

Configuration Manual

Docker Container Deployment in Diverse Network regions
Programme Name

Jayesh Chitnawis
Student ID: 22244328

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Jayesh Chitnawis
Student ID:	22244328
Programme:	Programme Name
Year:	2024
Module:	Docker Conatiner Deployment in Diverse Network regions
Supervisor:	Rashid Mijumbi
Submission Due Date:	16/09/2024
Project Title:	Configuration Manual
Word Count:	1500
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Jayesh Chitnawis
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Jayesh Chitnawis
22244328

Docker Container Deployment in Diverse Network regions

1 Introduction

This configuration manual will describe the steps of implementing and deploying the database or docker application in the network infrastructure proposed in the research paper.

Here is the detailed information for developing such kind of network. This scalable environment for docker containerised applications and relational database service instances. This environment will spread over multiple regions which gives us special features called high availability and fault tolerance with also multiple availability zones (AZs). This procedure includes Virtual Private cloud hosting for containers and also for RDS.

This environment requires ideal up time and data consistency across multiple nodes with heterogeneous network locations and diverse networks. This setup was proven to manage complex workloads, maintaining high standards.

1.1 Pre-requisites:

We confirm the following prerequisites are met:

1. AWS account: AWS Account with valid and appropriate super-user permissions to create and manage services. AWS CLI command line interface makes easier management of resources and IAM users.
2. Networking knowledge: understanding of networking and basic concepts. Routing and security groups. And familiar with AWS VPC controls for the creation of VPC with subnets and route tables.
3. Docker: docker installed on each Virtual machine (VM) and docker understanding to pull images and manage containers.
4. Text editor: to ensure and save YAML files on the local device.
5. Linux VM: we can create EC2 AWS Linux as well or else we can have our own Linux system.

2 Execution step to launch AWS EC2 Instance

2.1 AWS Management Console

Step1: open the AWS Management Console (Sign in using your AWS account credentials)

Step2: navigate to EC2 Dashboard

Step3: In the dashboard click “launch instance” It will redirect you “instance ” wizard.

Step4: Name your instance > select Amazon Machine Image > for current instance we go with “no key pair required”

Step5: for network settings we click on edit > select Subnet > for current instance we can go for subnet US-East-1a > in security group same as it.

Step6: now click on “Review and Launch”

Step7: After launching it should redirect to EC2 Dashboard and it goes for view instance on Dashboard, As it will come running after some time, so here instance will be ready to use.

Step8: once an instance is open, click on connect, and it will redirect to AWS CLI (Command line Interface).

