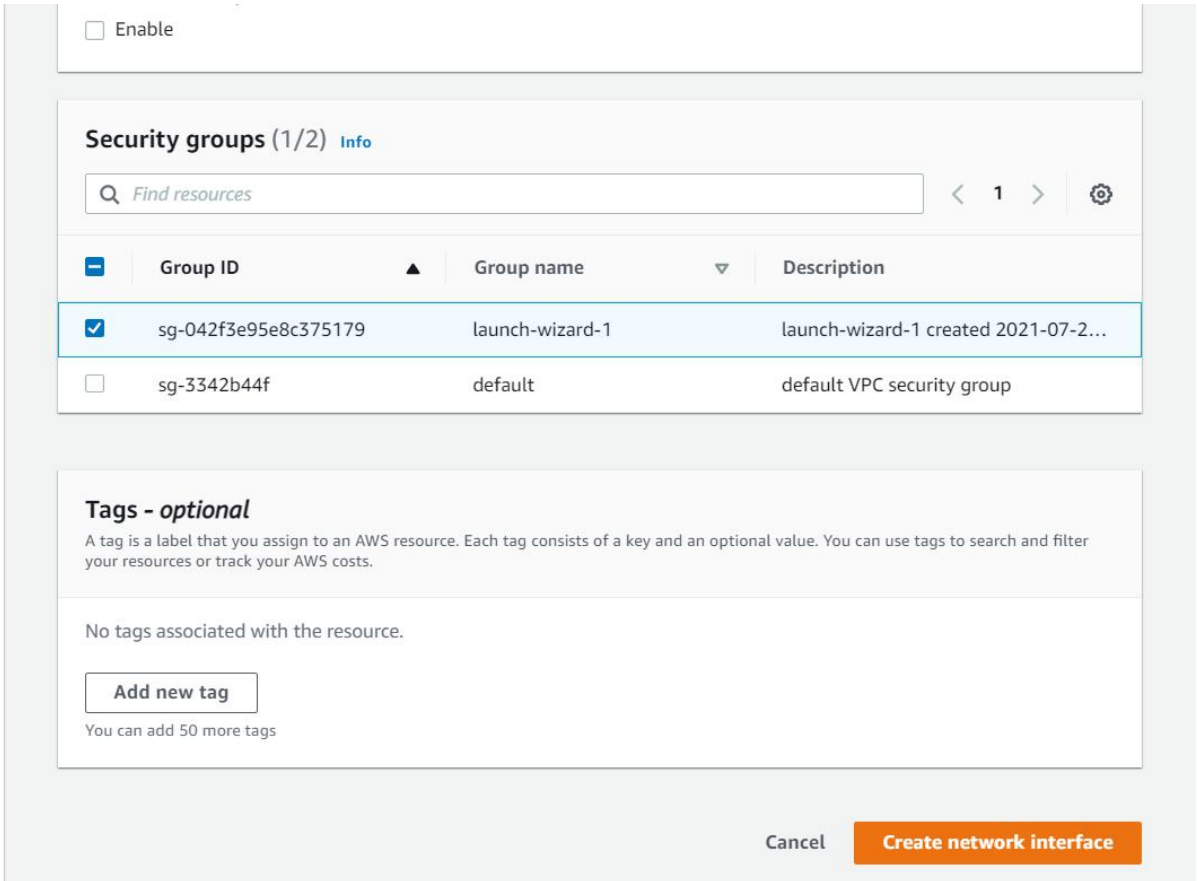
Now we will be playing with the ENI and we will be creating a new ENI and will be attaching it to our instance and verify it



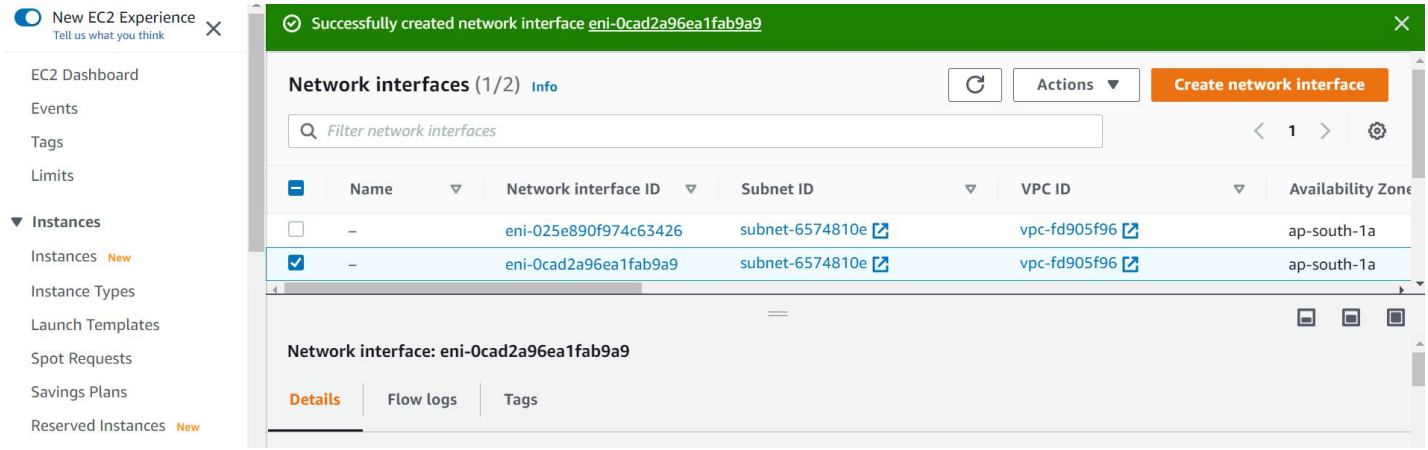
Now when we check the ENI attached to instance before we only see one ENI that is 172.31.46.242

Now to create an ENI we migrate into the ENI section from the EC2 dashboard and fill the details a our requirement

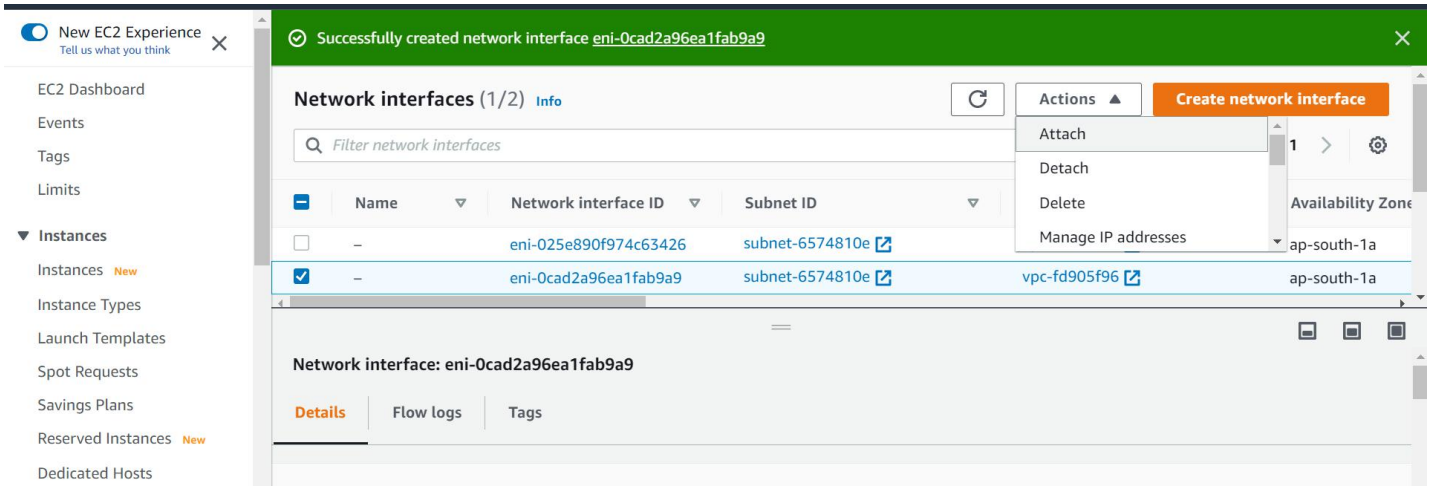




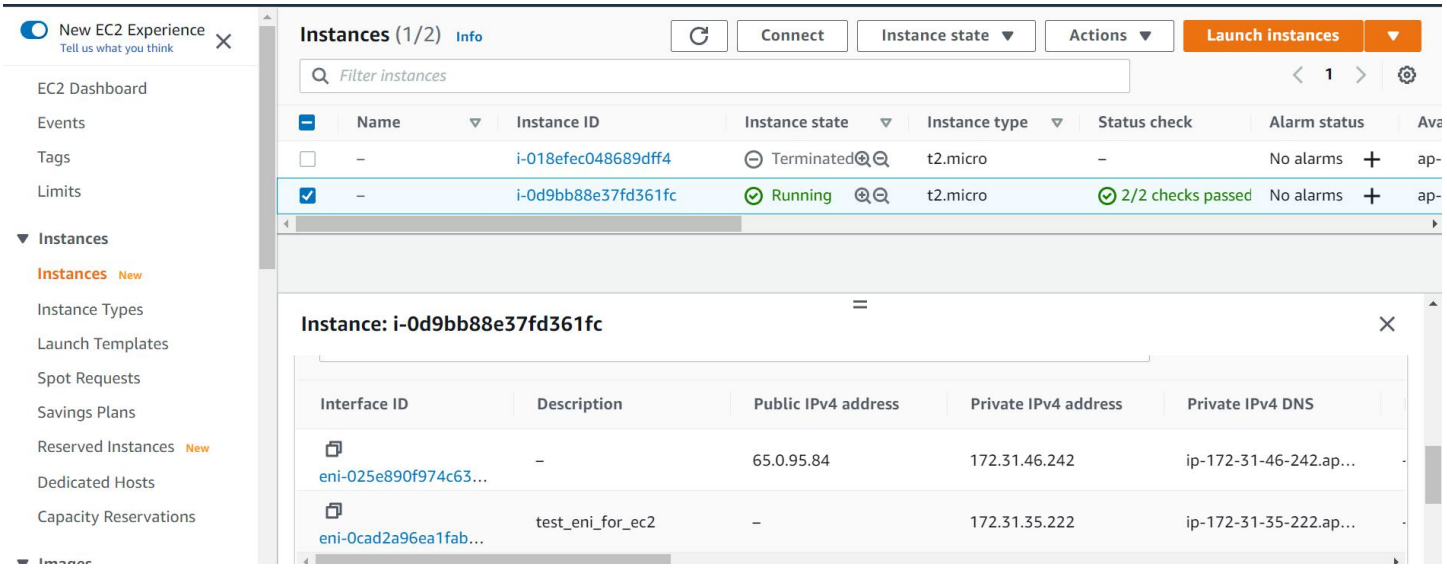
We can see that we have successfully created an ENI



Now to attach the ENI to the instance click the attach option and select the instance that we want to attach with



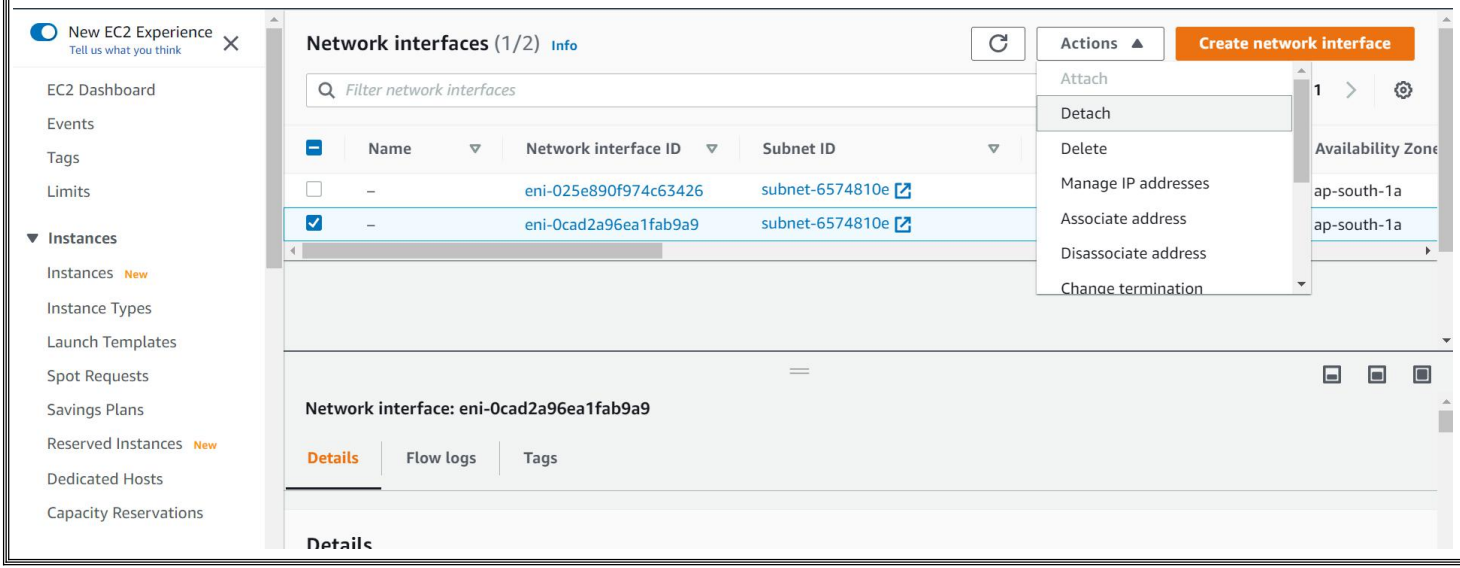
We can view the instance details and see the ENI that are attached to it



