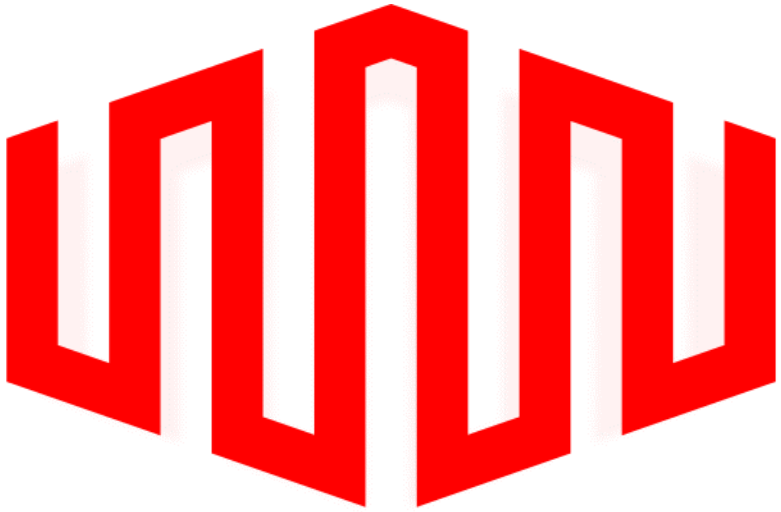


EQUINIX

SDN/NFV VNF Service Chaining

Test Readiness Review

Project Instructor: Dr. Kevin Gifford

Project Advisor: Dr. Levi Perigo, Mr. Brooke Mouland (Equinix)

TEAM 9:

Dashmeet Singh Anand

Hariharakumar Narasimhakumar

Rohit Dilip Kulkarni

Sarang Ninale

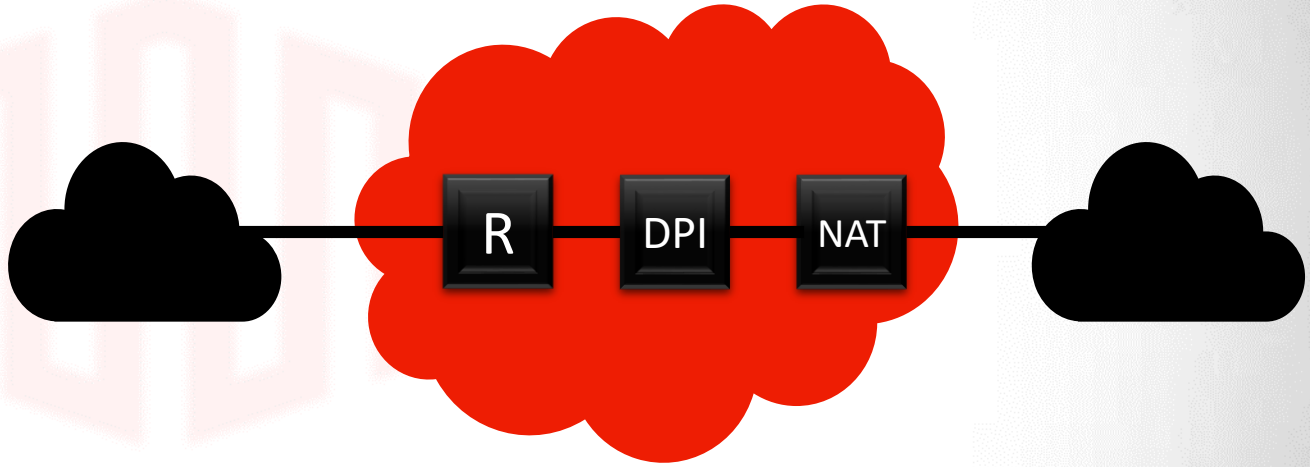
Agenda

1. Project Overview
2. Schedule
3. Test Readiness
4. Parts and Procurement

Project Overview

What is Service Chaining?

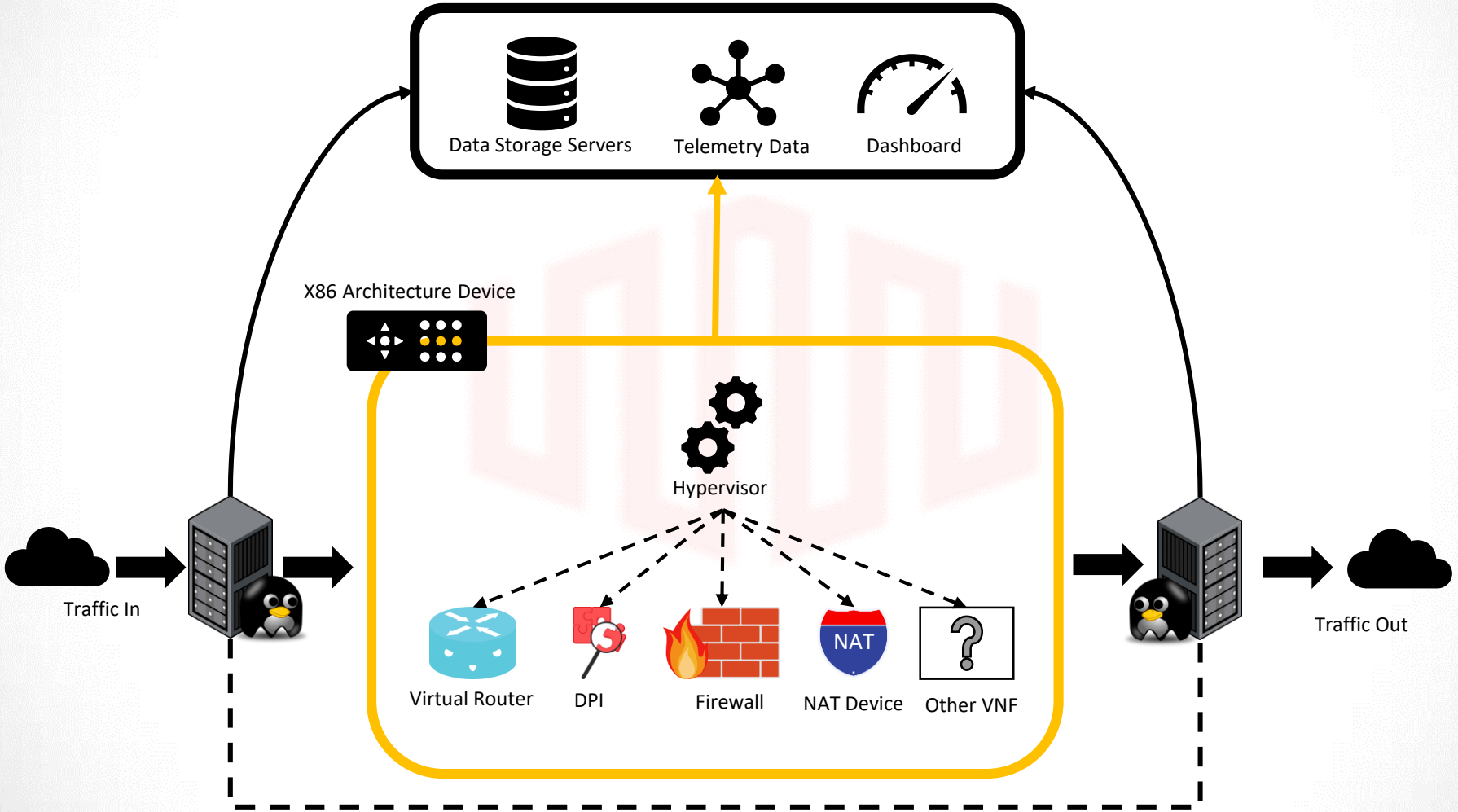
- Service chaining - set of network functions connected to support an application.
- SDN/NFV facilitates the ease of provisioning and reconfiguring the service chains.
- Building a service chain using SDN/NFV eliminates the need of acquiring network hardware.