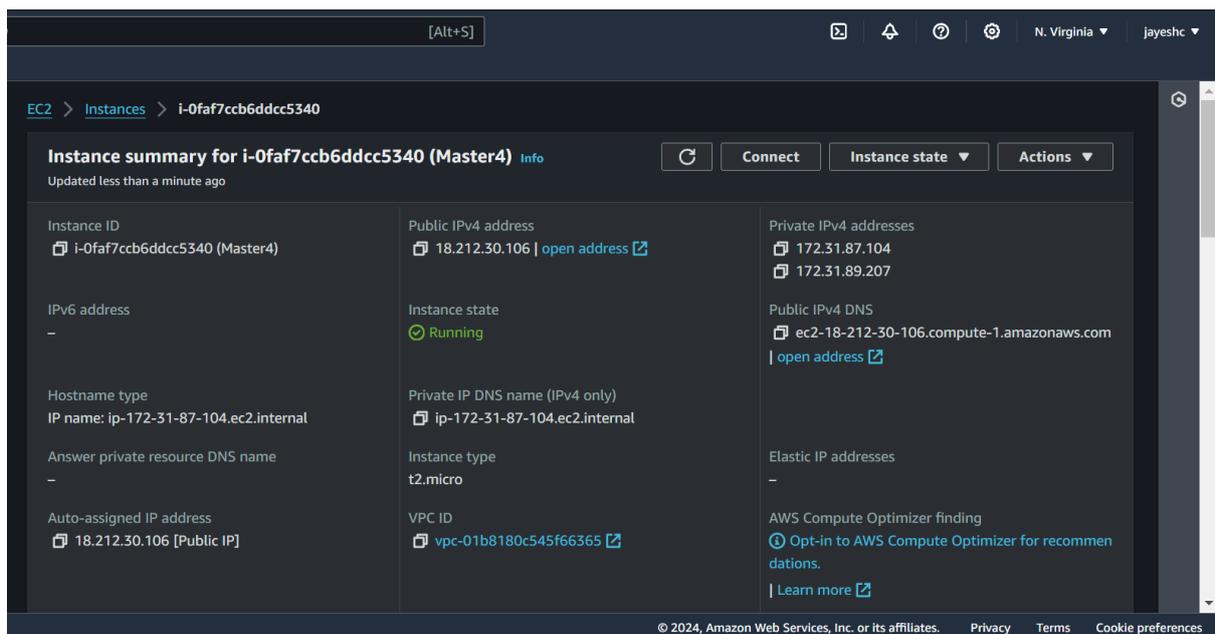


Figure 1: Master Node EC2

3 Docker installation on EC2 instances

Step1: Before installing Docker, you should update the package index.

“sudo yum update -y”

“sudo yum upgrade”

Step2: Installing docker To install Docker on Amazon Linux, you can use the following command:

“sudo yum install docker -y”

Step3: Install Docker Compose

“sudo yum install docker-compose”

```
-bash: $'\B[200-sudo': command not found
[ec2-user@ip-172-31-38-79 ~]$ sudo yum install -y docker
Last metadata expiration check: 0:05:38 ago on Wed Aug 7 14:16:55 2024.
Dependencies resolved.

```

Package	Architecture	Version	Repository	Size
Installing:				
docker	x86_64	25.0.6-1.amzn2023.0.1	amazonlinux	44 M
Installing dependencies:				
containerd	x86_64	1.7.20-1.amzn2023.0.1	amazonlinux	35 M
iptables-libs	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	401 k
iptables-nft	x86_64	1.8.8-3.amzn2023.0.2	amazonlinux	183 k
libnftnl	x86_64	3.0-1.amzn2023.0.1	amazonlinux	75 k
libnetfilter_comtrack	x86_64	1.0.8-2.amzn2023.0.2	amazonlinux	58 k
libnetfilter_cthelper	x86_64	1.0.1-19.amzn2023.0.2	amazonlinux	30 k
libnftnl	x86_64	1.2.2-2.amzn2023.0.2	amazonlinux	84 k
pigs	x86_64	2.5-1.amzn2023.0.3	amazonlinux	83 k
runc	x86_64	1.1.11-1.amzn2023.0.1	amazonlinux	3.0 M

```
Transaction Summary
Install 10 Packages
Total download size: 84 M
Installed size: 317 M
Downloading Packages:
(1/10): iptables-libs-1.8.8-3.amzn2023.0.2.x86_64.rpm 3.3 MB/s | 401 kB 00:00
```

Figure 2: Docker installation on Master node

here show case for master node, As we are using push model, we need to perform same steps on Slave nodes as well.

As we perform Same operations on Slave, for we need to deploy 1 EC2 as Slave, this slave can created any other region as well and use of different subnet is also allowed as well.

3.1 Pulling source code From GitHub

On master node and on Slave node we pull code from GitHub,

Docker file for Master and Slave is stored in GitHub repository. it also consist of one YAML file of Docker Compose, reason to use Docker Compose, as it will define and share multi-container applications. It have YAML file as it servers the purpose of defining the services with single command. If we do not use Docker Compose, then we would need to write many arguments in Docker.

3.2 VxLAN and OVS Connection

As Master is on in another region or in another subnet and Slave is another as well so, to connect them we use VxLAN as tunnel for connection. OVS supports VxLAN and provides managing and routing of traffic packets between VM's

Perform Both the Steps in Master as well as Slave Node.