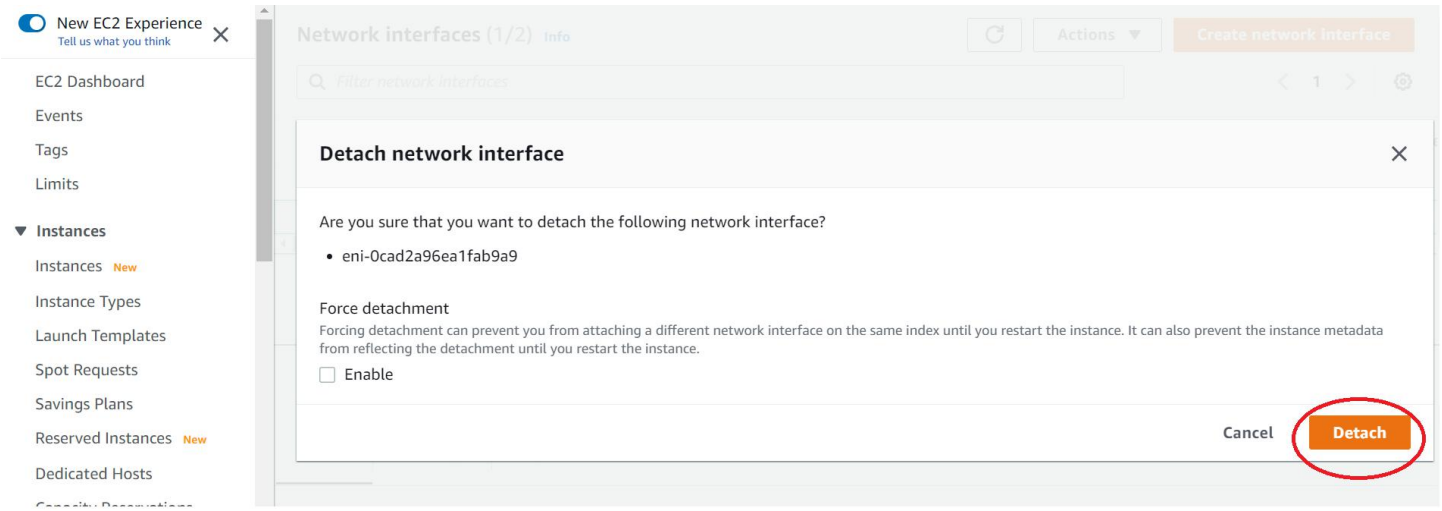
Now to verify it we can also see the ENI that we have attached to instance by ssh and hitting the command **ifconfig** to see the ENIs and we can see that there is a new ENI with IP 172.31.35.222

```
ec2-user@ip-172-31-46-242:~
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 32  bytes 2592 (2.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2592 (2.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ec2-user@ip-172-31-46-242 ~]$
[ec2-user@ip-172-31-46-242 ~]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.46.242  netmask 255.255.240.0  broadcast 172.31.47.255
        inet6 fe80::5a:84ff:fe61:cd8  prefixlen 64  scopeid 0x20<link>
        ether 02:5a:84:61:0c:d8  txqueuelen 1000  (Ethernet)
        RX packets 66988  bytes 96375673 (91.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5201  bytes 371332 (362.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.35.222  netmask 255.255.240.0  broadcast 172.31.47.255
        inet6 fe80::8a:7bff:fe43:de40  prefixlen 64  scopeid 0x20<link>
        ether 02:8a:7b:43:de:40  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1530 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 2008 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 32  bytes 2592 (2.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2592 (2.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ec2-user@ip-172-31-46-242 ~]$
[ec2-user@ip-172-31-46-242 ~]$
```

To detach the NEI we can hit the Detach option like this



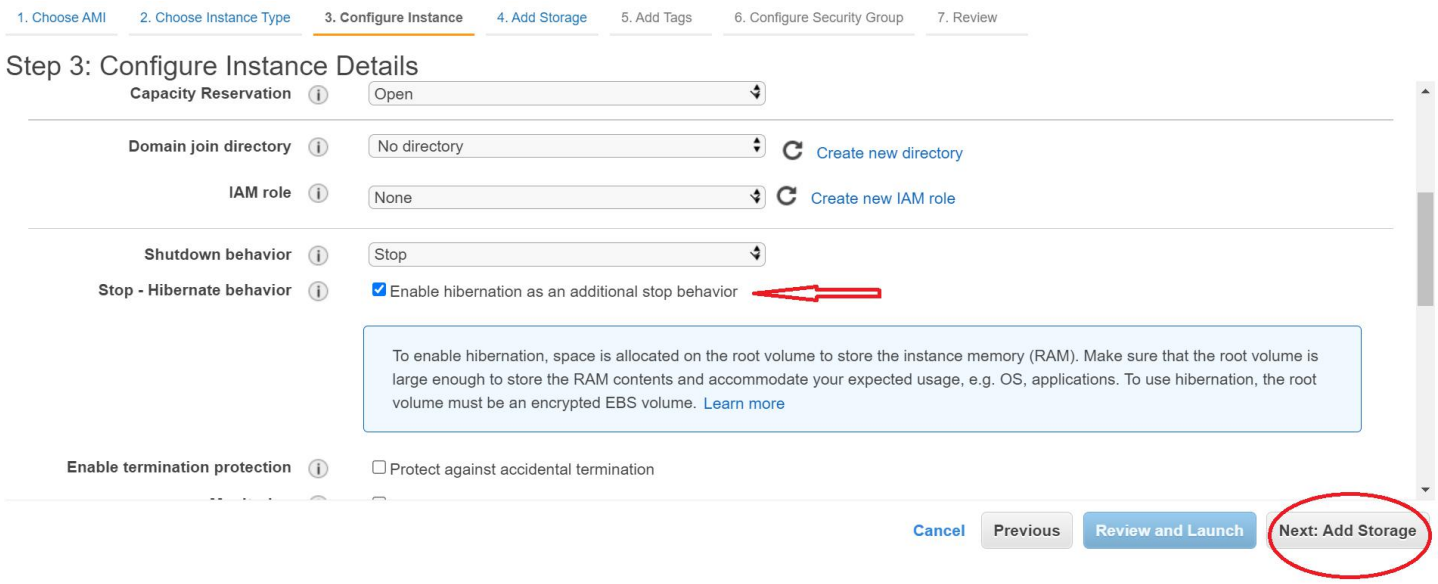
And confirm the detachment for the ENI



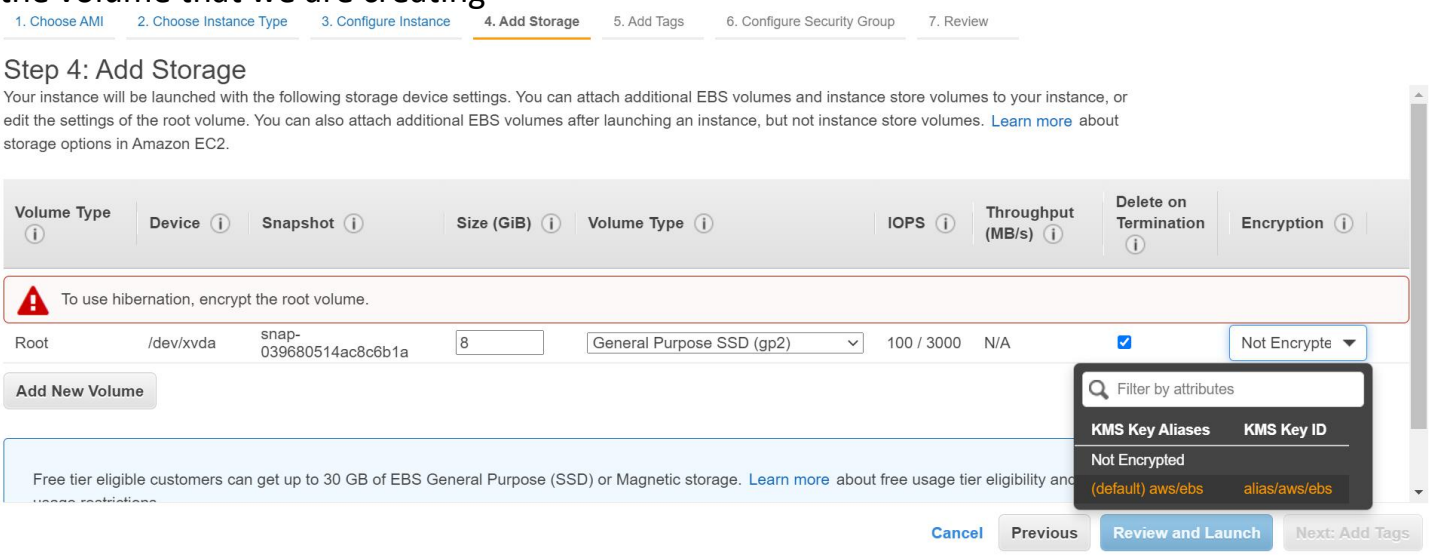
To verify it we can again check it through the command line

```
ec2-user@ip-172-31-46-242:~
        ether 02:8a:7b:43:de:40  txqueuelen 1000  (Ethernet)
        RX packets 13  bytes 1530 (1.4 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 2008 (1.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 32  bytes 2592 (2.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2592 (2.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ec2-user@ip-172-31-46-242 ~]$
[ec2-user@ip-172-31-46-242 ~]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.46.242  netmask 255.255.240.0  broadcast 172.31.47.255
        inet6 fe80::5a:84ff:fe61:cd8  prefixlen 64  scopeid 0x20<link>
        ether 02:5a:84:61:0c:d8  txqueuelen 1000  (Ethernet)
        RX packets 67030  bytes 96379994 (91.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5250  bytes 378324 (369.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 32  bytes 2592 (2.5 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 32  bytes 2592 (2.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ec2-user@ip-172-31-46-242 ~]$
[ec2-user@ip-172-31-46-242 ~]$
[ec2-user@ip-172-31-46-242 ~]$
```

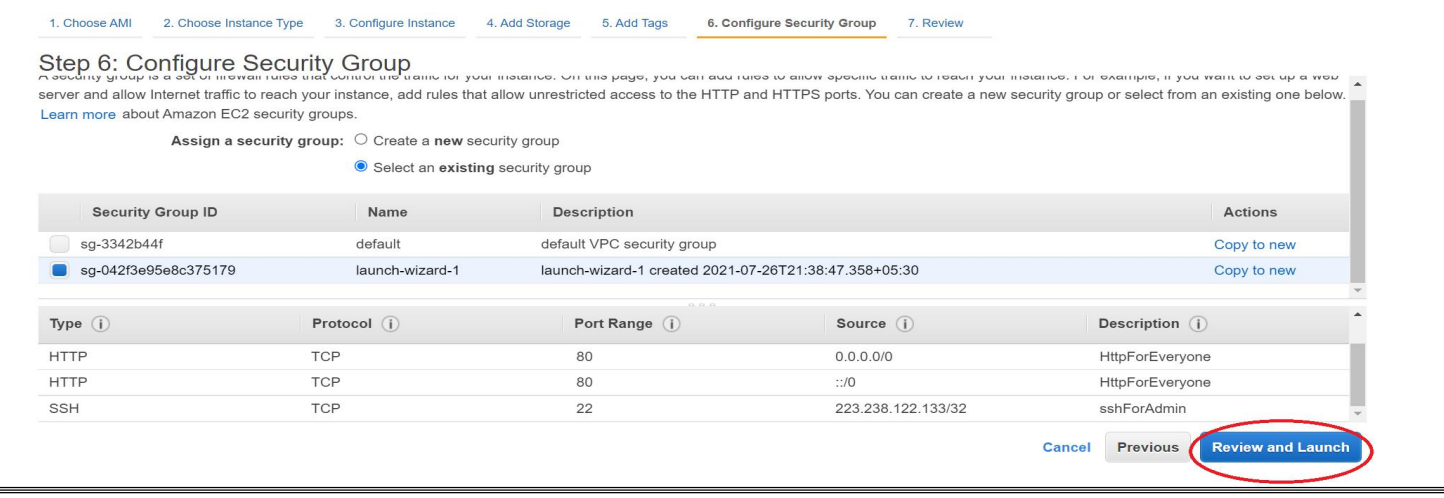
Now we will playing with the hibernate feature of the EC2 instance,so we will keep all the setting same but in the instance details part we will hit/check the checkbox of Stop-hibernate behaviour for hibernate to work



We set up the storage details as hibernate uses encrypted EBS volumes so we have to encrypt the volume that we are creating