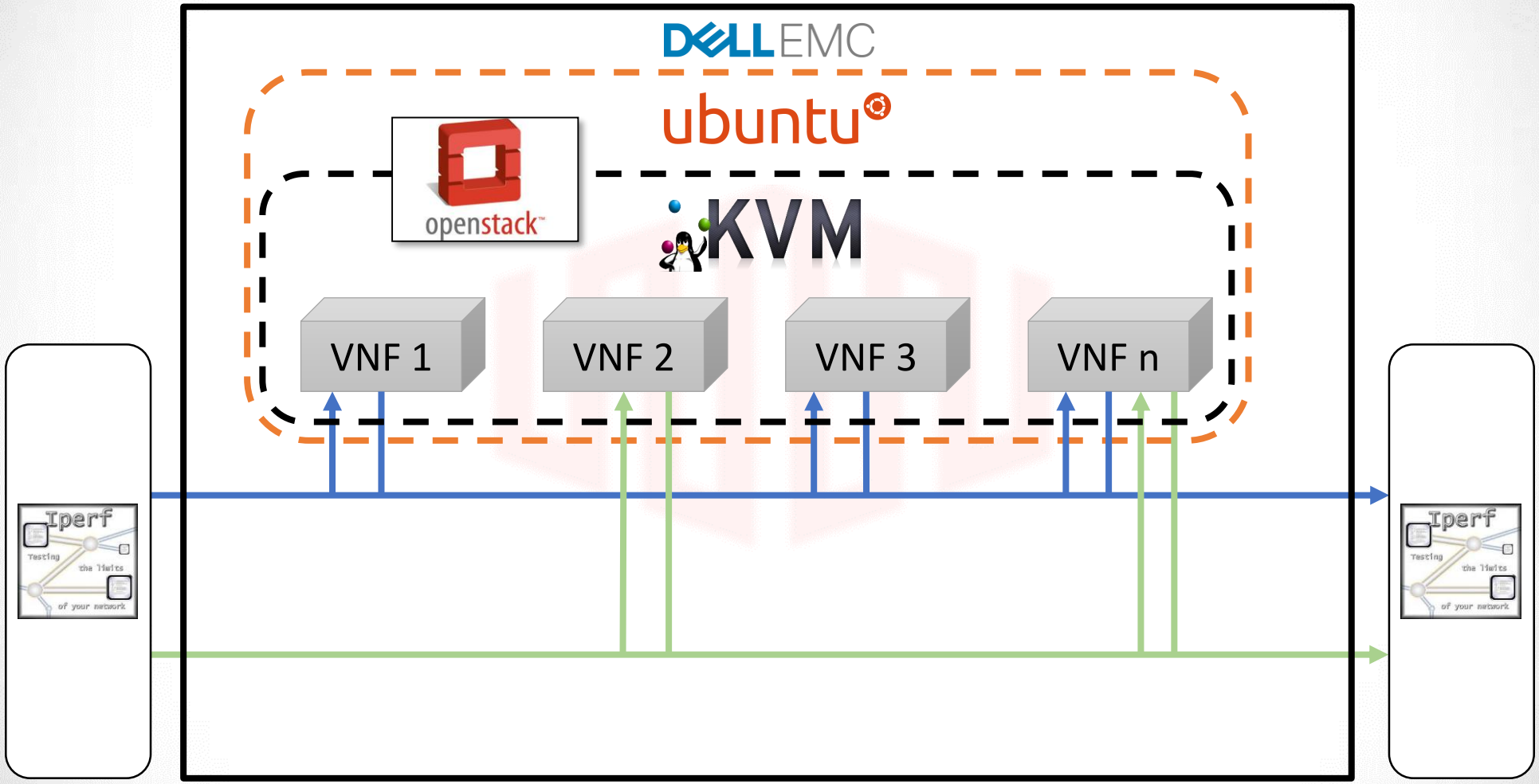
Objectives

- Creation of various combinations of service chains – using VNFs from vendors and open source services.
- Creation of test cases to test throughput and performance of the service chains.
- Subject the chains to undergo varying types of traffic.
- Carry out testing in a consistent environment.
- Creation of an abstraction layer to plug-in and test.
- Creation of a dashboard for performance monitoring.
- Store the performance related data in a database.

