- Detailed description of your cloud benchmarking methodology, including any scripts or other code

1) Querying the list of your VMs

For this scenario we executed Nova_list_servers script which takes time and concurrency as parameters. For obtaining a better average we gave the following to time and concurrency:

Time: 3

Concurrency: 10

To calibrate the results we checked it against the OpenStack command "openstack server list" and got equivalent performance in ms.

Script: NovaServers.list_servers.JSON

```
{
  "NovaServers.list_servers": [
    {
      "args": {
        "detailed": true
      },
      "context": {},
      "hooks": [],
      "runner": {
        "concurrency": 3,
        "times": 10,
        "type": "constant"
      },
      "sla": {}
    }
  ]
}
```

2) Creation Time of VMs (including the boot time)

For cloud benchmarking of creation/boot scenario we have used "NovaServers.boot_server" script as example. This benchmark scenario allows to test with different parameters, image and flavors.

The fields we used includes

Runner: The type selected here is 'constant' since the scenario required on different time intervals at particular times of a day and we had limited number of instances that could be initialized.

Time: The number of times iteration has to be executed for benchmarking. We have kept this value as **1** to allow concurrent execution of boot scripts keeping in view limited quota

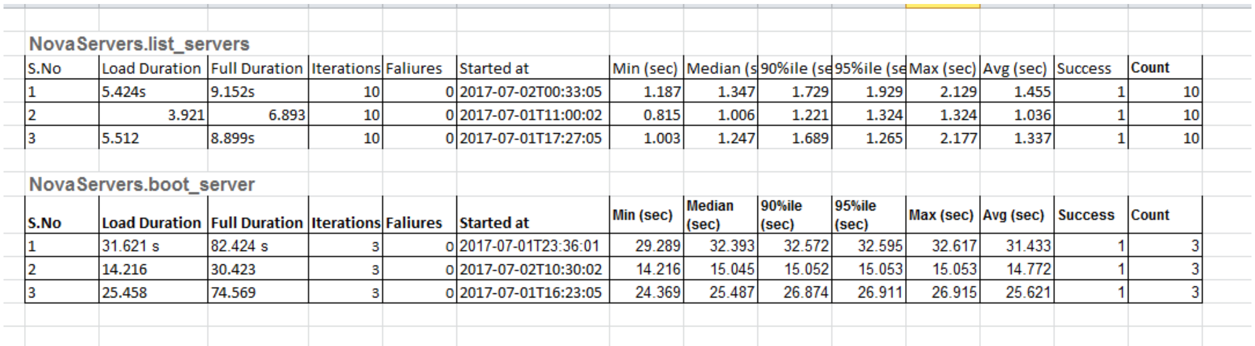
Concurrency: The number of times the script executes in parallel. We gave it value **3.**

Checking concurrent initiation/boot of our VMs is the most important factor in cloud benchmarking. This was tested in three different time slots. Apart from some observed downtime during night time, the difference in performance results is negligible.

Script: NovaServers.boot_server.JSON

```
{
  "NovaServers.boot_server": [
    {
      "args": {
        "flavor": {
          "name": "Cloud Computing"
        },
        "image": {
          "name": "ubuntu-16.04"
        }
      },
      "context": {},
      "hooks": [],
      "runner": {
        "concurrency": 3,
        "times": 3,
        "type": "constant"
      },
      "sla": {}
    }
  ]
}
```

- Benchmarking results of six different scenarios and time slots, including plots and interpretation of results



NovaServers.list_servers

S.No	Load Duration	Full Duration	Iterations	Faliures	Started at	Min (sec)	Median (s	90%ile (se	95%ile (se	Max (sec)	Avg (sec)	Success	Count
1	5.424s	9.152s	10	0	2017-07-02T00:33:05	1.187	1.347	1.729	1.929	2.129	1.455	1	10
2	3.921	6.893	10	0	2017-07-01T11:00:02	0.815	1.006	1.221	1.324	1.324	1.036	1	10
3	5.512	8.899s	10	0	2017-07-01T17:27:05	1.003	1.247	1.689	1.265	2.177	1.337	1	10

NovaServers.boot_server

S.No	Load Duration	Full Duration	Iterations	Faliures	Started at	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
1	31.621 s	82.424 s	3	0	2017-07-01T23:36:01	29.289	32.393	32.572	32.595	32.617	31.433	1	3
2	14.216	30.423	3	0	2017-07-02T10:30:02	14.216	15.045	15.052	15.053	15.053	14.772	1	3
3	25.458	74.569	3	0	2017-07-01T16:23:05	24.369	25.487	26.874	26.911	26.915	25.621	1	3

From this we can visibly see that during the morning and evening openstack work better than in the night.

- Commented listing of commands you executed for Task 2

```
//Task-1

//Loading environment variables
source openrc

//Creating rally openstack deployment


//Task-2

//Loading environment variables
source openrc

//Instantiate server.yaml file to start instance passing all the agruments
as parameters
openstack stack create --template server.yaml/ --parameter
"name=Server;image=ubuntu-16.04;flavor=Cloud Computing;key_pair=Cloud-
Computing;network=tu-internal;zone=Cloud Computing 2017" Mystack

//checking the status of the instance
openstack stack list

//Creating the floating ip for assigning it to the server.
openstack floating ip create tu-internal

//Assigning the flaoting ip to the server
openstack server add floating ip Server 10.200.1.154

//Adding ICMP and SSH rule so that the created VM can be pinged and
connect to SSH
openstack security group rule create default --protocol tcp --dst-port
22:22 --remote-ip 0.0.0.0/0
openstack security group rule create --protocol icmp default

//Ping the VM
ping 10.200.1.154

//Connect the VM through SSH
sudo ssh -i Cloud-Computing ubuntu@10.200.1.154

//Check the  internet connectivity of the VM
ping google.com


//Delete the stack
openstack stack delete Mystack
```

```
//Check the Stack have been deleted
openstack stack list
```