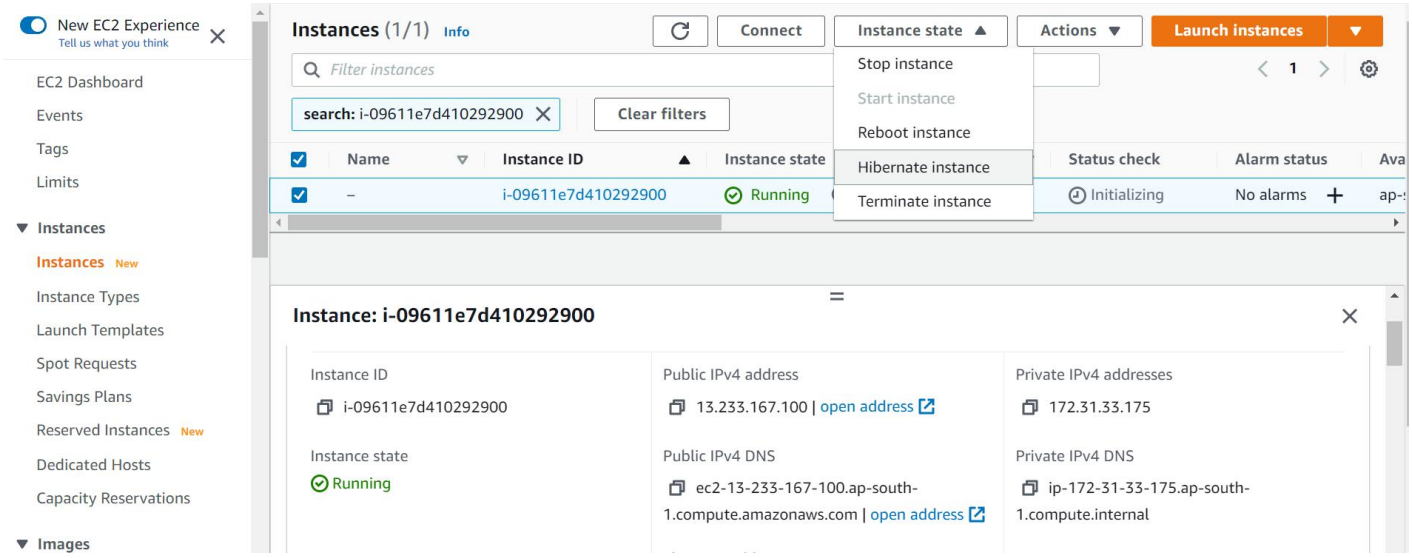
Using the same security group



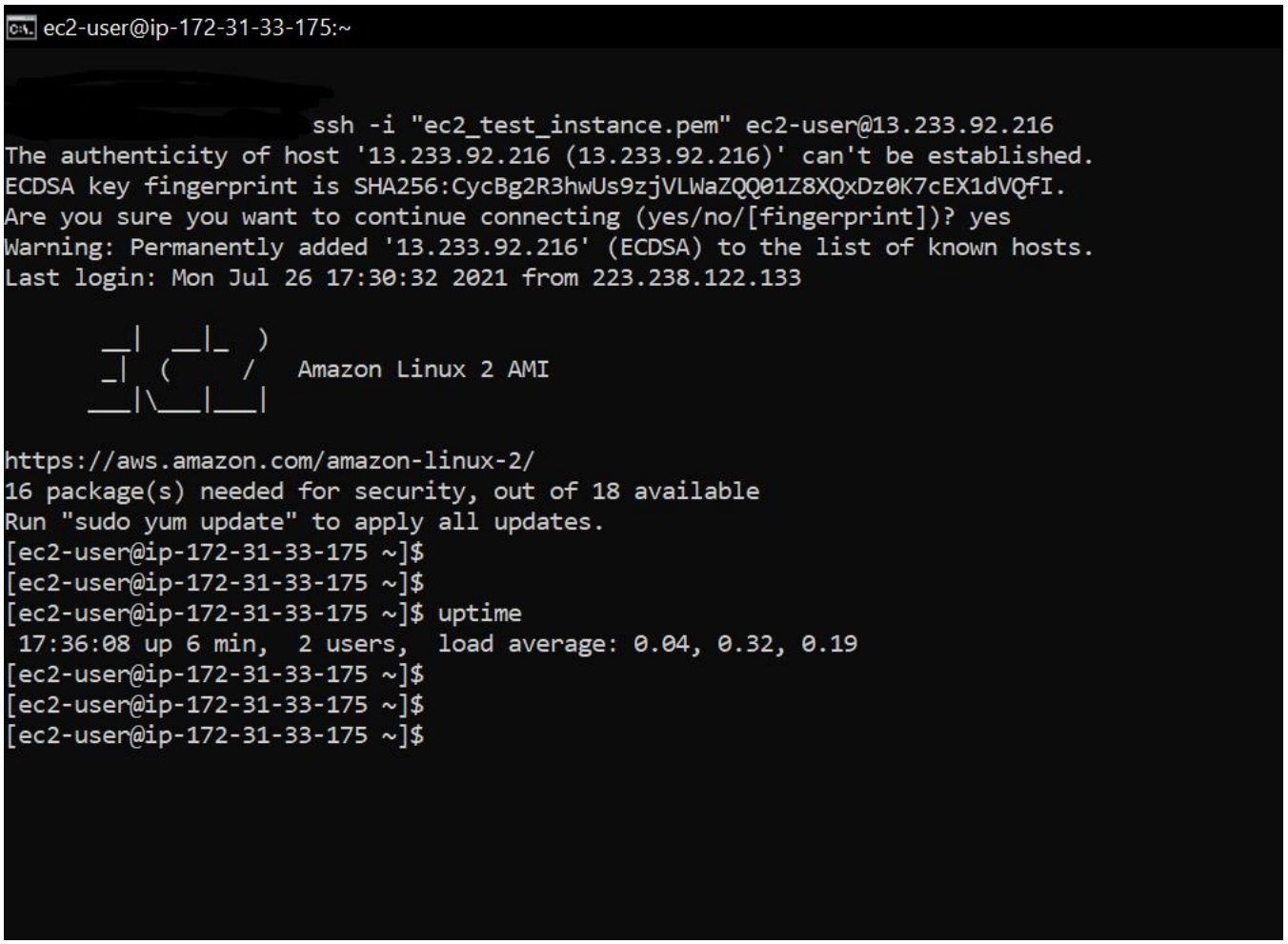
Now we hit some commands to acquire some data so that we can see that when we use hibernate e we can again regain the data of the instance

```
ec2-user@ip-172-31-33-175:~

                    ssh -i "ec2_test_instance.pem" ec2-user@13.233.167.100
The authenticity of host '13.233.167.100 (13.233.167.100)' can't be established.
ECDSA key fingerprint is SHA256:CycBg2R3hwUs9zjVLWaZQQ01Z8XQxDz0K7cEX1dVQfI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '13.233.167.100' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
16 package(s) needed for security, out of 18 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$ uptime
 17:30:37 up 1 min,  1 user,  load average: 1.67, 0.70, 0.25
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$ uptime
 17:30:40 up 1 min,  1 user,  load average: 1.67, 0.70, 0.25
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$ uptime
 17:30:52 up 1 min,  1 user,  load average: 1.30, 0.66, 0.25
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$ uptime
 17:31:04 up 1 min,  1 user,  load average: 1.10, 0.64, 0.25
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$ uptime
 17:32:14 up 2 min,  1 user,  load average: 0.34, 0.50, 0.23
[ec2-user@ip-172-31-33-175 ~]$
[ec2-user@ip-172-31-33-175 ~]$
```

Now we hit the hibernate option to hibernate the instance



Now again when we again start the instance and SSH into the instance and check the uptime and we can get our data back or we can say that the RAM is regained

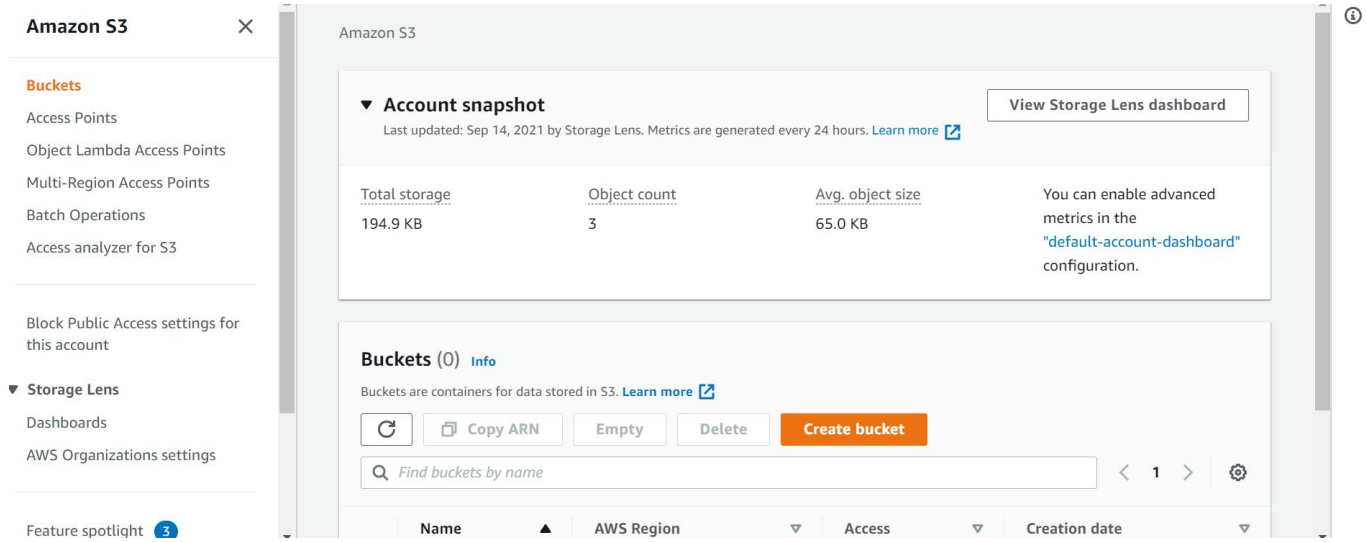


AWS-S3 & website hosting:

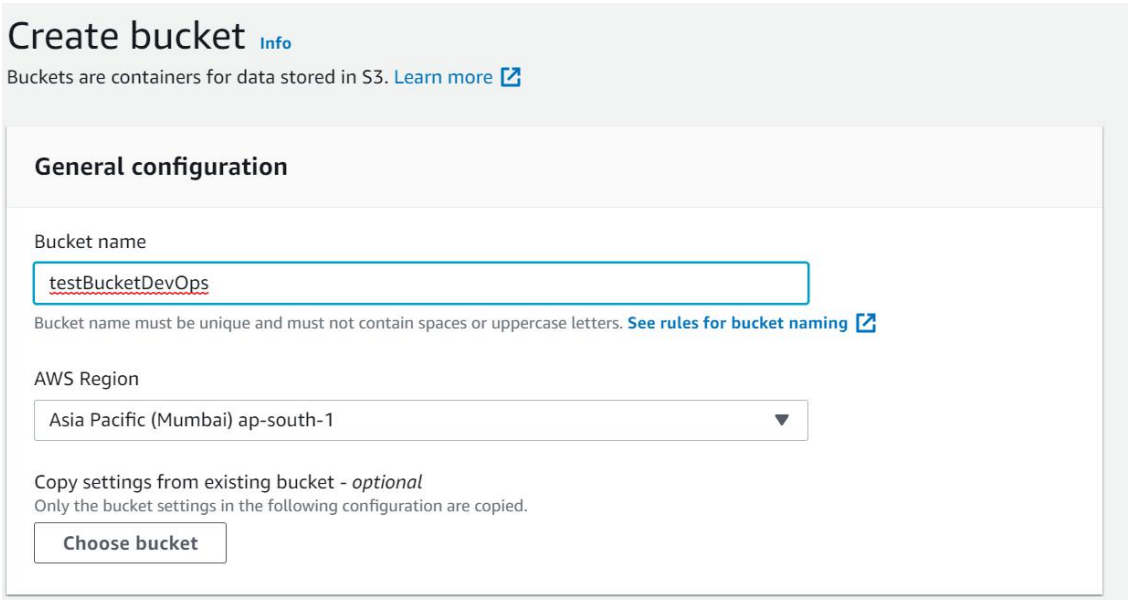
AWS-S3:

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. This means customers of all sizes and industries can use it to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides easy-to-use management features so you can organize your data and configure finely-tuned access controls to meet your specific business, organizational, and compliance requirements. Amazon S3 is designed for 99.999999999% (11 9's) of durability, and stores data for millions of applications for companies all around the world.

Creating a S3 bucket:



Choosing a name for our bucket that should be universally unique



Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. **Learn more**

- [x] **Block *all* public access**
  Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.
  - [x] **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
    S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
  - [x] **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
    S3 will ignore all ACLs that grant public access to buckets and objects.
  - [x] **Block public access to buckets and objects granted through *new* public bucket or access point policies**
    S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
  - [x] **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
    S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

Bucket Versioning

Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your Amazon S3 bucket. With versioning, you can easily recover from both unintended user actions and application failures. **Learn more**

Bucket Versioning

- (•) Disable
- ( ) Enable

Tags (0) - *optional*

Track storage cost or other criteria by tagging your bucket. **Learn more**

No tags associated with this bucket.

Add tag

Default encryption

Automatically encrypt new objects stored in this bucket. **Learn more**

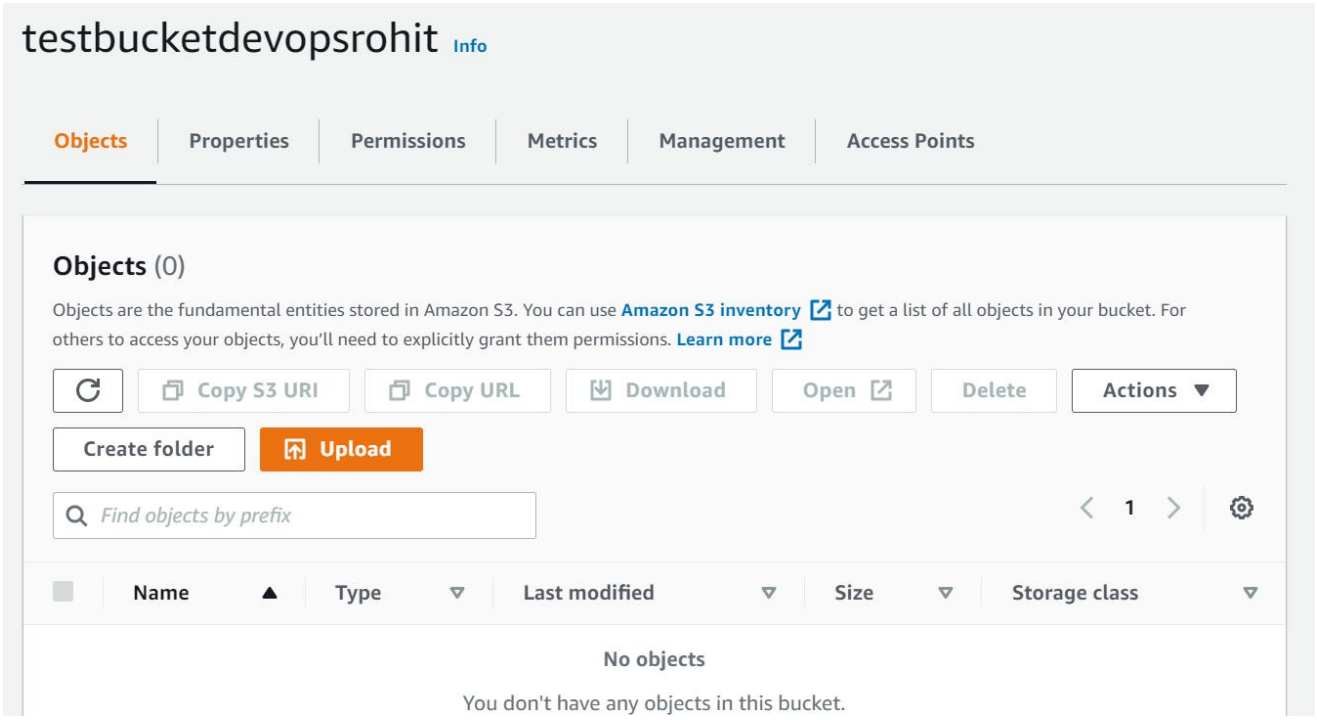
Server-side encryption

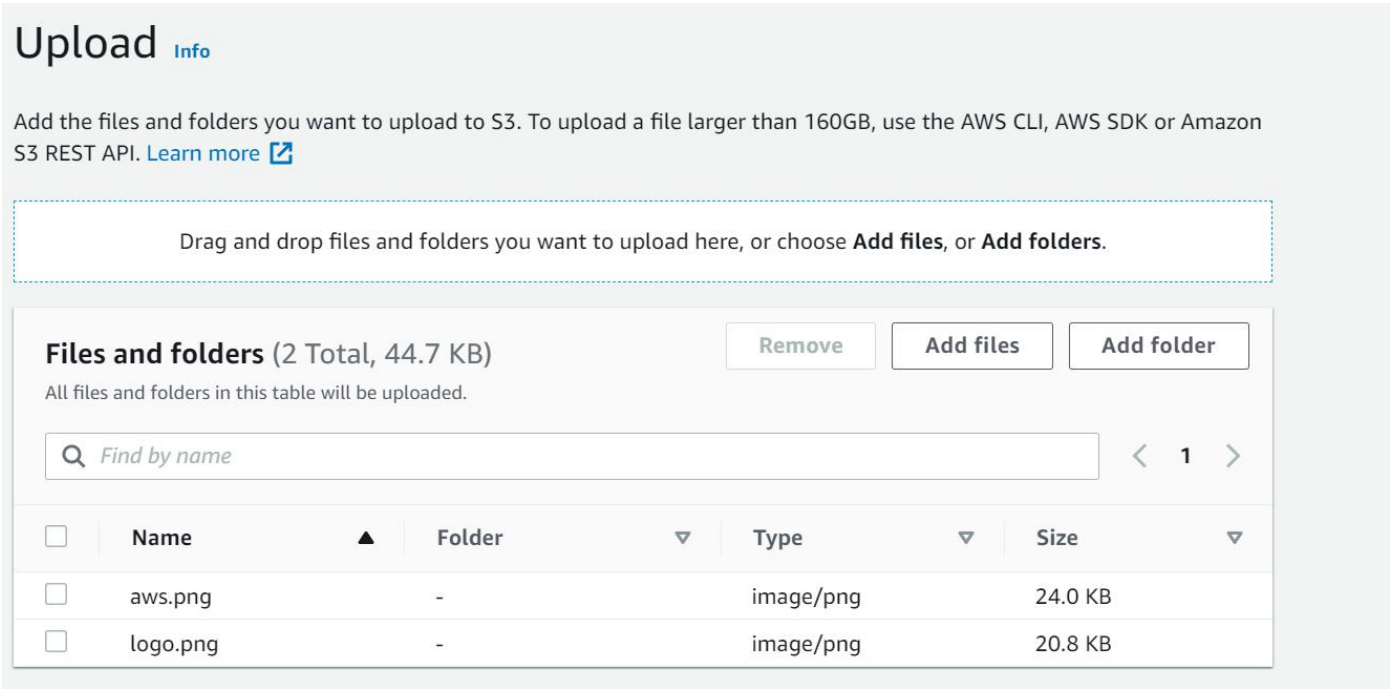
- (•) Disable
- ( ) Enable

▸ **Advanced settings**

ⓘ After creating the bucket you can upload files and folders to the bucket, and configure additional bucket settings.