Concept of Operations

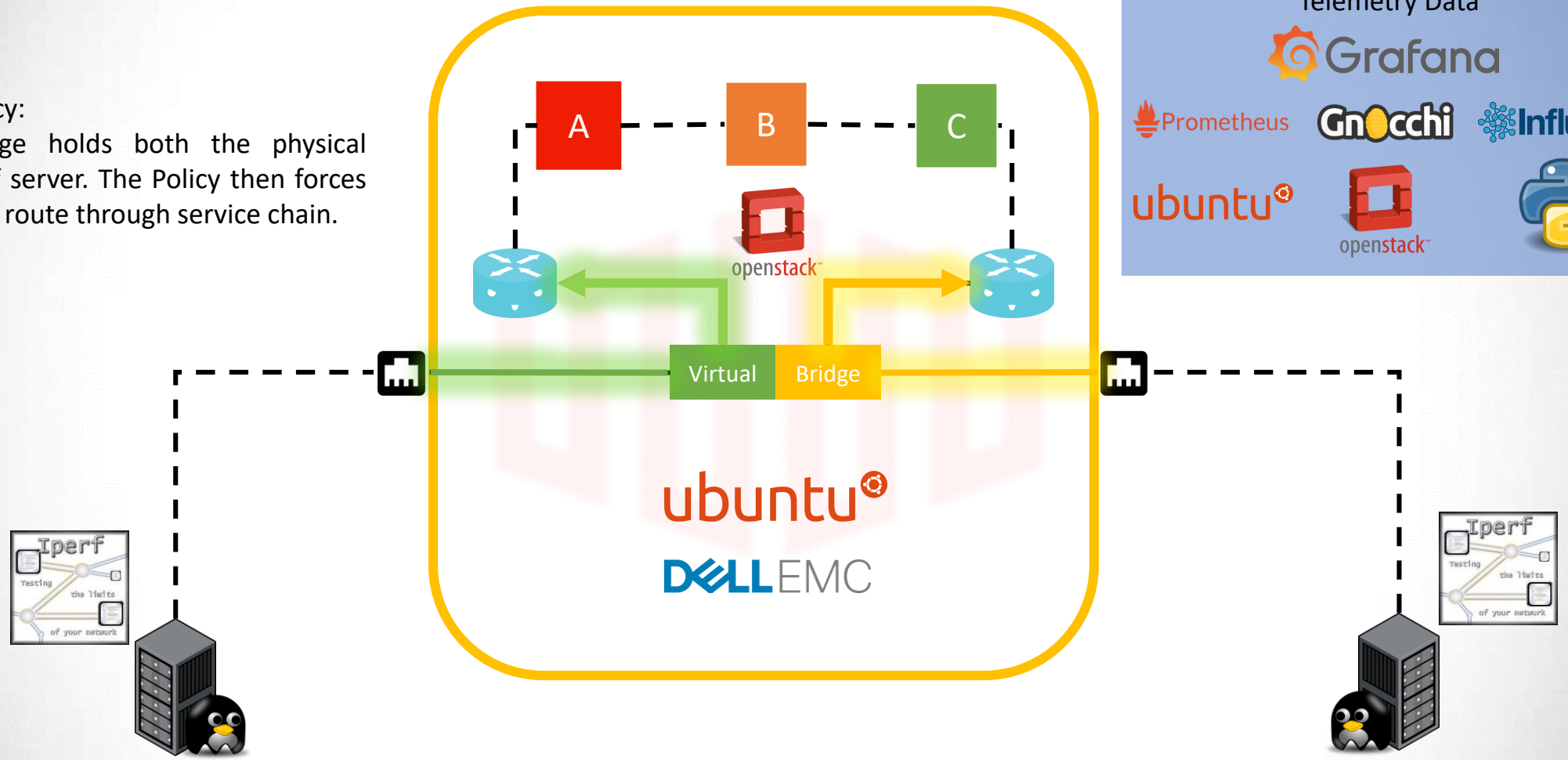


Functional Block Diagram



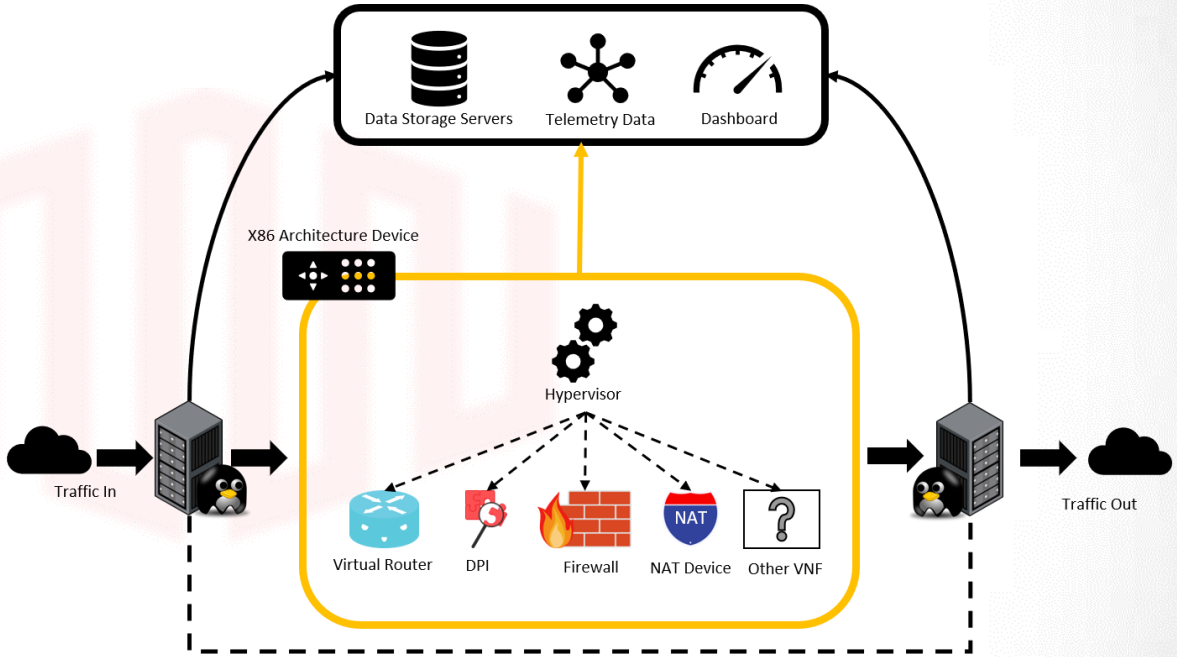
Project Implementation

Routing Policy:
Virtual bridge holds both the physical interfaces of server. The Policy then forces the traffic to route through service chain.



