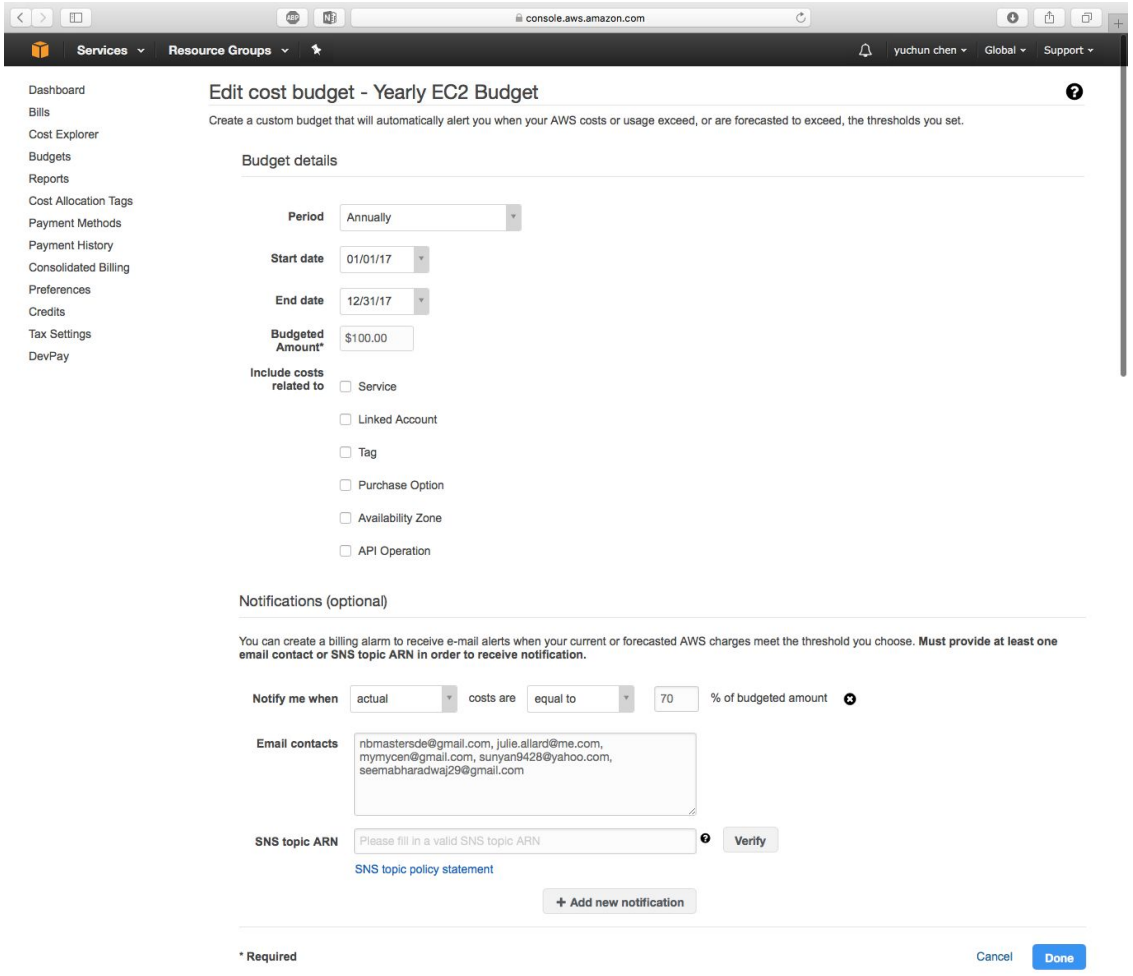
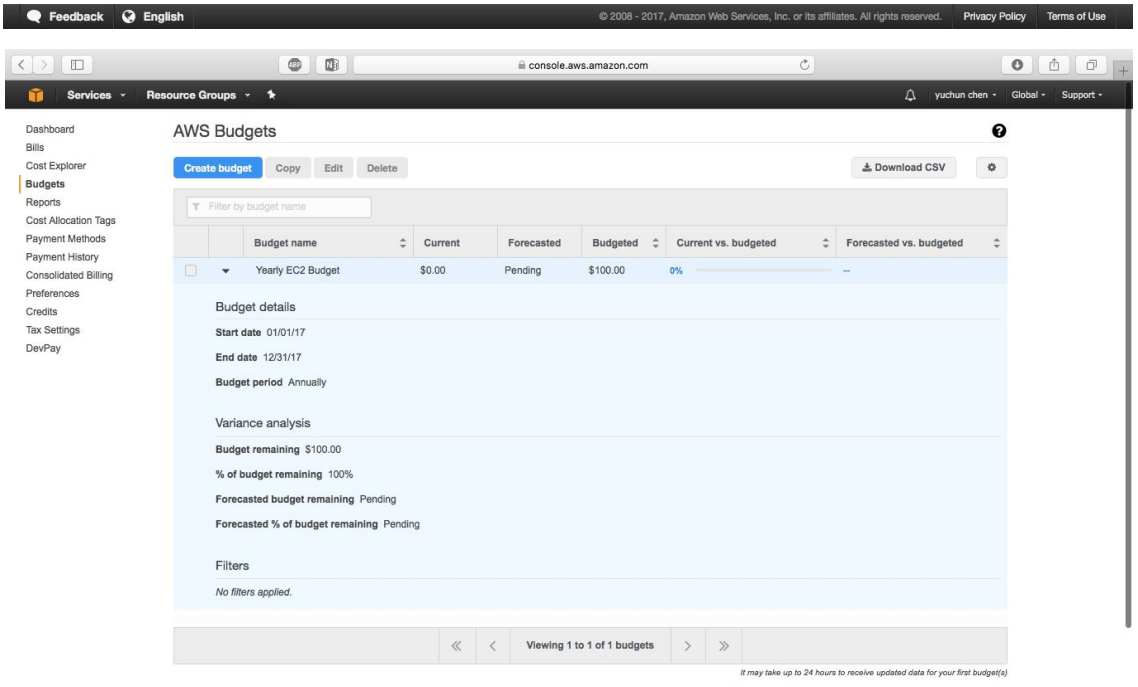
Cloud Computing