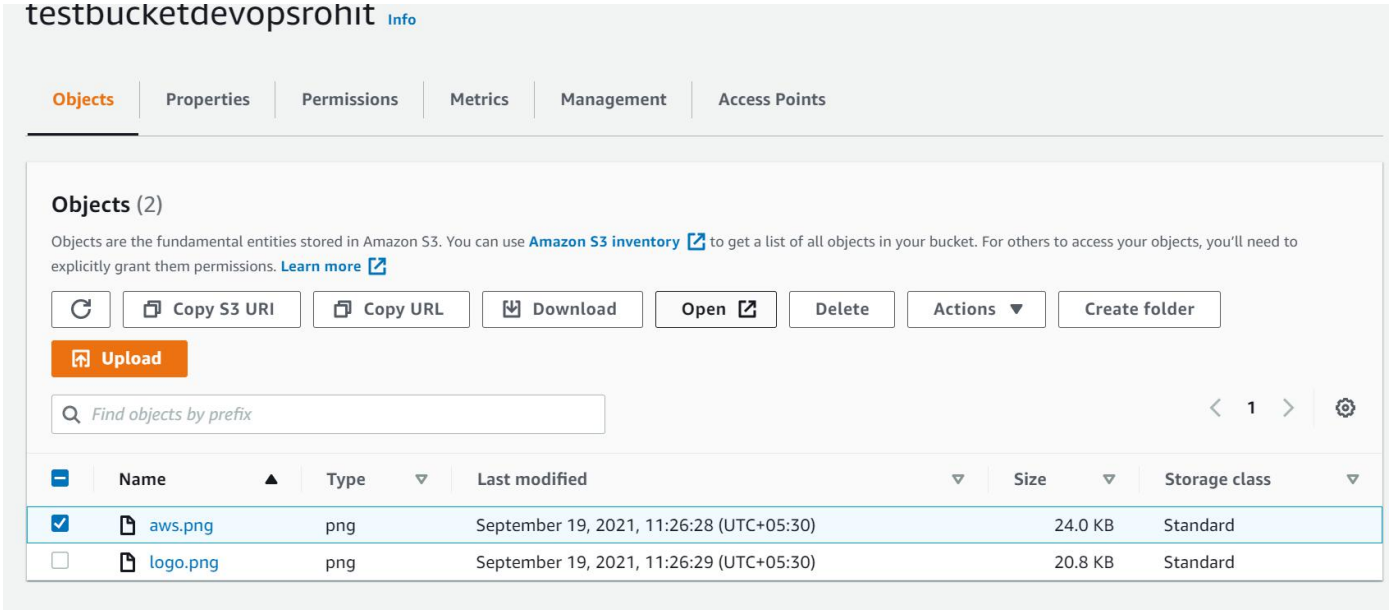
We can see that the bucket has been created jnow and we can upload files in it





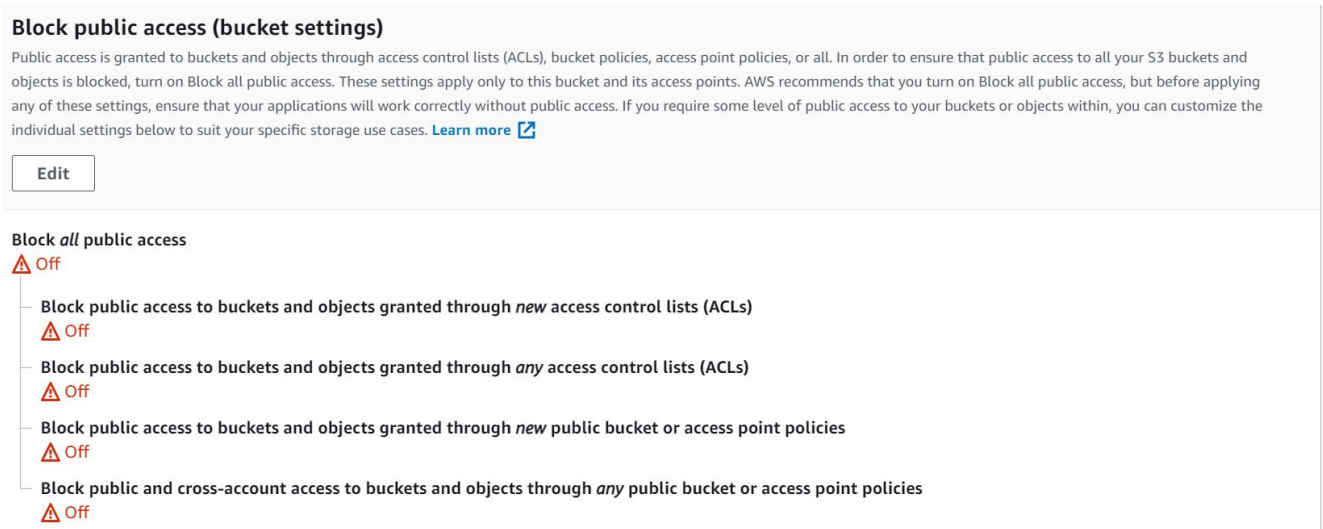
Then click on upload and now we can see our files in the bucket we can also enable versioning to see the versions of the files we have changed and re-uploaded also this helps in deleting the files which adds a delete marker on the files first then again if we delete the files by confirming with permanently delete the object is deleted permanently.

We can click/select the object and click on open button to preview the object on the browser



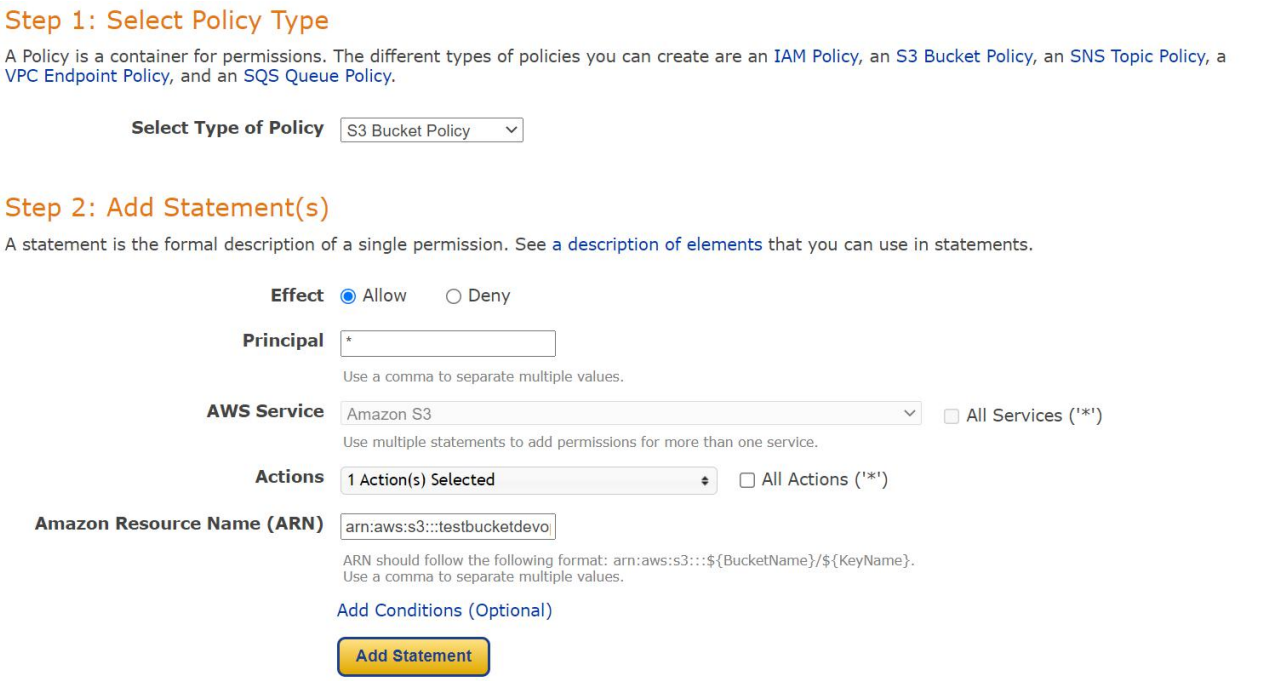
Now hosting a static website on S3

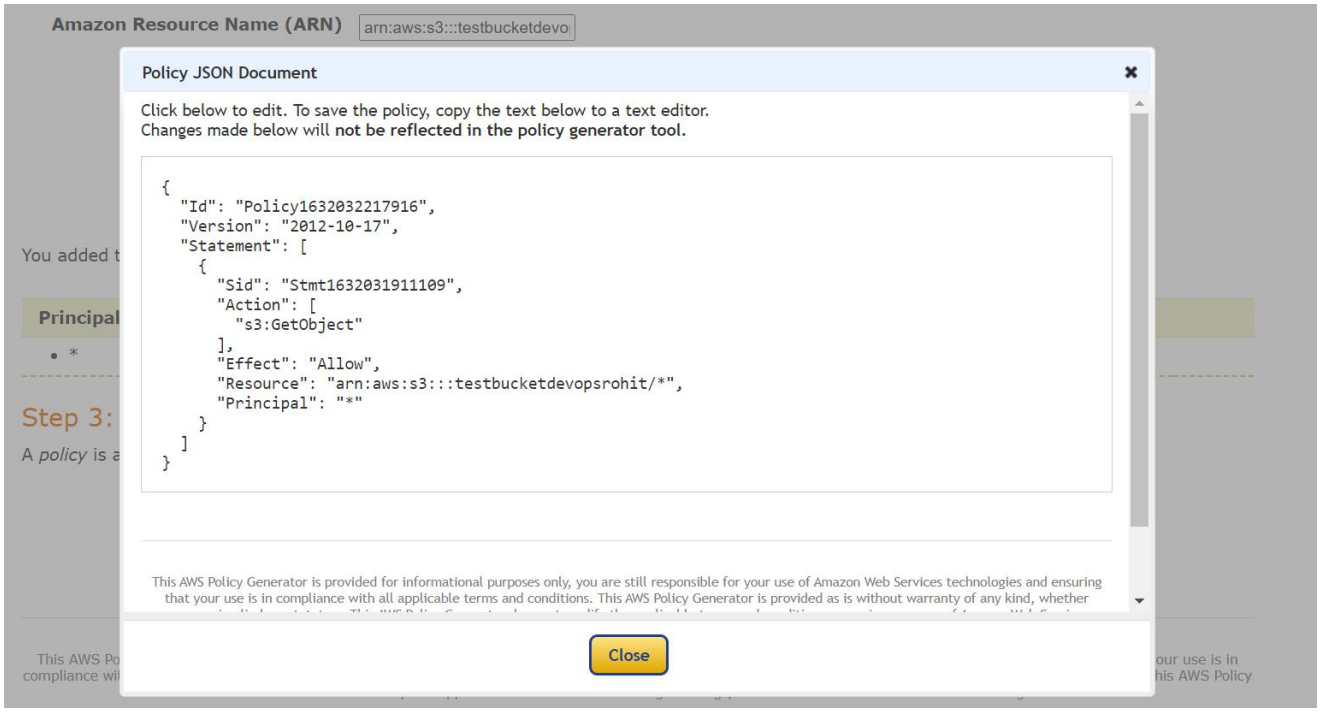
First have to allow all public access so that the site is visible to users



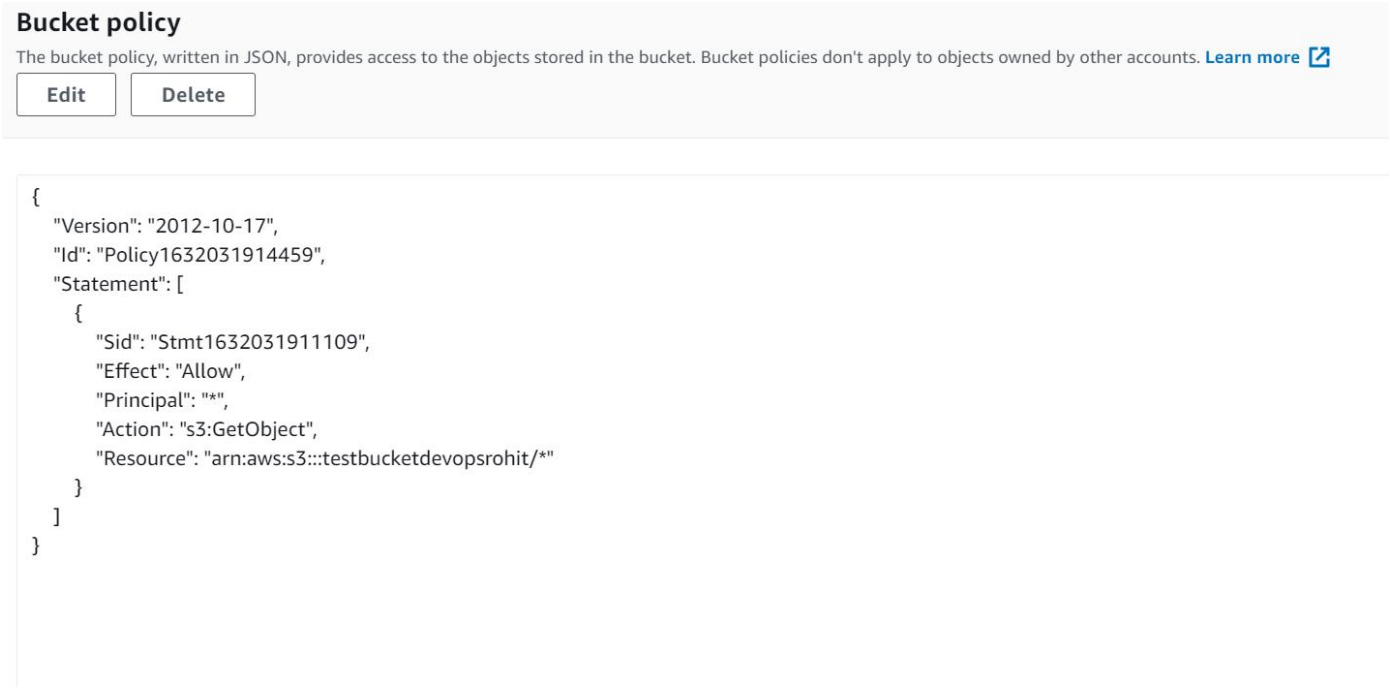
Now we have to add a policy to allow the users to view the objects/website