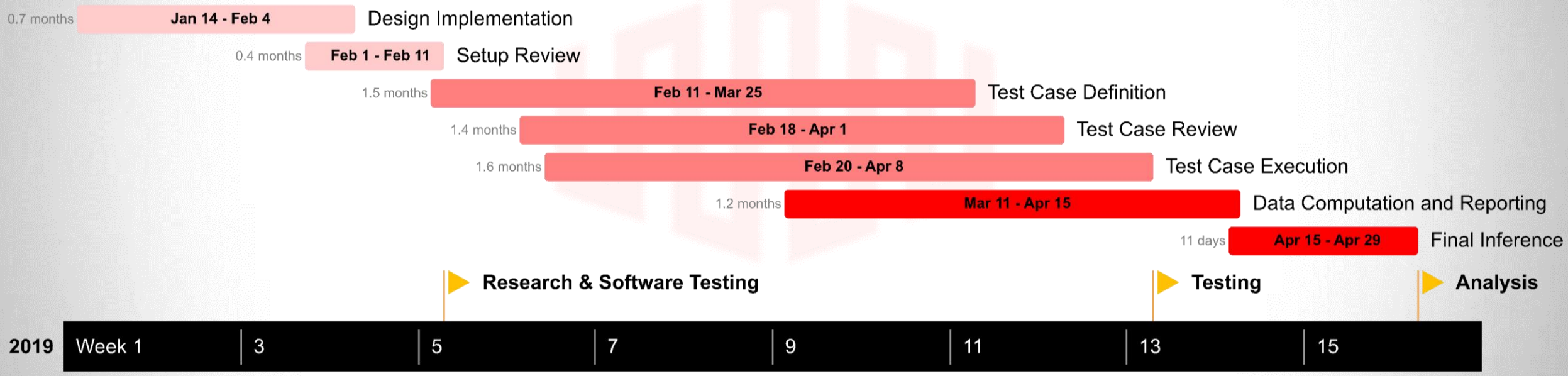
Critical Project Elements

DES.1.1	• VNF Deployment
DES.1.2	• Use of OpenStack and KVM Hypervisor
DES.1.3	• x86 Architecture Device
DES.2.1	• Consistent test environment
DES.2.2	• Testing using traffic generator

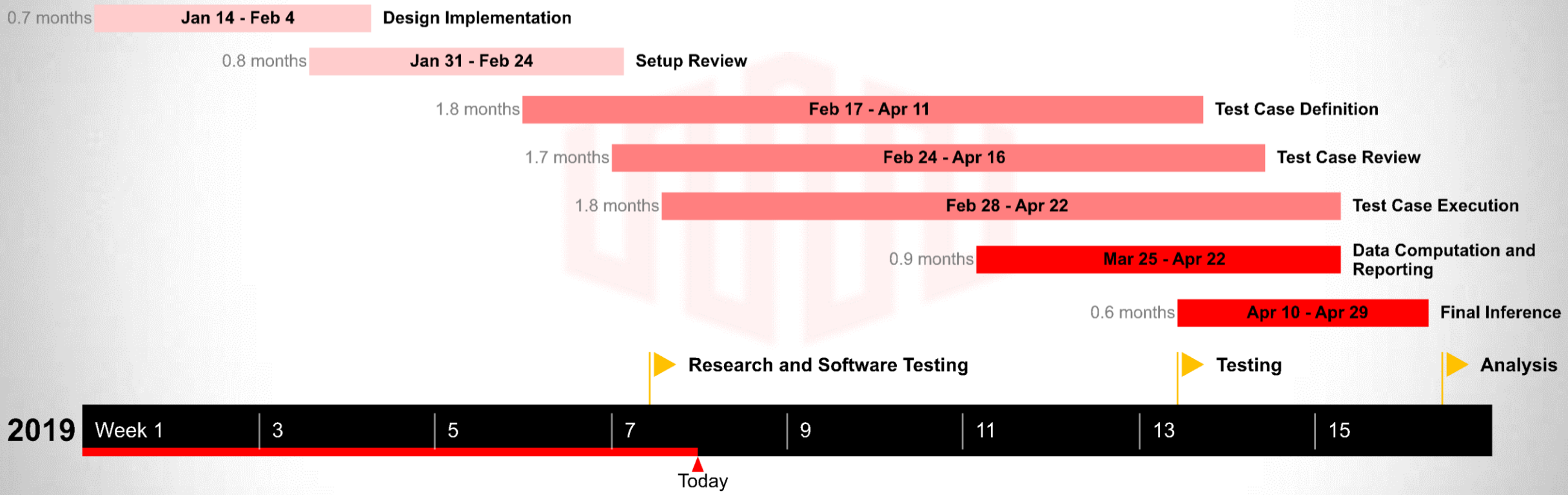


Schedule

Work Plan - Previous



Work Plan - Current



Status Report

Job Title	Description
Design Implementation	Completed – OpenStack Monitoring server and Iperf devices setup.
Setup Review	Completed – Service chain design confirmed with customer.
Test Case Definition	In Progress – Service Chain templates designed, subject to rework.
Test Case Review	In Progress – Basic service chain tested.
Test Case Execution	In Progress – Throughput data available for service chain tested.
Data Computation and Reporting	Pending.
Final Inference	Pending.