- The contents of your server-landscape.yml file

```
heat_template_version: 2015-10-15

description: Create a new neutron network plus a router to the public
  network, and for deploying one frontend servers and two backend server into the new network. The template also
  assigns floating IP addresses to the frontend server  so they are routable from the
  public network. and also assigning custom security groups to the front end server.


parameters:
  name:
    type: string
    label: Name of the VM
  key_pair:
    type: string
    label: Key Pair
    constraints:
      - custom_constraint: nova.keypair
  flavor:
    type: string
    label: Flavor
    constraints:
      - custom_constraint: nova.flavor
  image:
```

```
  type: string
  label: Image Name
  constraints:
    - custom_constraint: glance.image
# network:
#   type: string
#   label: Network
#   constraints:
#     - custom_constraint: neutron.network
zone:
  type: string
  label: Availability Zone
  default: Default
security_groups:
  type: comma_delimited_list
  label: Security Group(s)
  default: "default"
cidr_private:
  type: string
  label: cidr private
gateway_ip:
  type: string
  label: gateway_ip
networkcloud:
  type: string
```

```
    label: networkcloud
  public_network:
    type: string
    label: public_network


resources:

  # This port is a separate resource used to assign the security groups
  # to the VM. Can also be used to attach a OS::Neutron::FloatingIP to the VM.
  instance:
    type: server.yaml
    properties:
      name: { get_param: name }
      key_pair: { get_param: key_pair }
      image: { get_param: image }
      flavor: { get_param: flavor }
      zone: { get_param: zone }
      network: { get_attr: [private_subnet, network_id]}
      #subnet: {get_resource: private_subnet}
      security_groups:
       - default
       - {get_resource : secgroup}
```

```
my_backend_group:
  type: OS::Heat::ResourceGroup
  properties:
    count: 2
    resource_def:
      type: server.yaml
      properties:
        name: backend-%index%
        key_pair: { get_param: key_pair }
        image: { get_param: image }
        flavor: { get_param: flavor }
        zone: { get_param: zone }
        network: { get_attr: [private_subnet, network_id]}
        #subnet: {get_resource: private_subnet}


secgroup:
  type: OS::Neutron::SecurityGroup
  properties:
    rules:
      - protocol: tcp
        remote_ip_prefix: 0.0.0.0/0
        port_range_min: 80
        port_range_max: 80
```

```
   - protocol: icmp
   - protocol: tcp
    port_range_min: 22
    port_range_max: 22

private_net:
  type: OS::Neutron::Net
  properties:
    name: {get_param: networkcloud}

private_subnet:
  type: OS::Neutron::Subnet
  properties:
    network_id : { get_resource : private_net }
    cidr:  {get_param: cidr_private}
    gateway_ip: {get_param: gateway_ip}

router:
  type: OS::Neutron::Router
  properties:
    external_gateway_info:
      network: {get_param : public_network}

router_interface:
  type: OS::Neutron::RouterInterface
```

```
    properties:
      router_id : {get_resource: router}
      subnet_id : {get_resource: private_subnet}


  floating_ip:
   type: OS::Neutron::FloatingIP
   properties:
    floating_network: { get_param: public_network }

  floating_ip_assoc:
   type: OS::Neutron::FloatingIPAssociation
   properties:
    floatingip_id: { get_resource: floating_ip }
    port_id: { get_attr: [instance,port] }


outputs:
 test_out:
  description: Value of server id
  value: {get_attr: [instance,server]}
 details:
  description: Details of server
  value: {get_attr: [instance]}
 floating ip:
```

description: Floating IP of the instance

value: { get_attr: [floating_ip, floating_ip_address] }

- Commented listing of commands you executed to test your advanced Heat template

```
//Task-3

//Loading environment variables
source openrc

//Instantiate server-landscape.yaml file to start instance passing all the
agruments as parameters
openstack stack create --template A2/server-lanscape.yaml/ --parameter
"name=frontend;image=ubuntu-16.04;flavor=Cloud Computing;key_pair=Cloud-
Computing;cidr_private=10.12.1.0/24;gateway_ip=10.12.1.1;zone=Cloud
Computing 2017;public_network=tu-internal;networkcloud=Cloud-network"
Mystack

//checking the status of the instance
openstack stack list

//Extracting the floating ip of the Front-end server
openstack stack show mystack

//Pinging the VM
ping 10.200.1.163

//Transfer the private key for accessing back-end server from the frontend
servers
sudo scp -i Cloud-Computing Cloud-Computing ubuntu@10.200.1.163:

//Connect the frontend through SSH
sudo ssh -i Cloud-Computing ubuntu@10.200.1.163

//Check the  internet connectivity of the VM
ping google.com

//Connect the backend server through SSH from frontend server
sudo ssh -i Cloud-Computing ubuntu@10.12.1.5

//Check the  internet connectivity of the VM
ping google.

//Exit the backend server
exit

//Connect the backend server through SSH from frontend server
sudo ssh -i Cloud-Computing ubuntu@10.12.1.4

//Check the  internet connectivity of the VM
ping google.com
```

```
//Exit the backend server
exit

//Deleting the stack
openstack stack delete Mystack

//Check for the Stack have been deleted
openstack stack list
```