Add Bridge

For Master Node

Step1: `ovs-vsctl add-br master-br`

For Slave Node

Step2: `ovs-vsctl add-br slave-br`

Internal Port Bridge

For Master node

step3: `"sudo - ovs - vsctl - add - portmaster - br
master - internal - --set - interface
master - internal - type = internal"`

For Slave node

Step4: `"sudo ovs - vsctl add - portslave - br
slave - internal - --set - interfaceslave - internal - type = internal"`

Creating Vxlan tunnel Through Private IP

For Master node

step5: `sudo ovs - vsctl add - portmaster - brmaster - vxlan - --set interface master -
vxlan type = vxlan options : remote_ip = 172.31.20.180 options : key = 2000`

for Slave node

step6: `"sudo ovs - vsctl add - portslave - brslave - vxlan - --set interface slave -
vxlan type = vxlan options : remote_ip = 172.31.89.207 options : key = 2000"`

Now we Assigned them Static IP.

Adding overlay IP

For Master node

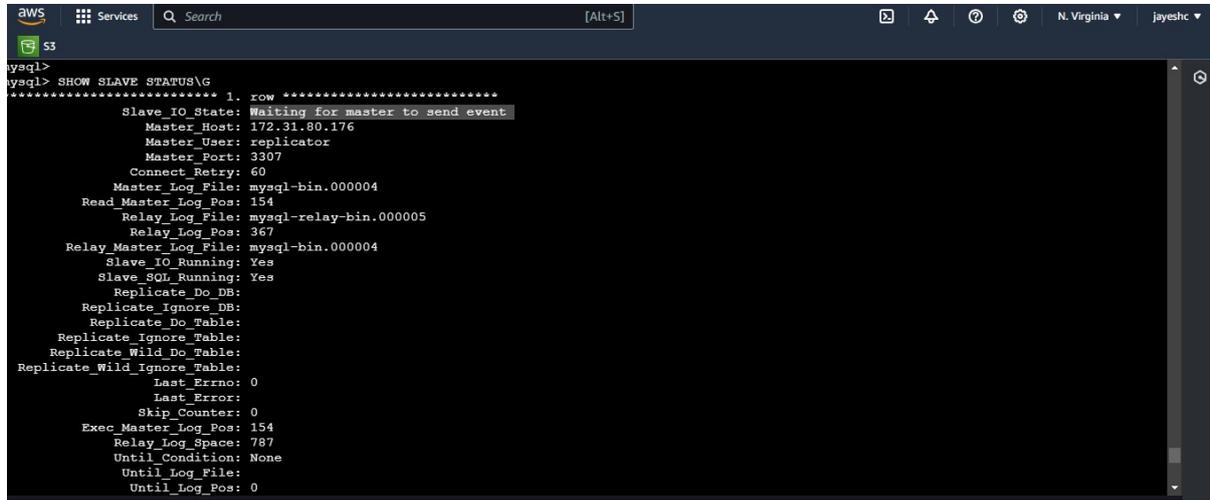
step7: `"sudo ifconfig master-internal 10.20.31.100/24 mtu 1450 up"`

for Slave node

step8:”sudo ifconfig slave-internal 10.20.32.200/24 mtu 1450 up”