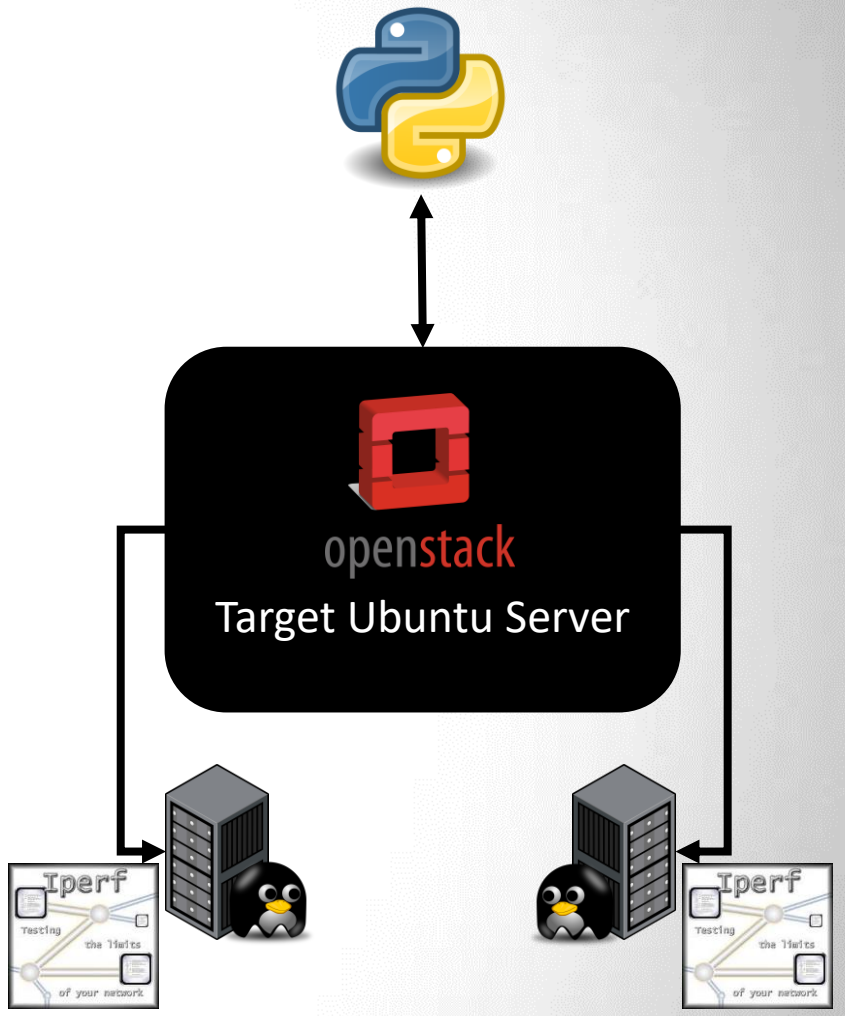
Test Readiness

Test 1: Infrastructure Setup

1 2 3 4

Verify working of hardware infrastructure

1. Check if server is powered on
2. Ping the VPN address of the server to check if there is VPN connectivity
3. SSH into the server hosting OpenStack
4. Check IP addresses of client and server machines and if they are able to ping the OpenStack server
5. Check working of OpenStack and current state of the server by running the script
6. Details provided by script are:
    I. CPU specification
    II. Memory utilization
    III. Disk usage
    IV. Interface status and configuration

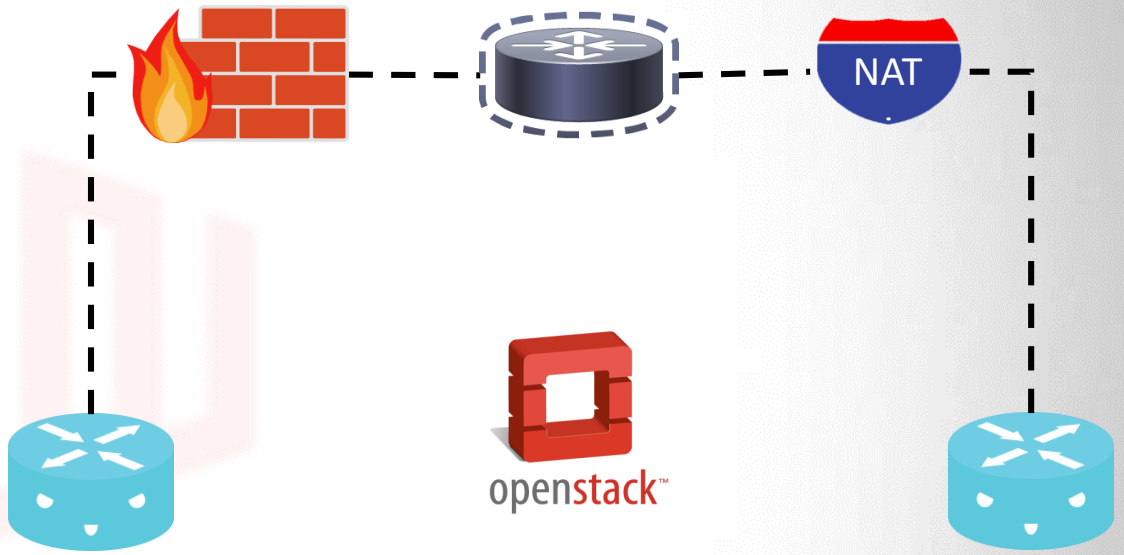


Test 2: Implement service chain

1 2 3 4

Service chain example : Firewall - Router - NAT

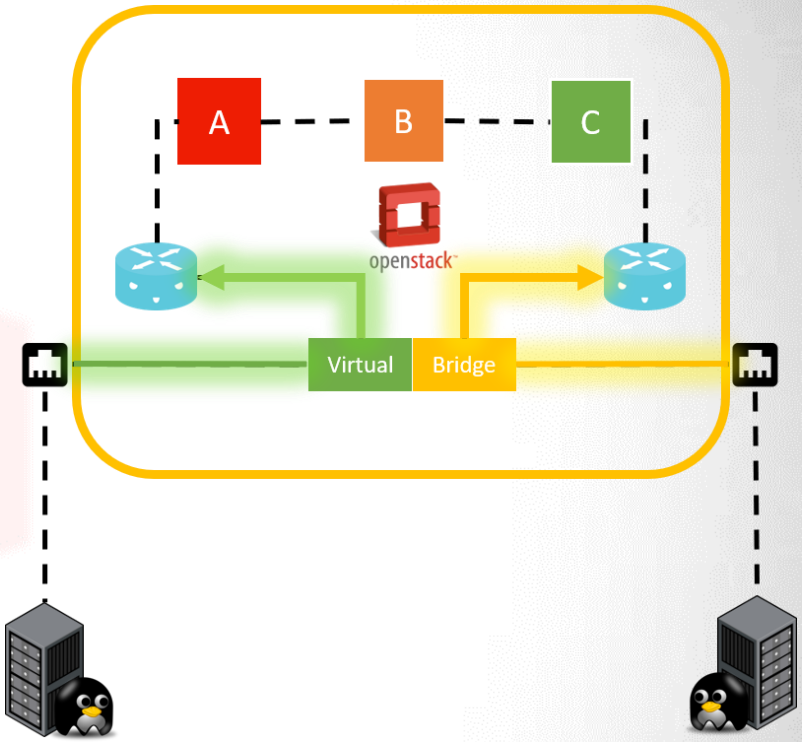
1. Use firewall to allow traffic from client and server machine IPs.
2. Router will route the packets to an "internal" IP address range.
3. NAT VM will destination NAT "internal" IP to the server machine IP.
4. Verify firewall rules and NAT translation in VNFs



Test 2: Implement service chain

1 2 3 4

1. Check if the service chain is set up on the OpenStack dashboard.
2. Confirm routing policies and static routes to support traffic flow.
3. Check ubuntu server routing policy and verify if the traffic is directed to vRouter.
4. Ping from client to server machine to check connectivity across service chain.
5. Run a traceroute to check whether traffic is being routed through the service chain.