We use the Edit option in the permissions section for the bucket policy and the click on policy generator for creating a policy for the bucket and fill the details of what type of policy we want on what resources and generate a JSON document and then paste it in the bucket policy space and save the changes

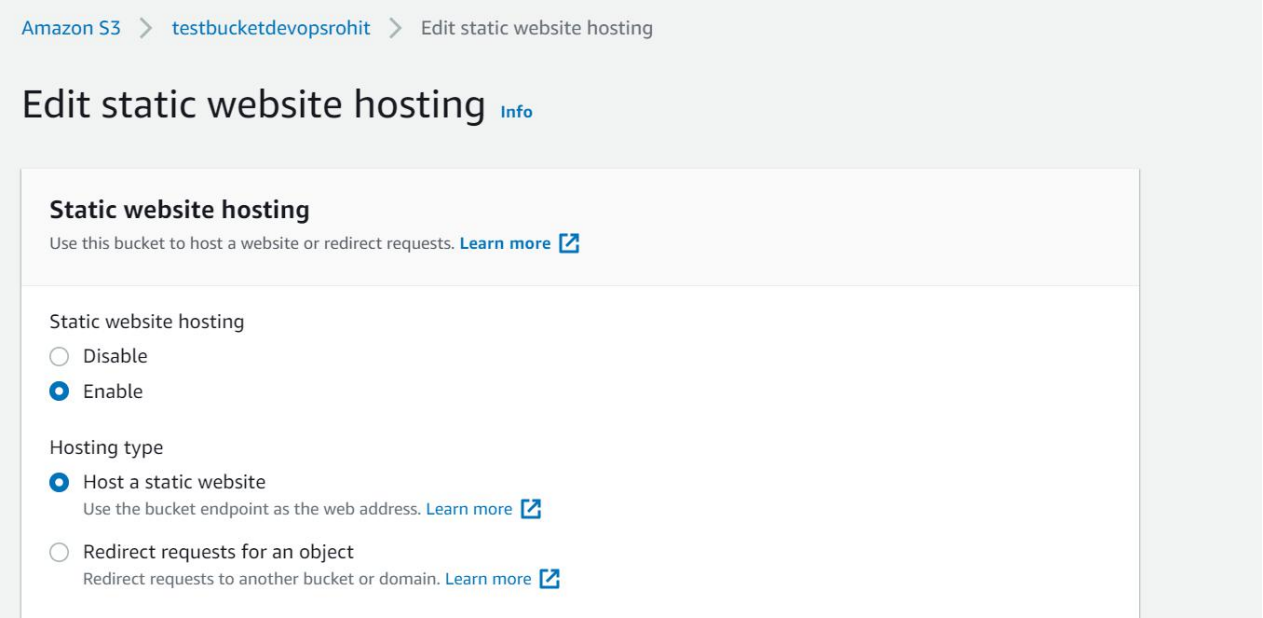




Now add the generated policy file to the bucket policy and save the changes



Now upload the html file with the name index.html and error.html and then go to properties and scroll down to Static website hosting and then edit



Add the required fields as per the requirement and then save the changes

Index document

Specify the home or default page of the website.

index.html

Error document - *optional*

This is returned when an error occurs.

error.html

Redirection rules – *optional*

Redirection rules, written in JSON, automatically redirect webpage requests for specific content. **Learn more**

```
1
```

Then you will get a url for the website that we have just hosted

Static website hosting

Use this bucket to host a website or redirect requests. **Learn more**

Edit

Static website hosting

Enabled

Hosting type

Bucket hosting

Bucket website endpoint

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. **Learn more**

http://testbucketdevopsrohit.s3-website.ap-south-1.amazonaws.com

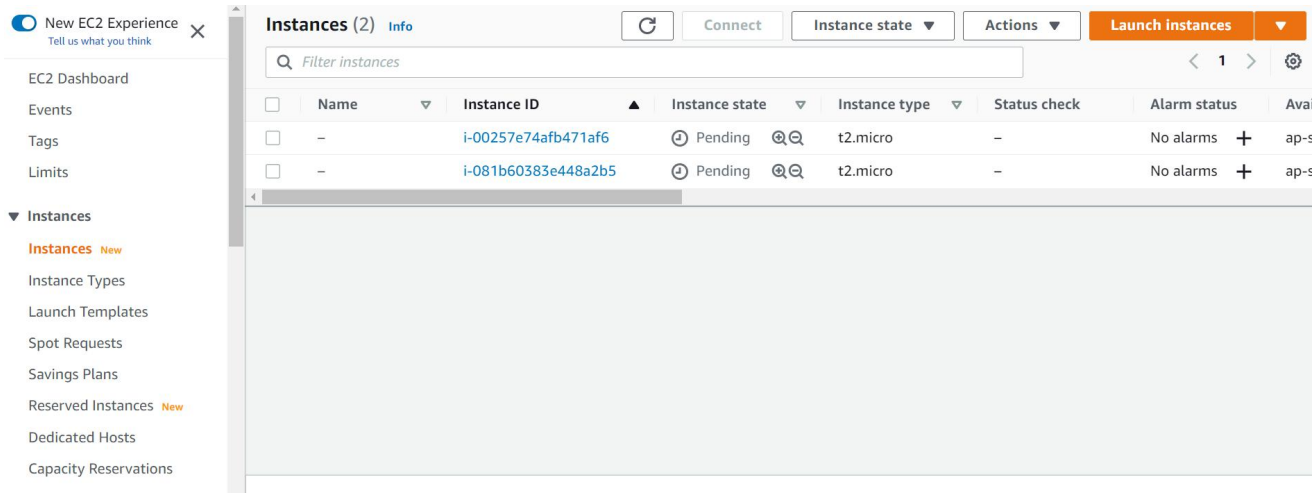
Click on the link/url and you can find the website that we have just created

I love DevOps and Cloud

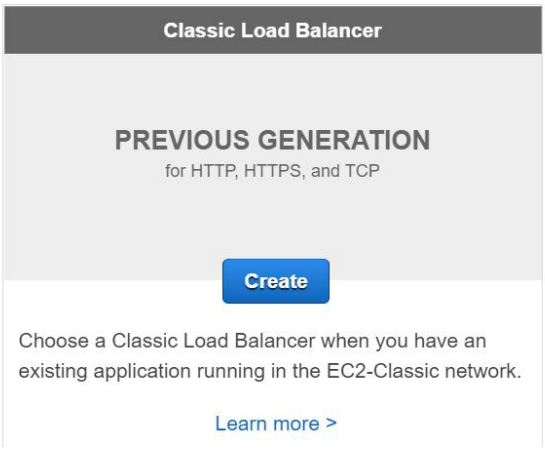
Hello world!

AWS-ELB & ASG:

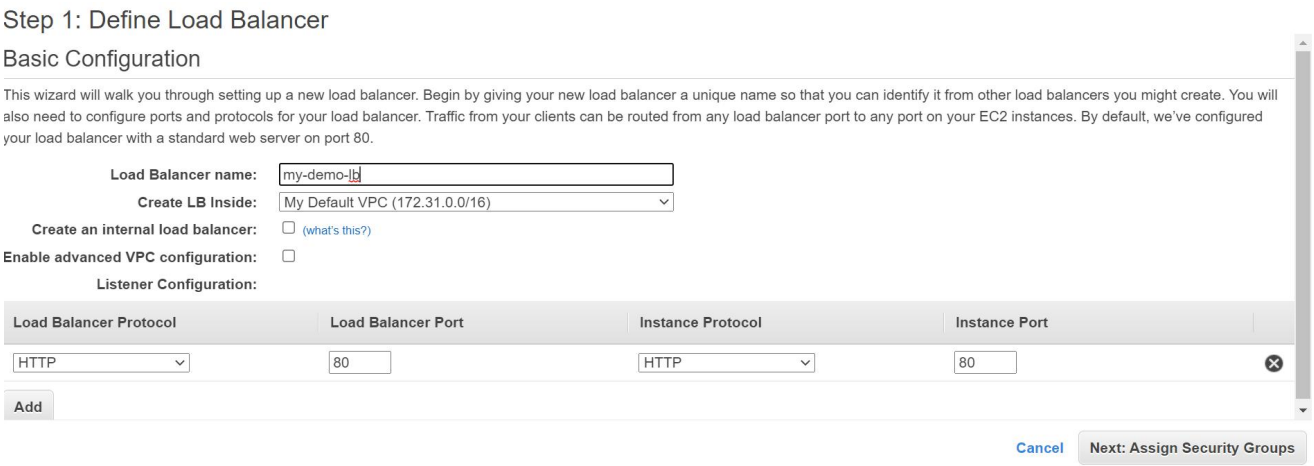
First of all we will launch to instances so that we can-attach ELB and ASG to them



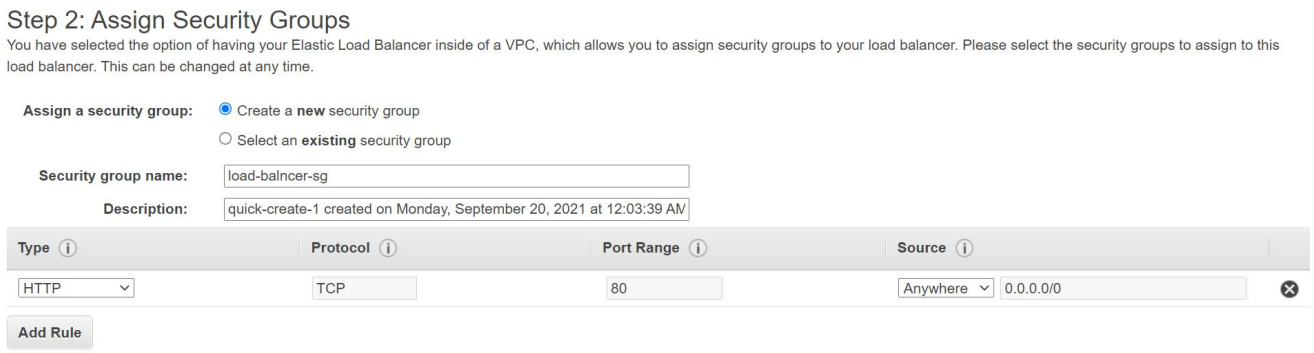
Then go to load balancers and select classic load balancer



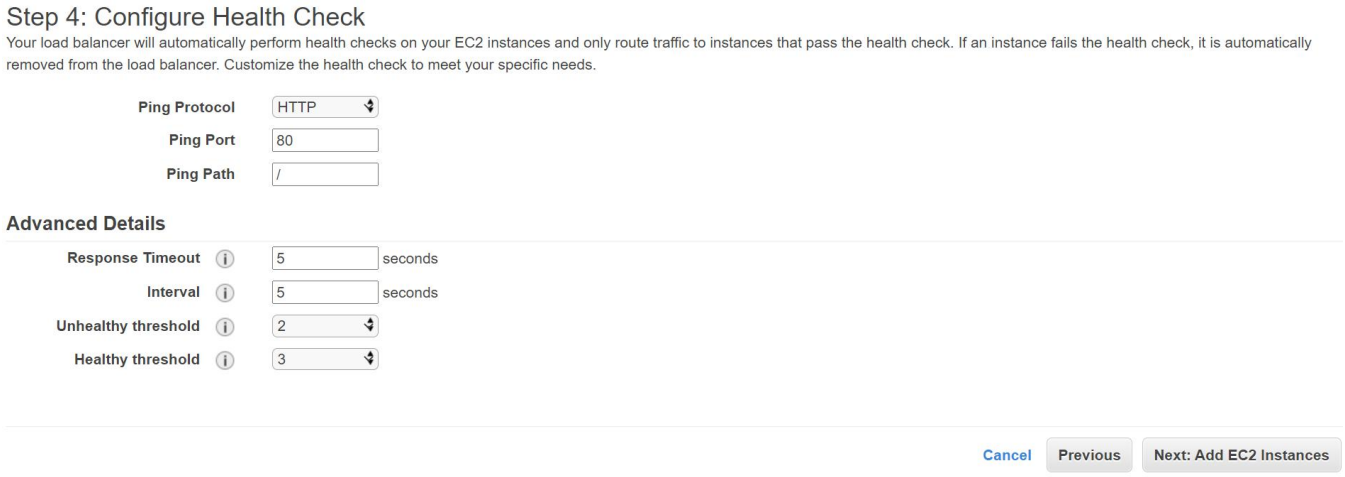
Selecting the load balancer type and rules



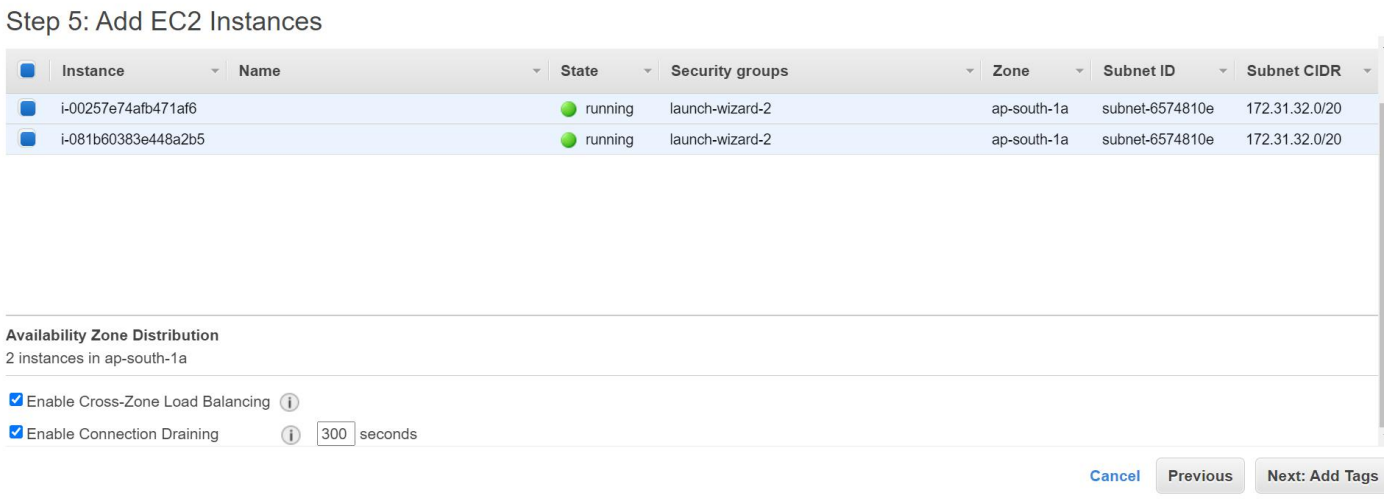
Creating a security group for our LB



Configuring the health checks:

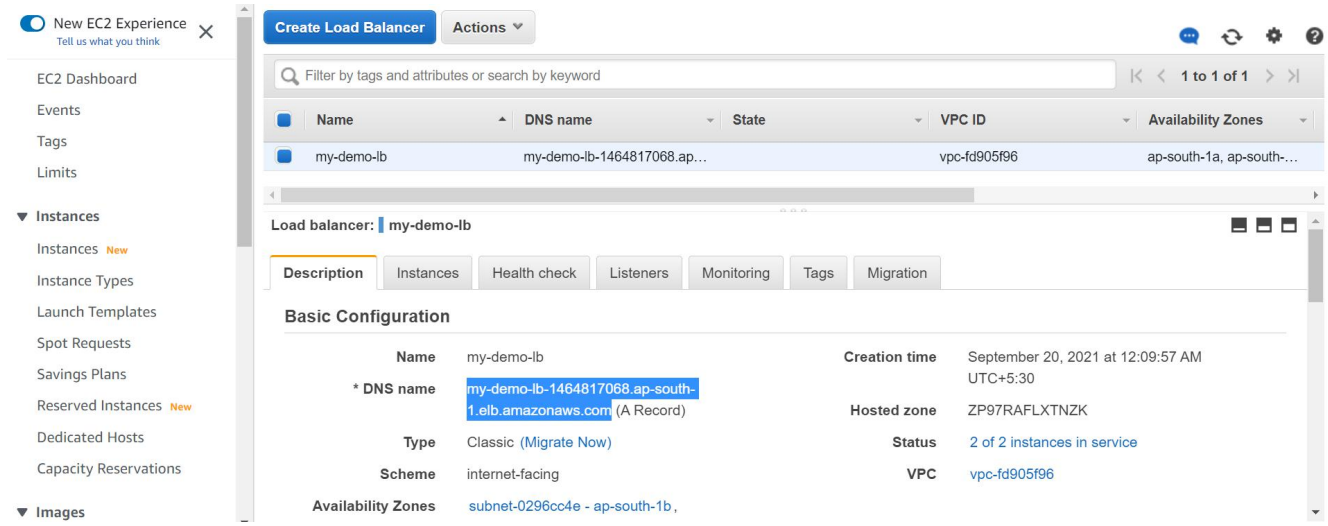


Adding the instances:



Now add tags and Review and create the Load balancer

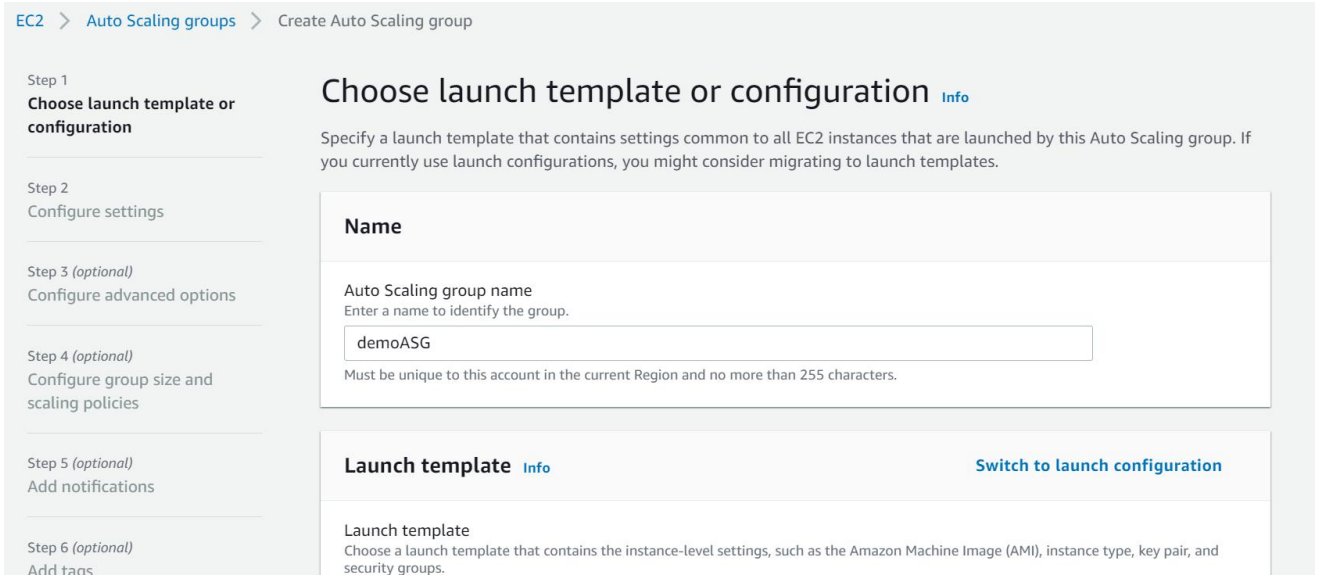
Now once when the load balancer is in service then we can go to the url and check it and when we refresh the browser we see different instance IPs

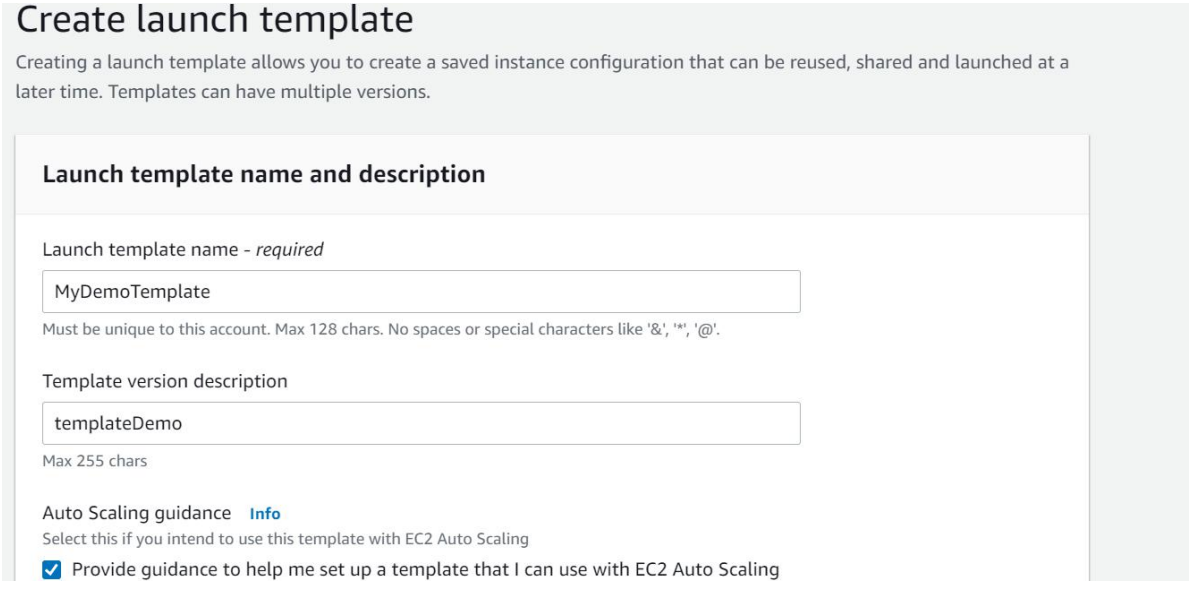


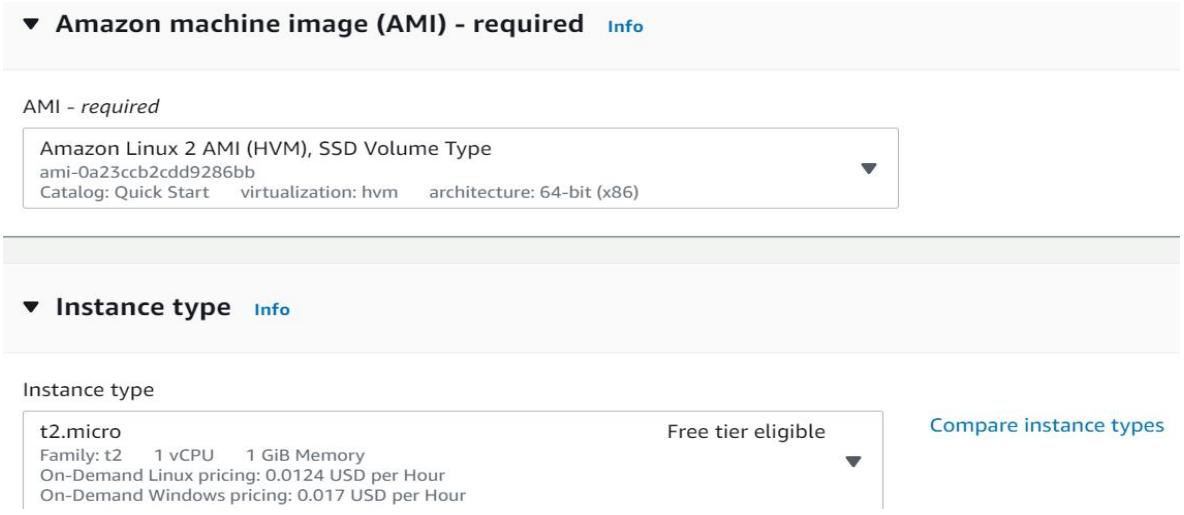
Hello World from ip-172-31-39-87.ap-south-1.compute.internal

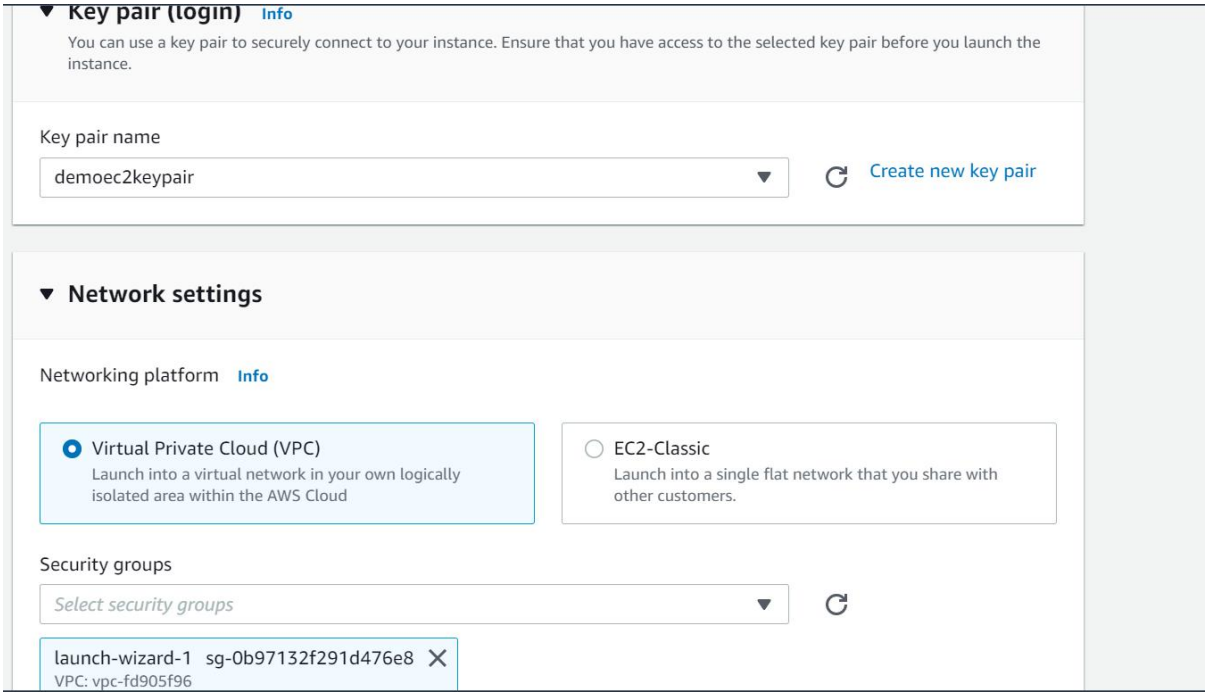
Hello World from ip-172-31-36-96.ap-south-1.compute.internal

Now we will attach an ASG for scaling of instances according to the requirement
First of all select the auto-scaling group section and click on create ASG
Then name the ASG and select the launch template