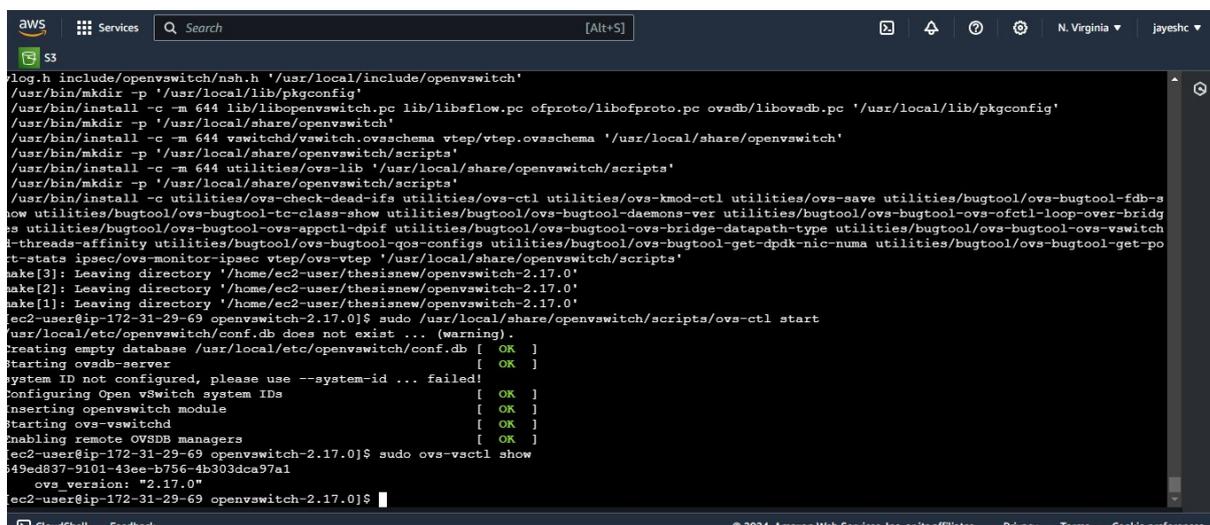
To Prove OVS architecture

step9:” sudo ovs-vsctl show”



```
mysql> SHOW SLAVE STATUS\G
mysql>
+-----+
| Slave_IO_State | Waiting for master to send event |
+-----+
| Master_Host    | 172.31.80.176                    |
+-----+
| Master_User    | replicator                       |
+-----+
| Master_Port    | 3307                             |
+-----+
| Connect_Retry  | 60                               |
+-----+
| Master_Log_File | mysql-bin.000004                |
+-----+
| Read_Master_Log_Pos | 154                             |
+-----+
| Relay_Log_File | mysql-relay-bin.000005          |
+-----+
| Relay_Log_Pos  | 367                             |
+-----+
| Relay_Master_Log_File | mysql-bin.000004                |
+-----+
| Slave_IO_Running | Yes                             |
+-----+
| Slave_SQL_Running | Yes                             |
+-----+
| Replicate_Do_DB |                                 |
+-----+
| Replicate_Ignore_DB |                                 |
+-----+
| Replicate_Do_Table |                                 |
+-----+
| Replicate_Ignore_Table |                                 |
+-----+
| Replicate_Wild_Do_Table |                                 |
+-----+
| Replicate_Wild_Ignore_Table |                                 |
+-----+
| Last_Errno    | 0                               |
+-----+
| Last_Error    |                                 |
+-----+
| Skip_Counter  | 0                               |
+-----+
| Exec_Master_Log_Pos | 154                             |
+-----+
| Relay_Log_Space | 787                             |
+-----+
| Until_Condition | None                            |
+-----+
| Until_Log_File |                                 |
+-----+
| Until_Log_Pos  | 0                               |
+-----+
```

Figure 3: Master and Slave valid connection



```
log.h include/openvswitch/nsh.h '/usr/local/include/openvswitch'
/usr/bin/mkdir -p '/usr/local/lib/pkgconfig'
/usr/bin/install -c -m 644 lib/libopenvswitch.pc lib/libflow.pc ofproto/libofproto.pc ovsdb/libovsdb.pc '/usr/local/lib/pkgconfig'
/usr/bin/mkdir -p '/usr/local/share/openvswitch'
/usr/bin/install -c -m 644 vswitchd/vswitch.ovsschema vtep/vtep.ovsschema '/usr/local/share/openvswitch'
/usr/bin/mkdir -p '/usr/local/share/openvswitch/scripts'
/usr/bin/install -c -m 644 utilities/ovs-lib '/usr/local/share/openvswitch/scripts'
/usr/bin/mkdir -p '/usr/local/share/openvswitch/scripts'
/usr/bin/install -c utilities/ovs-check-dead-ifs utilities/ovs-ctl utilities/ovs-kmod-ctl utilities/ovs-save utilities/bugtool/ovs-bugtool-fdb-s
ow utilities/bugtool/ovs-bugtool-tc-class-show utilities/bugtool/ovs-bugtool-daemons-ver utilities/bugtool/ovs-bugtool-ovs-ofctl-loop-over-bridg
s utilities/bugtool/ovs-bugtool-ovs-appctl-dpif utilities/bugtool/ovs-bugtool-ovs-bridge-datapath-type utilities/bugtool/ovs-bugtool-ovs-vswitch
t-threads-affinity utilities/bugtool/ovs-bugtool-qos-configs utilities/bugtool/ovs-bugtool-get-dpdk-nic-numa utilities/bugtool/ovs-bugtool-get-po
rt-stats ipsec/ovs-monitor-ipsec vtep/ovs-vtep '/usr/local/share/openvswitch/scripts'
make[3]: Leaving directory '/home/ec2-user/thesisnew/openvswitch-2.17.0'
make[2]: Leaving directory '/home/ec2-user/thesisnew/openvswitch-2.17.0'
make[1]: Leaving directory '/home/ec2-user/thesisnew/openvswitch-2.17.0'
ec2-user@ip-172-31-29-69 openvswitch-2.17.0]$ sudo /usr/local/share/openvswitch/scripts/ovs-ctl start
/usr/local/etc/openvswitch/conf.db does not exist ... (warning)
creating empty database /usr/local/etc/openvswitch/conf.db [ OK ]
starting ovsdb-server [ OK ]
system ID not configured, please use --system-id ... failed!
configuring Open vSwitch system IDs [ OK ]
inserting openvswitch module [ OK ]
starting ovs-vswitchd [ OK ]
enabling remote OVSDB managers [ OK ]
ec2-user@ip-172-31-29-69 openvswitch-2.17.0]$ sudo ovs-vsctl show
49ed837-9101-43ee-b756-4b303dca97a1
  ovs version: "2.17.0"
ec2-user@ip-172-31-29-69 openvswitch-2.17.0]$
```