SS17 | Assignment 1

Group 2
Yan Sun
Yuchun Chen
Seema Narasimha Swamy
Julie Allard
Navya Basavaraju

Budget creation of $100 in AWS and notification after using 70% of our budget





Preparing an AWS EC2 Instance

In terminal, configure the settings that the AWS CLI uses when interacting with AWS:

```
~ ❯❯❯ aws configure
AWS Access Key ID [None]: MYACCESSKEYID
AWS Secret Access Key [None]: MYSECRETACCESSKEY
Default region name [None]: eu-central-1
Default output format [None]:
```

Create a new security group:

```
~ ❯❯❯ aws ec2 create-security-group --group-name devenv-sg --description "security group for development environment in EC2"
{
    "GroupId": "sg-2b315240"
}
~ ❯❯❯ aws ec2 authorize-security-group-ingress --group-name devenv-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
```

Create a key pair:

```
~ ❯❯❯ aws ec2 create-key-pair --key-name devenv-key --query 'KeyMaterial' --output text > devenv-key.pem
```

Change the file mode of the key pair file so only we have access to the key file:

```
~ ❯❯❯ chmod 400 devenv-key.pem
```

Launch the AWS instance:

```
~ ❯❯❯ aws ec2 run-instances --image-id ami-f52bfa9a  --security-group-ids sg-2b315240 --count 1 --instance-type m3.medium --key-name devenv-key --query 'Instances[0].InstanceId'
"i-044712fd1bfd298ea"
```

Query the public IP address that we will use to connect to the AWS instance:

```
~ ❯❯❯ aws ec2 describe-instances --instance-ids i-041d6b6a2cd5960c2 --query 'Reservations[0].Instances[0].PublicIpAddress'
"52.29.44.172"
```