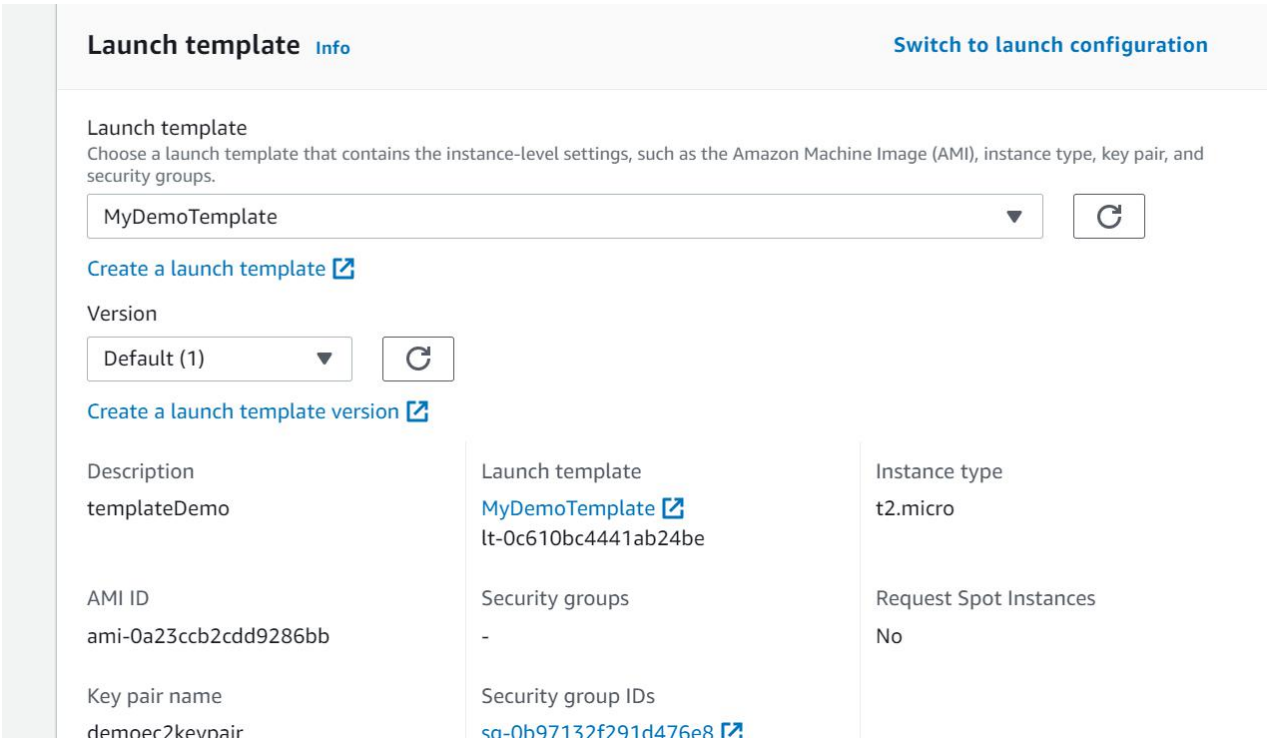
User data Info

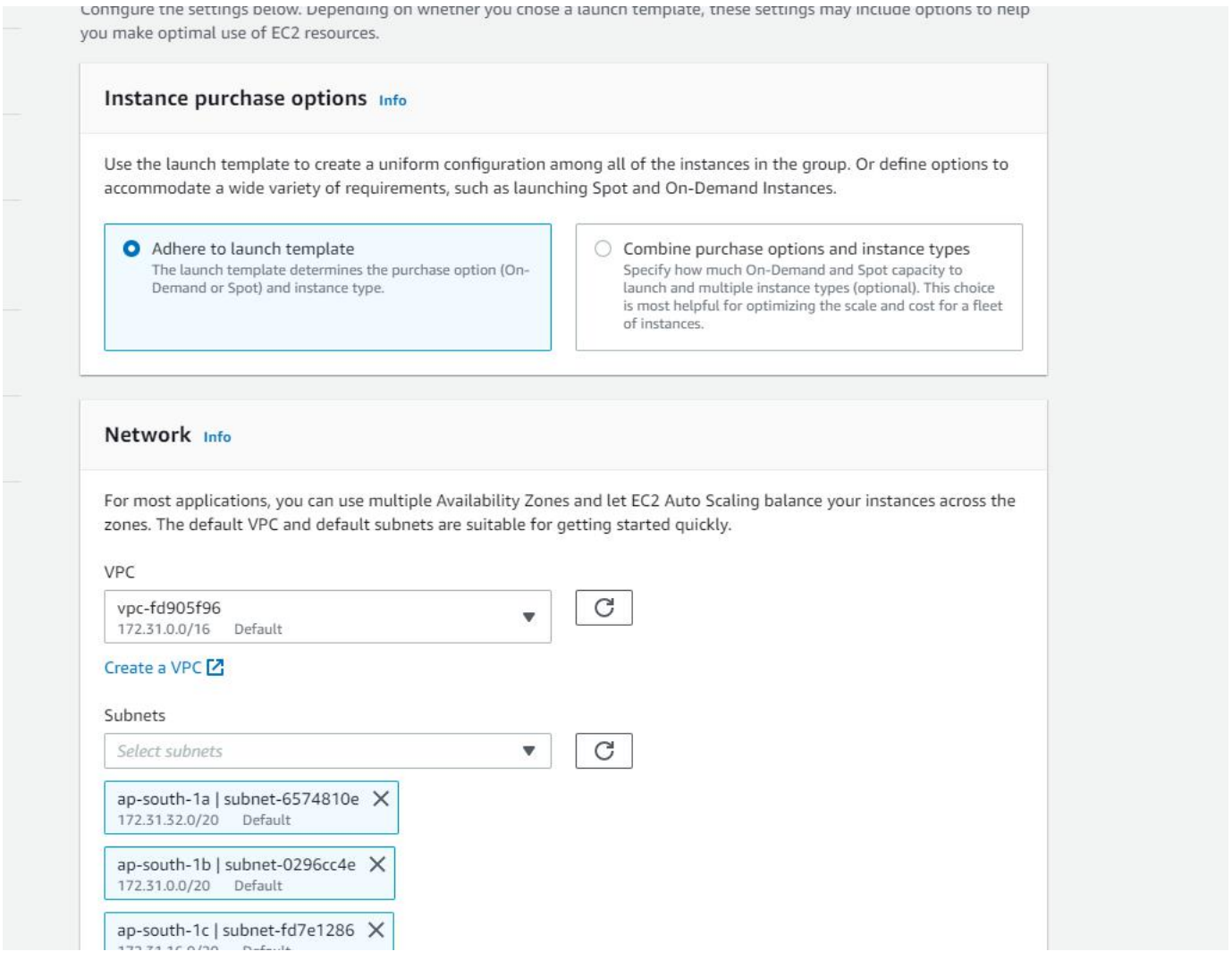
```
#!/bin/bash
sudo su
yum update -y
yum install -y httpd.x86_64
systemctl start httpd.service
systemctl enable httpd.service
echo "Hello World from $(hostname -f)" > /var/www/html/index.html
```

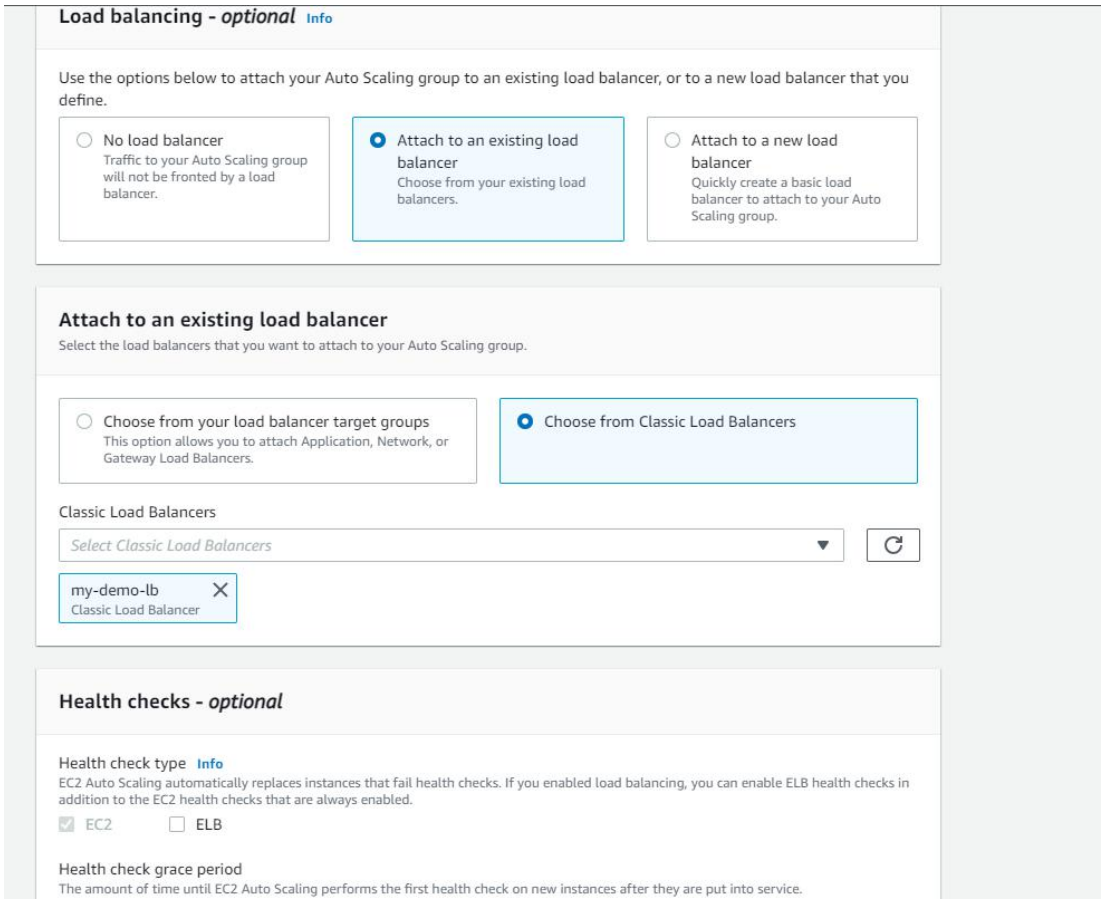
☐ User data has already been base64 encoded

Cancel **Create launch template**

Now select the launch template we just created







Group size - *optional* Info

Specify the size of the Auto Scaling group by changing the desired capacity. You can also specify minimum and maximum capacity limits. Your desired capacity must be within the limit range.

Desired capacity

2

Minimum capacity

1

Maximum capacity

5

Scaling policies - *optional*

Choose whether to use a scaling policy to dynamically resize your Auto Scaling group to meet changes in demand. Info

- ○ Target tracking scaling policy
  Choose a desired outcome and leave it to the scaling policy to add and remove capacity as needed to achieve that outcome.
- ● None

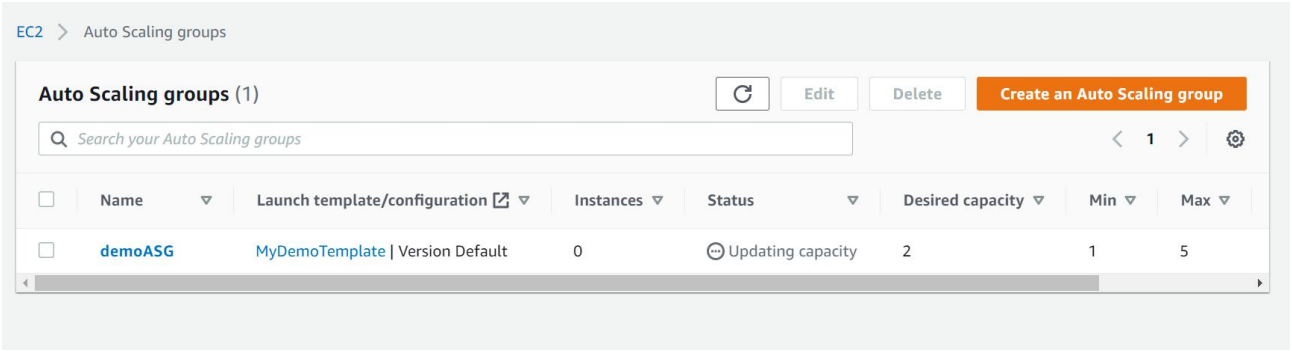
Instance scale-in protection - *optional*

Instance scale-in protection

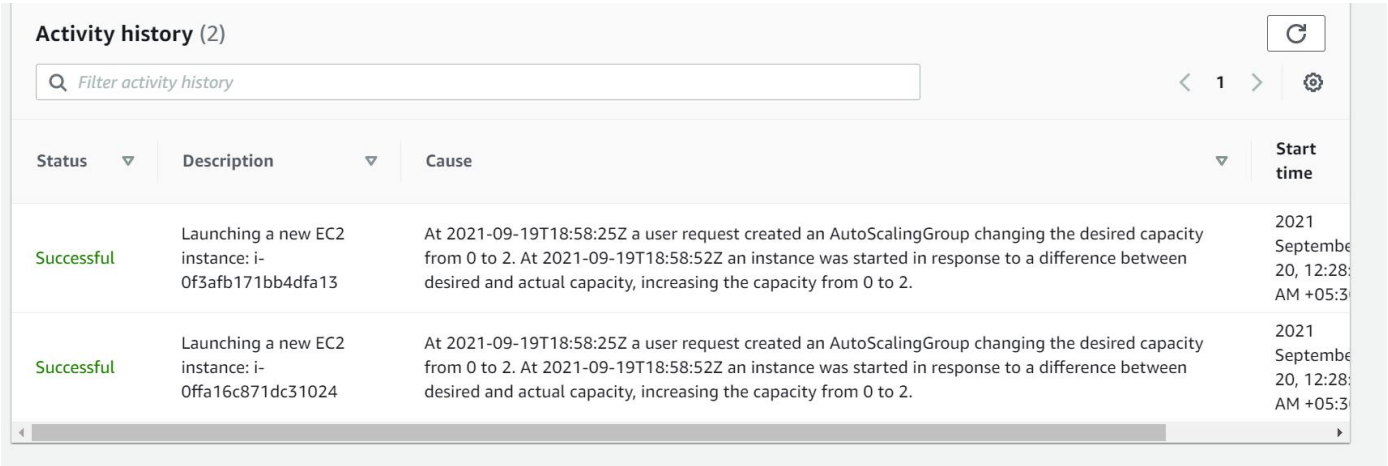
If protect from scale in is enabled, newly launched instances will be protected from scale in by default.

☐ Enable instance scale-in protection

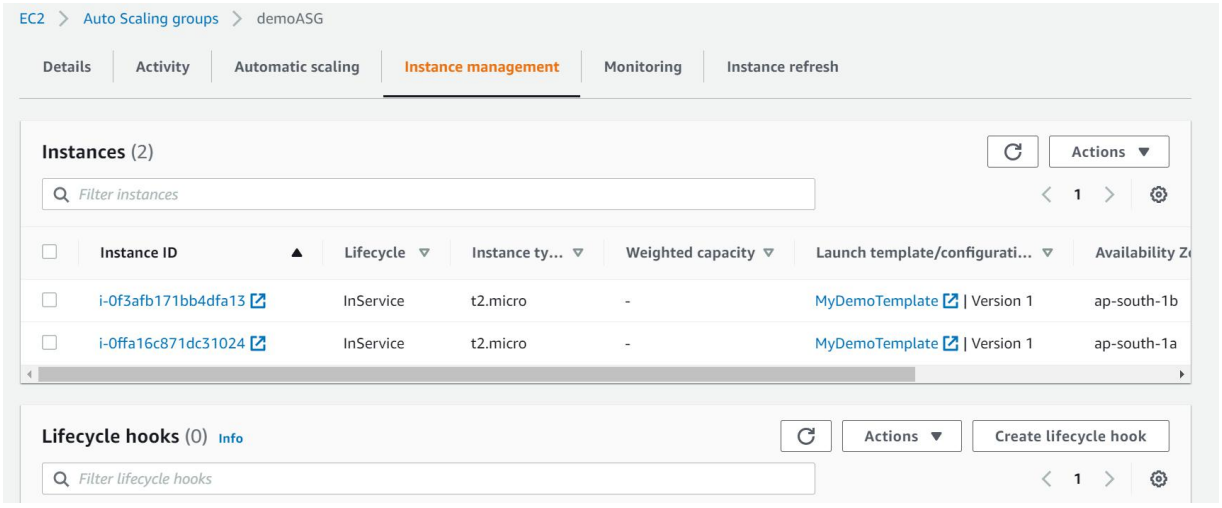
Next add tags and Review and Launch



By inspecting the Activity section of the ASG we can see the logs of how the instances are being created according to the rules



Inside instance management we can see we have 2 instance running



So if we go now to the load balancers url we can see two ips of the instances that are running and if we change the desired capacity to 3 or more(less than maximum capacity) then we can see that 3 instances are running and have been registered automatically inside the ASG also when one or more instances fails the health check or shuts-down the ASG will automatically configure another instance in place of those lost instances

Hello World from ip-172-31-1-117.ap-south-1.compute.internal

Hello World from ip-172-31-46-196.ap-south-1.compute.internal

Hello World from ip-172-31-36-96.ap-south-1.compute.internal

CHEF

Chef is an open source technology developed by Opscode. Adam Jacob, co-founder of Opscode is known as the founder of Chef. This technology uses Ruby encoding to develop basic building blocks like recipe and cookbooks. Chef is used in infrastructure automation and helps in reducing manual and repetitive tasks for infrastructure management.
Chef have got its own convention for different building blocks, which are required to manage and automate infrastructure.