Figure 4: OVS installation

4 For Creation of NFV

For creation of NFV, virtualized network.

We create an EC2 instance for virtualized functions. And Navigate to Route tables. Here we use NFV instance IP on VxLAN master, so all packets route through NFV.

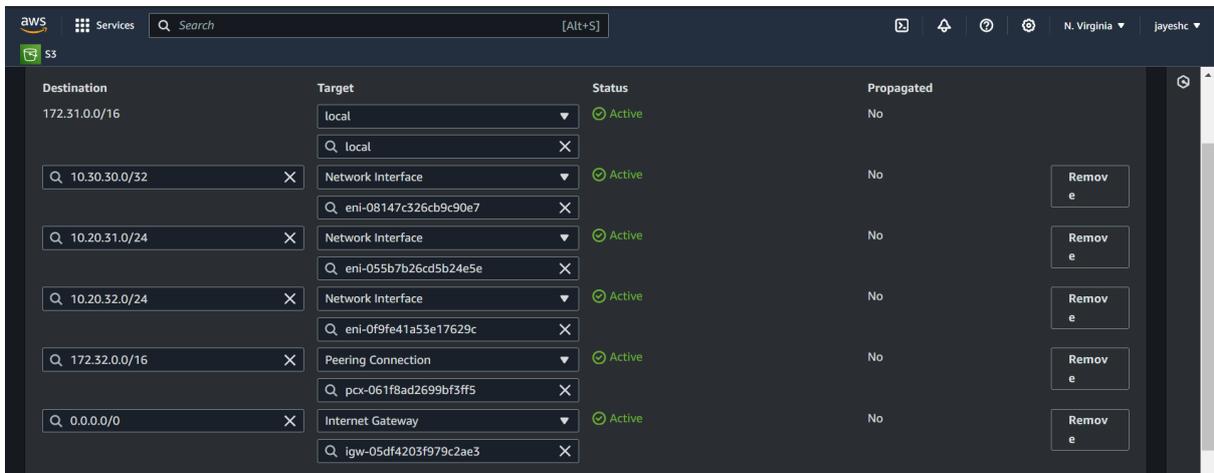


Figure 5: Routing Traffic to NFV

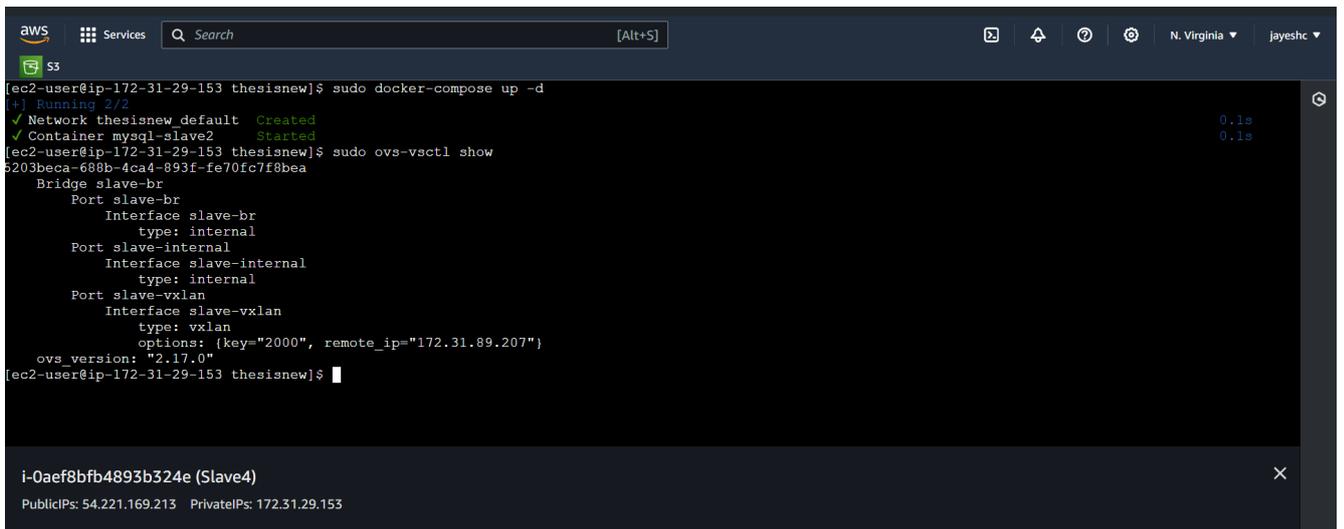


Figure 6: Connection at slave bridges

5 Deploying RDS Multi region and Availability Zone

```
docker exec -it mysql-slave2 mysql -uroot -prootpassword -e "
STOP SLAVE IO_THREAD FOR CHANNEL '';
CHANGE MASTER TO
    MASTER_HOST='10.20.31.100',
    MASTER_USER='replicator',
    MASTER_PASSWORD='passpass23',
    MASTER_PORT=3306,
    MASTER_LOG_FILE='mysql-bin.000001',
    MASTER_LOG_POS=154;|
START SLAVE;"
```

Figure 7: Connection of RDS at Master Node

6 To check status of RDS database at Master node

```
-- Check Master status
docker exec -it mysql-master2 mysql -uroot -prootpassword -e "SHOW MASTER STATUS\G"
```

Figure 8: Connection of RDS at Master Node

If we want to "Ping" and check connection, then we need to get Master node Internal IP. As here we ping Slave from Master Node. To check Master-Internal IP we use *ifconfig* command. And note down Master IP.