Connect to the AWS instance and accept the server's public key to complete the connection:

```
~ ❯❯❯ ssh -i devenv-key.pem ec2-user@52.29.44.172

The authenticity of host '52.28.94.19 (52.28.94.19)' can't be established.
ECDSA key fingerprint is SHA256:sWrb/h0v9IFkK57TvyZqNl03lo24Afq88EZSRu+2G9M.
Are you sure you want to continue connecting (yes/no)? yes

       __|  __|_  )
```

```
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.03-release-notes/
7 package(s) needed for security, out of 11 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-20-224 ~]$
```

Install gcc and fio, as dd is already installed default on Linux:

```
[ec2-user@ip-172-31-20-224 ~]$ sudo yum groupinstall "Development Tools"
LONG OUTPUT
[ec2-user@ip-172-31-20-224 ~]$ sudo yum install -y fio
Loaded plugins: priorities, update-motd, upgrade-helper
Resolving Dependencies
--> Running transaction check
---> Package fio.x86_64 0:2.1.5-1.5.amzn1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
================================================================================
=========================================
 Package                 Arch                    Version
Repository                Size
================================================================================
================================================================================
=========================================
Installing:
 fio                     x86_64                  2.1.5-1.5.amzn1
amzn-main                 287 k

Transaction Summary
================================================================================
================================================================================
=========================================
Install  1 Package

Total download size: 287 k
Installed size: 1.2 M
Downloading packages:
fio-2.1.5-1.5.amzn1.x86_64.rpm
| 287 kB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : fio-2.1.5-1.5.amzn1.x86_64
1/1
  Verifying  : fio-2.1.5-1.5.amzn1.x86_64
1/1

Installed:
```

```
  fio.x86_64 0:2.1.5-1.5.amzn1

Complete!
```

Download the RPM file to install the EC2 API tool, with Ruby which is already installed:

```
[ec2-user@ip-172-31-20-224 ~]$ sudo wget
https://s3.amazonaws.com/ec2-downloads/ec2-ami-tools.noarch.rpm
--2017-06-08 15:39:30--  https://s3.amazonaws.com/ec2-downloads/ec2-ami-tools.noarch.rpm
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.224.67
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.224.67|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 201239 (197K) [application/x-rpm]
Saving to: 'ec2-ami-tools.noarch.rpm'

ec2-ami-tools.noarch.rpm
100%[=========================================================================
==============================>] 196.52K   500 KB/s    in 0.4s

2017-06-08 15:39:31 (500 KB/s) - 'ec2-ami-tools.noarch.rpm' saved [201239/201239]
```

Install the RPM:

```
[ec2-user@ip-172-31-20-224 ~]$ sudo yum install ec2-ami-tools.noarch.rpm
Loaded plugins: priorities, update-motd, upgrade-helper
Examining ec2-ami-tools.noarch.rpm: ec2-ami-tools-1.5-7.noarch
Marking ec2-ami-tools.noarch.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package ec2-ami-tools.noarch 0:1.5-7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
================================================================================
=========================================
 Package                 Arch                 Version                 Repository
Size
================================================================================
================================================================================
=========================================
Installing:
 ec2-ami-tools           noarch               1.5-7
/ec2-ami-tools.noarch                766 k

Transaction Summary
================================================================================
================================================================================
=========================================
Install  1 Package

Total size: 766 k
Installed size: 766 k
```

```
Is this ok [y/d/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : ec2-ami-tools-1.5-7.noarch
1/1
  Verifying  : ec2-ami-tools-1.5-7.noarch
1/1

Installed:
  ec2-ami-tools.noarch 0:1.5-7

Complete!
```

Verify the AMI tools installation:

```
[ec2-user@ip-172-31-20-224 ~]$ ec2-ami-tools-version
/usr/share/ruby/vendor_ruby/2.0/rubygems/core_ext/kernel_require.rb:55:in `require': cannot load such
file -- ec2/amitools/version (LoadError)
        from /usr/share/ruby/vendor_ruby/2.0/rubygems/core_ext/kernel_require.rb:55:in `require'
        from /usr/lib/ruby/site_ruby/ec2/amitools/showversion.rb:11:in `<main>'
```