Test 3: Metric Collection

1 2 3 4

1. Check node exporter and Grafana on the ubuntu server. Benchmark statistics.
2. Check node exporter and influx DB on client and server machines to check ping latency and traceroute
3. Check VM statistics using utilities like nova/ceilometer
4. Run iPerf3 client and server applications
5. Monitor statistics of the server using node exporter
6. Check throughput of the link using iperf3 with varying parameters:
    a. TCP traffic
    b. UDP traffic
    c. Varying amount of data
    d. Multiple parallel streams
    e. Different window sizes of TCP
    f. Reverse mode
7. Write the throughput information from iperf to influx DB and link it to Grafana.
8. Store metric data on local device for future analysis.

Grafana Prometheus Gnocchi InfluxDB ubuntu openstack

Test 4: Reproducibility

1 2 3 4

- To confirm reproducibility of the test cases, reproduce server configuration and service chain in identical server.
- Run the same test case against both servers to verify output.
- If the captured metrics are within 5% margin error, then we can ascertain that the tests are successful

Parts & Procurement

Server

Feature	Specifications
Processor	Intel® Xeon® processor E5-2600 v4 product family
Memory	64GB
Storage	500GB SATA
Networking	4x1Gb Ethernet NIC's
Quantity	2
Procured From	Telecom Lab

POWEREDGE R430

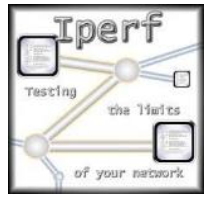
DELL EMC

Images

Product	Description	Cost
Juniper - vSRX 1G	Firewall	License - Equinix
Cisco – CSR 1G	Router	License - Equinix
Cirros	Linux Machine	Opensource
Ubuntu [16/18] – Cloud	Linux Machine	Opensource
Pfsense	Firewall	Opensource
VYOS	Router/NAT	Opensource

Software

Product	Description	Cost
Ubuntu 16.04.6 LTS OS	Operating System	Opensource
OpenStack (Rocky/Ocata)	Cloud Orchestrator	Opensource
Iperf	Traffic Generator	Opensource
Grafana	Dashboard	Opensource
Gnocchi, Influxdb, Prometheus	Database Source	Opensource



FIN/ACK?

Appendix

Test 1 : Infrastructure Setup

1 2 3 4

The python scrip runs on any machine having reachability to servers network. Following can be seen from the scripts output.

1. Ubuntu Version
2. Architectural information of server.
3. Server Time
4. CPU Statistics.
5. Memory Usage.
6. Disk Space.
7. Interface Details.
8. Iperf server and client reachability.
9. OpenStack Status on server and images available.

```
==========UBUNTU DETAILS==========
The Ubuntu release on the server is: 16.04.

==========CPU INFORMATION==========
The CPU Architecture: x86_64
The CPU modes available on the server: 32-bit, 64-bit
Number of CPU cores in the server: 12
Processor actual speed: 1371.812 MHz
Processor maximum speed: 1600.0000 MHz
Processor minimum speed: 1200.0000 MHz

==========CURRENT SERVER TIME==========
22:28:10 MST

==========CPU STATISTICS==========
Server CPU is 81% idle.

==========MEMORY USAGE==========
Total memory available: 62G
Used memory: 16G
Free memory: 44G

==========DISK SPACE==========
Details of disk /dev/sda1:
Total size: 458G
Available space: 393G

==========INTERFACE DETAILS==========
Details of the interface eno1:
IP Address: 172.16.218.10 255.255.0.0

Details of the interface eno3:
IP Address: 1.1.1.100 255.255.255.0

Details of the interface eno4:
IP Address: 2.2.2.100 255.255.255.0

==========IPERF SERVER AND CLIENT REACHABILITY==========
iPerf Client at 1.1.1.1 is reachable.
iPerf Server at 2.2.2.2 is reachable.

==========OPENSTACK OPERATION CONFIRMATION==========
Images in OpenStack using the command 'openstack image list':
+--------------------------------------+--------------------------+--------+
| ID                                   | Name                     | Status |
+--------------------------------------+--------------------------+--------+
| 38660642-882e-4828-bf8d-101823098873 | cirros-0.3.5-x86_64-disk | active |
| 4b3ad6e3-61ed-4a64-b313-7c28202aed2e | csr_image                | active |
| 02f23e98-a123-4575-bb0e-795698d1e579 | vsrx_juniper             | active |
+--------------------------------------+--------------------------+--------+
```

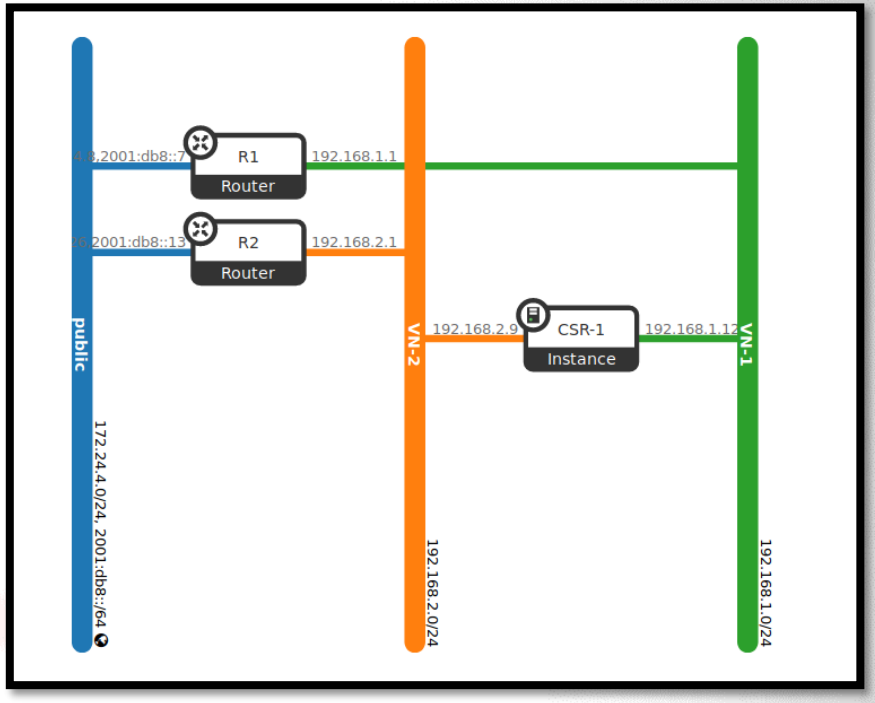
Appendix

Test 2 : Service Chain

1 2 3 4

Checks in Service chain:

1. The network diagram on OpenStack shows the network after issuing of the traceroutes from one Iperf server to the other.
2. The traceroute is done at Client 1.1.1.1 destined for 2.2.2.2.
3. It can be seen from the traceroute that traffic first passes through the OpenStack running one CSR instance before reaching the Iperf server.




```
t9eqx@t9eqx:~$ traceroute 2.2.2.2
traceroute to 2.2.2.2 (2.2.2.2), 64 hops max
  1   1.1.1.100  0.238ms  0.195ms  0.143ms
  2   172.24.4.8  0.622ms  0.176ms  0.138ms
  3   192.168.1.12  1.547ms  0.768ms  0.682ms
  4   192.168.2.1  0.961ms  0.556ms  0.595ms
  5   1.1.1.100  2.254ms  0.440ms  0.464ms
  6   2.2.2.2  1.032ms  0.835ms  1.479ms
```

Appendix

Test 2 : Service Chain

1 2 3 4

Checks in Service chain:

Throughput loss while routing traffic through Cisco CSR.

```
t9eqx@t9eqx:~$ iperf3 -c 2.2.2.2
Connecting to host 2.2.2.2, port 5201
[  4] local 1.1.1.1 port 53764 connected to 2.2.2.2 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  99.0 MBytes   830 Mbits/sec    0   2.98 MBytes
[  4]   1.00-2.00   sec   104 MBytes   870 Mbits/sec    0   2.98 MBytes
[  4]   2.00-3.00   sec   105 MBytes   881 Mbits/sec    0   2.98 MBytes
[  4]   3.00-4.00   sec   104 MBytes   870 Mbits/sec    0   2.98 MBytes
[  4]   4.00-5.00   sec   104 MBytes   870 Mbits/sec    0   2.98 MBytes
[  4]   5.00-6.00   sec   102 MBytes   860 Mbits/sec    0   2.98 MBytes
[  4]   6.00-7.00   sec   102 MBytes   860 Mbits/sec    0   2.98 MBytes
[  4]   7.00-8.00   sec  95.0 MBytes   797 Mbits/sec    0   2.98 MBytes
[  4]   8.00-9.00   sec   102 MBytes   860 Mbits/sec    0   2.98 MBytes
[  4]   9.00-10.00  sec   102 MBytes   860 Mbits/sec    0   2.98 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1020 MBytes   856 Mbits/sec    0             sender
[  4]   0.00-10.00  sec  1019 MBytes   855 Mbits/sec                  receiver
```

Checks in Service chain:

Throughput while routing traffic through Cirros VM

```
t9eqx@t9eqx:~$ iperf3 -c 2.2.2.2
Connecting to host 2.2.2.2, port 5201
[  4] local 1.1.1.1 port 53748 connected to 2.2.2.2 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   114 MBytes   956 Mbits/sec    0    542 KBytes
[  4]   1.00-2.00   sec   112 MBytes   940 Mbits/sec    0    597 KBytes
[  4]   2.00-3.00   sec   111 MBytes   929 Mbits/sec    0    796 KBytes
[  4]   3.00-4.00   sec   112 MBytes   944 Mbits/sec    0    932 KBytes
[  4]   4.00-5.00   sec   111 MBytes   933 Mbits/sec    0   1.07 MBytes
[  4]   5.00-6.00   sec   111 MBytes   933 Mbits/sec    0   1.12 MBytes
[  4]   6.00-7.00   sec   112 MBytes   944 Mbits/sec    0   1.12 MBytes
[  4]   7.00-8.00   sec   111 MBytes   933 Mbits/sec    0   1.12 MBytes
[  4]   8.00-9.00   sec   112 MBytes   944 Mbits/sec    0   1.25 MBytes
[  4]   9.00-10.00  sec   110 MBytes   923 Mbits/sec    0   1.43 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  1.09 GBytes   938 Mbits/sec    0             sender
[  4]   0.00-10.00  sec  1.09 GBytes   936 Mbits/sec                  receiver
```