7 To check status of RDS database at Master node

We use *tcpdump* command for analyzing packet traffic.

8 Creating AWS VPC's with Multi region and Multi-AZ

This infrastructure aims to design robust and scalable RDS setup here will achieve Multi region and Multi-AZ.

```

master-internal: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1450
  inet 10.20.31.100 netmask 255.255.255.0 broadcast 10.20.31.255
  inet6 fe80::6498:1ff:feb2:2fe9 prefixlen 64 scopeid 0x20<link>
  ether 66:98:01:b2:2f:e9 txqueuelen 1000 (Ethernet)
  RX packets 0 bytes 0 (0.0 B)
  RX errors 0 dropped 10 overruns 0 frame 0
  TX packets 0 bytes 0 (0.0 B)
  TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vethd7297fc: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
  inet6 fe80::b884:54ff:feae:bf09 prefixlen 64 scopeid 0x20<link>
  ether ba:84:54:ae:bf:09 txqueuelen 0 (Ethernet)
  RX packets 797 bytes 1842620 (1.7 MiB)
  RX errors 0 dropped 0 overruns 0 frame 0
  TX packets 811 bytes 62587 (61.1 KiB)

```

i-0faf7ccb6ddcc5340 (Master4)

PublicIPs: 18.212.30.106 PrivateIPs: 172.31.87.104

Figure 9: Note Master-Internal IP

```

-- Check Master status
docker exec -it mysql-master2 mysql -uroot -prootpassword -e "SHOW MASTER STATUS\G"