Preparing a VM in OpenStack

Install the OpenStack client

The following example shows the command for installing the OpenStack client with pip(pip and python were already present in the local machine)

$pip install python-openstackclient

Download and source the OpenStack RC file

1. Log in to the dashboard and from the drop-down list select the project for which you want to download the OpenStack RC file.
2. On the Project tab, open the Compute tab and click Access & Security.
3. On the API Access tab, click Download OpenStack RC File and save the file. The filename will be of the form cc17-group02-openrc.sh of the project for which you downloaded the file.
4. On any shell from which you want to run OpenStack commands, source the cc17-group02-openrc.sh file for the respective project.

Sourcing the openrc file

$ source cc17-group02-openrc.sh

Please enter your OpenStack Password:

Gathering the parameters required to launch an instance

$ openstack image list

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

| ID | Name | Status |

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

| 77c18cca-2971-4cc0-b385-e2186e26 | CC17 Group2 Openstack Instance | active |

| 2813 | | |

| 69ca7dc6-51d9-43e1-bee4-68844498 | cirros | active |

| e202 | | |

| bc54114b-9fea-43af-81fd- | ubuntu-14-04 | active |

| ba449f29b766 | | |

| 11f6b8aa-31df-4b66-8b42-5ee9760c | ubuntu-16.04 | active |

| 47ba | | |

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

$ openstack flavor list

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

| ID | Name | RAM | Disk | Ephemeral | VCPUs | Is Public |

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

| 604de11c-32 | Cloud | 512 | 10 | 0 | 1 | False |

| 22-4902-852 | Computing | | | | | |

| 3-11cc61b5b | | | | | | |

| 485 | | | | | | |

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

$ openstack security group list

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

| ID | Name | Description | Project |

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

| a87e65e1-f959-4255-9 | default | Default security | 4894c882bfcc4558ab51 |

| b40-cafe1d9db47c | | group | 5cdc55b32446 |

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

$ openstack keypair list

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

| Name | Fingerprint |

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

| cc17g2os | 70:e5:87:c8:33:40:47:e5:99:f8:5f:de:2d:e2:0f:06 |

| cc17g2ssh | 28:d7:ad:56:3e:e5:90:f4:f6:ed:9a:3b:ba:c1:9f:e3 |

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

After you gather required parameters, run the following command to launch an instance. Specify the server name, flavor ID, and image ID

$openstack server create --flavor 604de11c-3222-4902-8523-11cc61b5b485 --image 11f6b8aa-31df-4b66-8b42-5ee9760c47ba --key-name cc17g2os --security-group default CC17GROUP2

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

| Field | Value |

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

| OS-DCF:diskConfig | MANUAL |

| OS-EXT-AZ:availability_zone | |

| OS-EXT-STS:power_state | NOSTATE |

| OS-EXT-STS:task_state | scheduling |

| OS-EXT-STS:vm_state | building |

| OS-SRV-USG:launched_at | None |

| OS-SRV-USG:terminated_at | None |

| accessIPv4 | |

| accessIPv6 | |

| addresses | |

| adminPass | ZCDZzpw8j5DA |

| config_drive | |

| created | 2017-06-08T12:23:12Z |

| flavor | Cloud Computing (604de11c-3222-4902-8523-11cc61b5b485) |

| hostId | |

| id | c036c942-b758-47f5-b67d-f7fbd7543650|

| image | ubuntu-16.04 (11f6b8aa-31df-4b66-8b42-5ee9760c47ba) |

| key_name | cc17g2os |

| name | CC17GROUP2 |

| progress | 0 |

| project_id | 4894c882bfcc4558ab515cdc55b32446 |

| properties | |

| security_groups | name='default' |

| status | BUILD |

| updated | 2017-06-08T12:23:13Z |

| user_id | 15e8e0d4ac9c4909b90f9f2c4f0994f2 |

| volumes_attached | |

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

After opening the TCP/SSH Port 22 for Ingress and Outgress for the associated Security group using UI, associating Floating Point IP to the launched instance

openstack server add floating ip c036c942-b758-47f5-b67d-f7fbd7543650 10.200.2.199