Why Chef?
Chef is a configuration management technology used to automate the infrastructure provisioning. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company's server. It has the capability to get integrated with any of the cloud technology.
In DevOps, we use Chef to deploy and manage servers and applications in-house and on the cloud.

Features of Chef
Following are the most prominent features of Chef −

- Chef uses popular Ruby language to create a domain-specific language.
- Chef does not make assumptions on the current status of a node. It uses its mechanisms to get the current status of machine.
- Chef is ideal for deploying and managing the cloud server, storage, and software.

Building Blocks of Chef:
Recipe:
It can be defined as a collection of attributes which are used to manage the infrastructure. These attributes which are present in the recipe are used to change the existing state or setting a particular infrastructure node. They are loaded during Chef client run and comparted with the existing attribute of the node (machine). It then gets to the status which is defined in the node resource of the recipe. It is the main workhorse of the cookbook.

Cookbook
A cookbook is a collection of recipes. They are the basic building blocks which get uploaded to Chef server. When Chef run takes place, it ensures that the recipes present inside it gets a given infrastructure to the desired state as listed in the recipe.

Resource
It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **package** − Manages the packages on a node
- **service** − Manages the services on a node
- **user** − Manages the users on the node
- **group** − Manages groups
- **template** − Manages the files with embedded Ruby template
- **cookbook_file** − Transfers the files from the files subdirectory in the cookbook to a location on the node
- **file** − Manages the contents of a file on the node
- **directory** − Manages the directories on the node
- **execute** − Executes a command on the node
- **cron** − Edits an existing cron file on the node

Attribute:
They are basically settings. They can be thought of as a key value pair of anything which one wants to use in the cookbook. There are several different kinds of attributes that can be applied, with a different level of precedence over the final settings that the node operates under.

File:
It's a subdirectory within the cookbook that contains any static file which will be placed on the nodes that uses the cookbooks. A recipe then can be declared as a resource that moves the files from that directory to the final node.

Templates:
They are similar to files, but they are not static. Template files end with the .ebr extension, which means they contain embedded Ruby. They are mainly used to substitute an attribute value into the files to create the final file version that will be placed on the node.

Metadata.rb:
It is used to manage the metadata about the package. This includes details like the name and details of the package. It also includes things such as dependency information that tells which cookbooks this cookbook needs to operate. This allows the Chef server to build the run-list of the node correctly and ensures that all of the pieces are transferred correctly.

Default Cookbook Structure:

```
C:\chef\cookbooks\nginx>tree
Folder PATH listing for volume Local Disk
Volume serial number is BE8B-6427
C: ├──attributes
├──definitions
├──files
│   └──default
├──libraries
├──providers
├──recipes
├──resources
└──templates
    └──default
```

Installation:
Opscode provides a fully packaged version, which does not have any external prerequisites. This fully packaged Chef is called the omnibus installer.

On Windows Machine
Step 1 – Download the setup .msi file of chefDK on the machine.
Step 2 – Follow the installation steps and install it on the target location**.**

ChefDK Path Variable

$ echo $PATH
/c/opscode/chef/bin:/c/opscode/chefdk/bin:

On Linux Machine
In order to set up on the Linux machine, we need to first get curl on the machine.

Step 1 – Once curl is installed on the machine, we need to install Chef on the workstation using Opscode's omnibus Chef installer.

$ curl –L https://www.opscode.com/chef/install.sh | sudo bash

Step 2 – Install Ruby on the machine.

Step 3 – Add Ruby to path variable.

$ echo 'export PATH = "/opt/chef/embedded/bin:$PATH"' ≫ ~/.bash_profile && source ~/.bash_profile

The Omnibus Chef will install Ruby and all the required Ruby gems into **/opt/chef/embedded** by adding **/opt/chef/embedded/bin** directory to the .bash_profile file.
If Ruby is already installed, then install the Chef Ruby gem on the machine by running the following command.

$ gem install chef

KNIFE SETUP:

Knife is Chef's command-line tool to interact with the Chef server. One uses it for uploading cookbooks and managing other aspects of Chef. It provides an interface between the chefDK (Repo) on the local machine and the Chef server. It helps in managing –

- Chef nodes
- Cookbook
- Recipe
- Environments
- Cloud Resources
- Cloud Provisioning
- Installation on Chef client on Chef nodes

Knife provides a set of commands to manage Chef infrastructure.

Bootstrap Commands

- knife bootstrap [SSH_USER@]FQDN (options)

Client Commands

- knife client bulk delete REGEX (options)
- knife client create CLIENTNAME (options)
- knife client delete CLIENT (options)
- knife client edit CLIENT (options)
- Usage: C:/opscode/chef/bin/knife (options)
- knife client key delete CLIENT KEYNAME (options)
- knife client key edit CLIENT KEYNAME (options)
- knife client key list CLIENT (options)
- knife client key show CLIENT KEYNAME (options)
- knife client list (options)
- knife client reregister CLIENT (options)
- knife client show CLIENT (options)

Configure Commands

- knife configure (options)
- knife configure client DIRECTORY

Cookbook Commands

- knife cookbook bulk delete REGEX (options)
- knife cookbook create COOKBOOK (options)
- knife cookbook delete COOKBOOK VERSION (options)
- knife cookbook download COOKBOOK [VERSION] (options)
- knife cookbook list (options)
- knife cookbook metadata COOKBOOK (options)
- knife cookbook metadata from FILE (options)
- knife cookbook show COOKBOOK [VERSION] [PART] [FILENAME] (options)
- knife cookbook test [COOKBOOKS...] (options)
- knife cookbook upload [COOKBOOKS...] (options)

Cookbook Site Commands

- knife cookbook site download COOKBOOK [VERSION] (options)
- knife cookbook site install COOKBOOK [VERSION] (options)
- knife cookbook site list (options)
- knife cookbook site search QUERY (options)
- knife cookbook site share COOKBOOK [CATEGORY] (options)
- knife cookbook site show COOKBOOK [VERSION] (options)
- knife cookbook site unshare COOKBOOK

Data Bag Commands

- knife data bag create BAG [ITEM] (options)
- knife data bag delete BAG [ITEM] (options)
- knife data bag edit BAG ITEM (options)
- knife data bag from file BAG FILE|FOLDER [FILE|FOLDER..] (options)
- knife data bag list (options)
- knife data bag show BAG [ITEM] (options)

Environment Commands

- knife environment compare [ENVIRONMENT..] (options)
- knife environment create ENVIRONMENT (options)
- knife environment delete ENVIRONMENT (options)

- knife environment edit ENVIRONMENT (options)
- knife environment from file FILE [FILE..] (options)
- knife environment list (options)
- knife environment show ENVIRONMENT (options)

Exec Commands

- knife exec [SCRIPT] (options)

Help Commands

- knife help [list|TOPIC]

Index Commands

- knife index rebuild (options)

Node Commands

- knife node bulk delete REGEX (options)
- knife node create NODE (options)
- knife node delete NODE (options)
- knife node edit NODE (options)
- knife node environment set NODE ENVIRONMENT
- knife node from file FILE (options)
- knife node list (options)
- knife node run_list add [NODE] [ENTRY[,ENTRY]] (options)
- knife node run_list remove [NODE] [ENTRY[,ENTRY]] (options)
- knife node run_list set NODE ENTRIES (options)
- knife node show NODE (options)

OSC Commands

- knife osc_user create USER (options)
- knife osc_user delete USER (options)
- knife osc_user edit USER (options)
- knife osc_user list (options)
- knife osc_user reregister USER (options)
- knife osc_user show USER (options)

Path-Based Commands

- knife delete [PATTERN1 ... PATTERNn]

- knife deps PATTERN1 [PATTERNn]
- knife diff PATTERNS
- knife download PATTERNS
- knife edit [PATTERN1 ... PATTERNn]
- knife list [-dfR1p] [PATTERN1 ... PATTERNn]
- knife show [PATTERN1 ... PATTERNn]
- knife upload PATTERNS
- knife xargs [COMMAND]

Raw Commands

- knife raw REQUEST_PATH

Recipe Commands

- knife recipe list [PATTERN]

Role Commands

- knife role bulk delete REGEX (options)
- knife role create ROLE (options)
- knife role delete ROLE (options)
- knife role edit ROLE (options)
- knife role env_run_list add [ROLE] [ENVIRONMENT] [ENTRY[,ENTRY]] (options)
- knife role env_run_list clear [ROLE] [ENVIRONMENT]
- knife role env_run_list remove [ROLE] [ENVIRONMENT] [ENTRIES]
- knife role env_run_list replace [ROLE] [ENVIRONMENT] [OLD_ENTRY] [NEW_ENTRY]
- knife role env_run_list set [ROLE] [ENVIRONMENT] [ENTRIES]
- knife role from file FILE [FILE..] (options)
- knife role list (options)
- knife role run_list add [ROLE] [ENTRY[,ENTRY]] (options)
- knife role run_list clear [ROLE]
- knife role run_list remove [ROLE] [ENTRY]
- knife role run_list replace [ROLE] [OLD_ENTRY] [NEW_ENTRY]
- knife role run_list set [ROLE] [ENTRIES]
- knife role show ROLE (options)

Serve Commands

- knife serve (options)

SSH Commands

- knife ssh QUERY COMMAND (options)

SSL Commands

- knife ssl check [URL] (options)
- knife ssl fetch [URL] (options)

Status Commands

- knife status QUERY (options)

Tag Commands

- knife tag create NODE TAG ...
- knife tag delete NODE TAG ...
- knife tag list NODE

User Commands

- knife user create USERNAME DISPLAY_NAME FIRST_NAME LAST_NAME EMAIL PASSWORD (options)
- knife user delete USER (options)
- knife user edit USER (options)
- knife user key create USER (options)
- knife user key delete USER KEYNAME (options)
- knife user key edit USER KEYNAME (options)
- knife user key list USER (options)
- knife user key show USER KEYNAME (options)
- knife user list (options)
- knife user reregister USER (options)
- knife user show USER (options)

Knife Setup

In order to set up knife, one needs to move to **.chef** directory and create a **knife.rb** inside the chef repo, which tells knife about the configuration details. This will have a couple up details.

In the above code, we are using the hosted Chef server which uses the following two keys

```
validation_client_name   'ORG_NAME-validator'
validation_key           "#{current_dir}/ORGANIZATION-validator.pem"
```

Here, knife.rb tells knife which organization to use and where to find the private key. It tells knife where to find the users' private key

```
client_key               "#{current_dir}/USER.pem"
```

The following line of code tells knife we are using the hosted server.

```
chef_server_url         'https://api.chef.io/organizations/ORG_NAME'
```

Using the knife.rb file, the validator knife can now connect to your organization's hosted Opscode.

```
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                'node_name'
client_key               "#{current_dir}/USER.pem"
validation_client_name   'ORG_NAME-validator'
validation_key           "#{current_dir}/ORGANIZATION-validator.pem"
chef_server_url          'https://api.chef.io/organizations/ORG_NAME'
cache_type               'BasicFile'
cache_options( :path =>  "#{ENV['HOME']}/.chef/checksums" )
cookbook_path            ["#{current_dir}/../cookbooks"]
```

Cookbooks:

Cookbooks are fundamental working units of Chef, which consists of all the details related to working units, having the capability to modify configuration and the state of any system configured as a node on Chef infrastructure. Cookbooks can perform multiple tasks. Cookbooks contain values about the desired state of node. This is achieved in Chef by using the desired external libraries.

Key Components of a Cookbook

- Recipes
- Metadata
- Attributes
- Resources
- Templates

- Libraries
- Anything else that helps to create a system
-

Creating a Cookbook:

There are two ways to dynamically create a cookbook.

- **Using chef command**
- **Using knife utility**

Cookbook dependencies:

The features of defining cookbook dependencies help in managing cookbook. This feature is used when we want to use the functionality of one cookbook in other cookbooks.

For example, if one wants to compile C code then one needs to make sure that all the dependencies required to compile are installed. In order to do so, there might be separate cookbook which can perform such a function.

When we are using chef-server, we need to know such dependencies in cookbooks which should be decelerated in the cookbooks metadata file. This file is located at the top on the cookbook directory structure. It provides hints to the Chef server which helps in deploying cookbooks on the correct node.

Features of metadata.rb File

- Located at the top in the cookbook directory structure.
- Compiled when the cookbook is uploaded to Chef server using knife command.
- Compiled with knife cookbook metadata subcommand.
- Created automatically when the knife cookbook create command is run.

Configuration of metadata.rb:

Following is the default content of a metadata file.

```
name             'nginx'
maintainer       'YOUR_COMPANY_NAME'
maintainer_email 'YOUR_EMAIL'
license          'All rights reserved'
description      'Installs/Configures nginx'
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version          '0.1.0'
```

Role:
Roles in Chef are a logical way of grouping nodes. Typical cases are to have roles for web servers, database servers, and so on. One can set custom run list for all the nodes and override attribute value within roles.

Create a Role:

```
~/chef-repo $ subl roles/web_servers.rb
name "web_servers"
description "This role contains nodes, which act as web servers"
run_list "recipe[ntp]"
default_attributes 'ntp' => {
   'ntpdate' => {
      'disable' => true
   }
}
```

Once we have the role created, we need to upload to the Chef server.

Upload Role to Chef Server:

```
~/chef-repo $ knife role from file web_servers.rb
```

Now, we need to assign a role to a node called server.

Assign a Role to Node:

```
~/chef-repo $ knife node edit server
"run_list": [
   "role[web_servers]"
]
Saving updated run_list on node server
```

Run the Chef-Client

```
user@server:~$ sudo chef-client
...TRUNCATED OUTPUT...
[2013-07-25T13:28:24+00:00] INFO: Run List is [role[web_servers]]
[2013-07-25T13:28:24+00:00] INFO: Run List expands to [ntp]
...TRUNCATED OUTPUT...
```

How It Works:

- Define a role in a Ruby file inside the roles folder of Chef repository.
- A role consists of a name and a description attribute.
- A role consists of role-specific run list and role-specific attribute settings.
- Every node that has a role in its run list will have the role's run list exacted into its own.
- All the recipes in the role's run list will be executed on the node.
- The role will be uploaded to Chef server using the knife role from file command.
- The role will be added to the node run list.
- Running Chef client on a node having the role in its run list will execute all the recipes listed in the role

That's all for the beginner's guide also we have a lot more to cover like kubernetes, terraform,puppet,nagios,maven and other such important tools