```

Figure 10: Connection of RDS at Master Node

```

aws Services Search [Alt+S] N. Virginia jayeshc
s3
[ec2-user@ip-172-31-87-104 thesisnew]$ docker-compose up -d
[*] Running 1/1
[ec2-user@ip-172-31-87-104 thesisnew]$ sudo tcpdump -i enX0 host 172.32.2.146
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enX0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
08:31:06.734373 IP 172.32.2.146.43092 > ip-172-31-87-104.ec2.internal.mysql: Flags [S], seq 725237878, win 62727, options [mss 1460,sackOK,TS val 2631765683 ecr 0,nop,wscale 7], length 0
08:31:06.734457 IP ip-172-31-87-104.ec2.internal.mysql > 172.32.2.146.43092: Flags [S.], seq 3125548337, ack 725237879, win 65160, options [mss 1460,sackOK,TS val 4282369979 ecr 2631765683,nop,wscale 7], length 0
08:31:06.744868 IP 172.32.2.146.43092 > ip-172-31-87-104.ec2.internal.mysql: Flags [.], ack 1, win 491, options [nop,nop,TS val 2631765693 ecr 4282369979], length 0
08:31:06.745059 IP ip-172-31-87-104.ec2.internal.mysql > 172.32.2.146.43092: Flags [P.], seq 1:83, ack 1, win 510, options [nop,nop,TS val 4282369990 ecr 2631765693], length 82
08:31:06.755409 IP 172.32.2.146.43092 > ip-172-31-87-104.ec2.internal.mysql: Flags [.], ack 83, win 491, options [nop,nop,TS val 2631765704 ecr 4282369990], length 0
08:31:16.755225 IP ip-172-31-87-104.ec2.internal.mysql > 172.32.2.146.43092: Flags [P.], seq 83, ack 1, win 510, options [nop,nop,TS val 428238000 ecr 2631765704], length 0
08:31:16.808025 IP 172.32.2.146.43092 > ip-172-31-87-104.ec2.internal.mysql: Flags [.], ack 84, win 491, options [nop,nop,TS val 2631775757 ecr 4282380000], length 0

i-0faf7ccb6ddcc5340 (Master4)
PublicIPs: 18.212.30.106 PrivateIPs: 172.31.87.104

```

Figure 11: Packet Traffic in Master node

Step 1: Creating VPC with AWS VPC dashboard Here we assume IPv4 CIDR and Tenancy set as basic/default Launching VPC with 3 subnets

Step2: Subnet 1: MasterDB this needs to be a private subnet

Step3: Current availability zone can be US-East-1a

Step 4: And small range of IPv4 CIDR

Step 5 Subnet 2: slaveDB

step 6: Availability zone be US-EAST-1b

Step 7:And small range of IPv4 CIDR

Step 8: Public IP: (auto assigned)

Step 9: Subnet 3: (NFV Load Balancer): NFV-sLB

Step 10: Availability zone be US-EAST-1a And small range of IPv4 CIDR

Step 11: Public IP: (auto assigned)

Step 12: Configuring routing table to confirm MasterDB subnet

Step 13: Private routing table For masterDB subnet / no internet gateway association

Step 14: Public routing table For slave and NFV connected to internet gateway.

Step 15: SlaveDB will use LB's ip to connected masterDB

Step 16: Configure listeners to handle incoming packets and request from MasterDB

Step 17: Creating secondary VPC with one subnet Single subnet

Step 18: Availability zone be US-EAST-1a And small range of IPv4 CIDR

Step 19: Public IP: (auto assigned) connected to internet gateway VPC Peering

Step 20 Connection 2 VPC is established by Transit gateway which is also known as AWS Transit Gateway.

Step 21: So here we prove cross region connectivity. Setting up RDS instance

Step 22: MasterDB from first VPC on instance and hence subnet enabling Multi-AZ

Step 23: So slaveDB configure for read and mimic Setting cross region Cloning.

Step 24: As cross connectivity is ensured, MasterDB from first VPC and SingleDB from another VPC prove to achieve RDS Multi-region