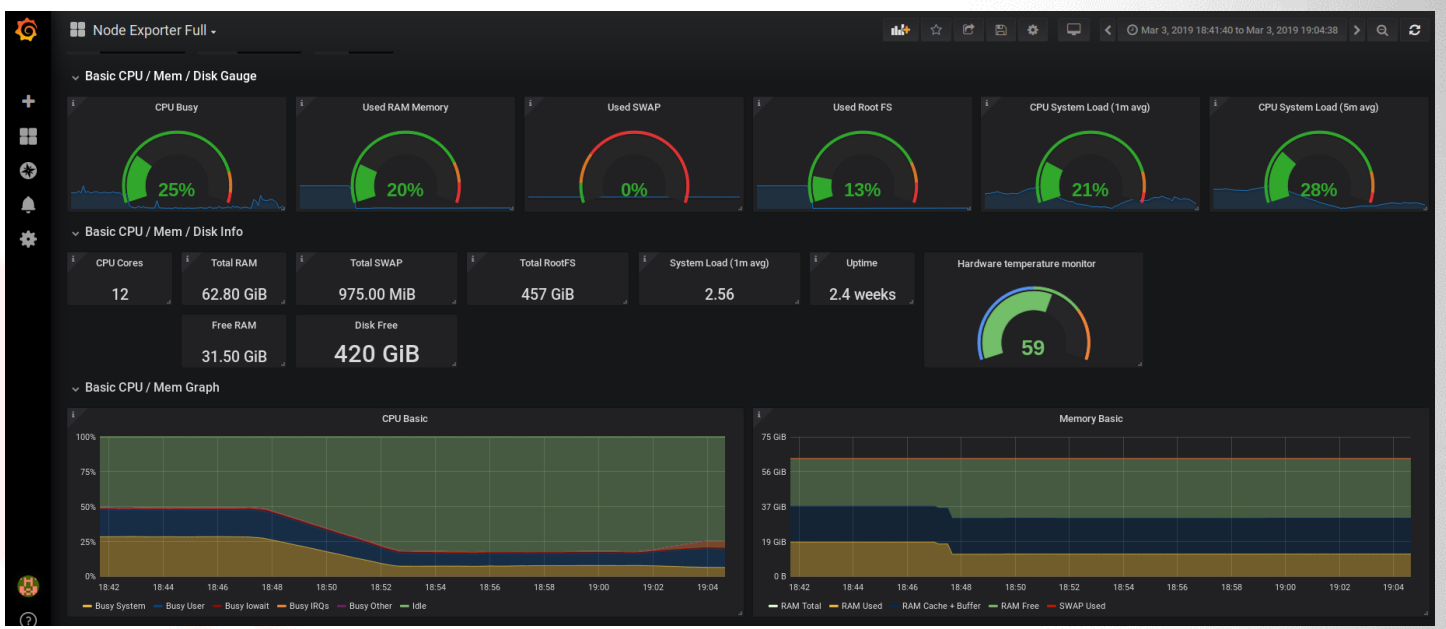
Appendix

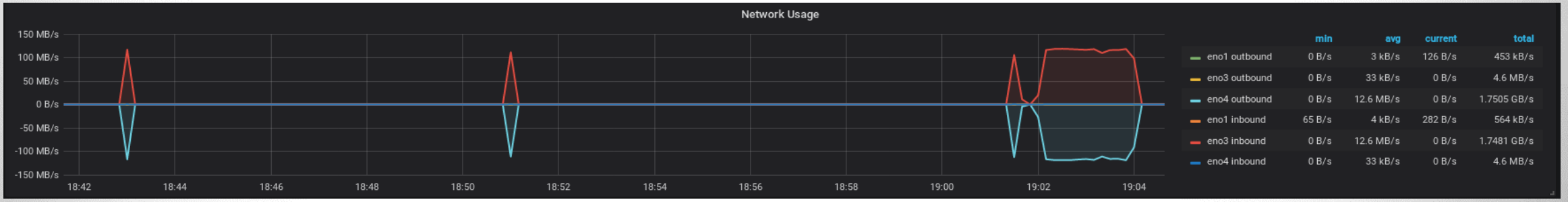
Test 3: Metric Collection

1 2 3 4

Server Metrics has been polled from Prometheus Node Exporter.
We are concerned with CPU, Memory and Network Statistics.
There is a list of metrics available at:
https://github.com/prometheus/node_exporter

We have generated a standard dashboard for these metrics on Grafana – Screenshots as seen.



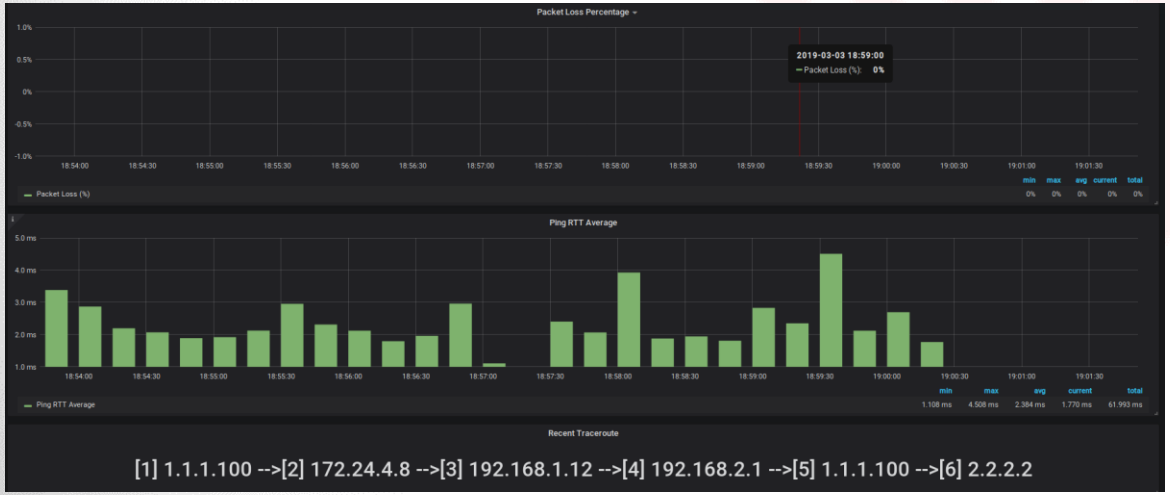


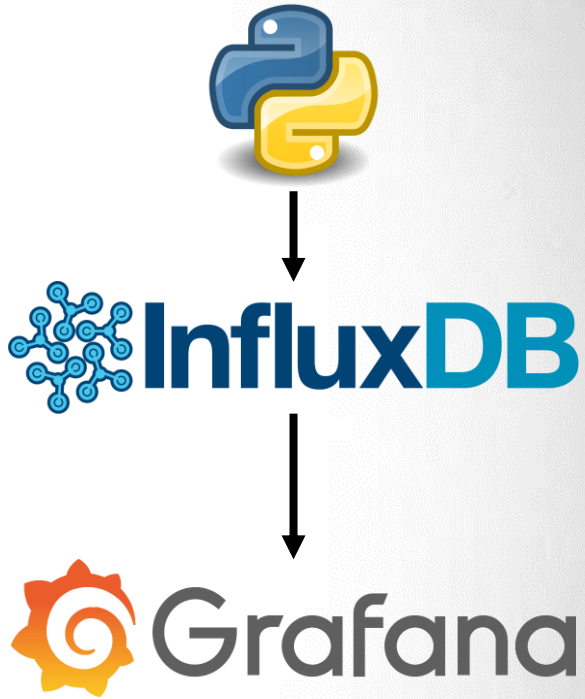
Appendix

Test 3: Metric Collection

1 2 3 4

- Ping/Traceroute from source traffic generator to the server via the service chain.
- Capture RTT and Hops.
- Also confirms traffic flow through service chain.





Appendix