Connect to OpenStack server using SSH

$ ssh -i cc17g2os.key ubuntu@10.200.2.199

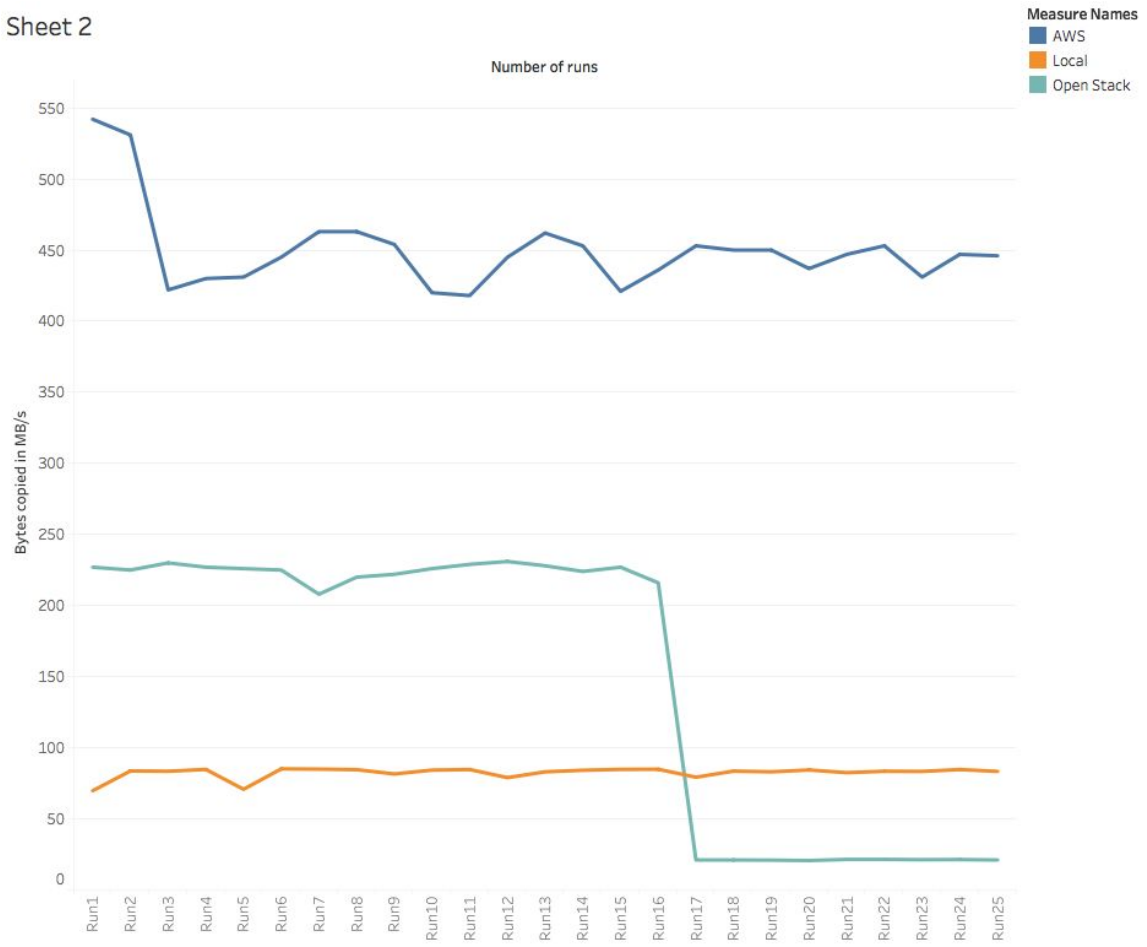
ubuntu@cc17group2:~/group2$

Performance Benchmarks

Disk benchmark

By running our script with DISKDUMP (dd), 4,3 GB of data is read. Blocks of 4096B are copied from /dev/vda1(the disk changes depending on the machine) to /dev/null, hence committed into a RAM's write cache, while the server simply continues to write data from the RAM cache to the hard disk to the null device which simple discards the written data.

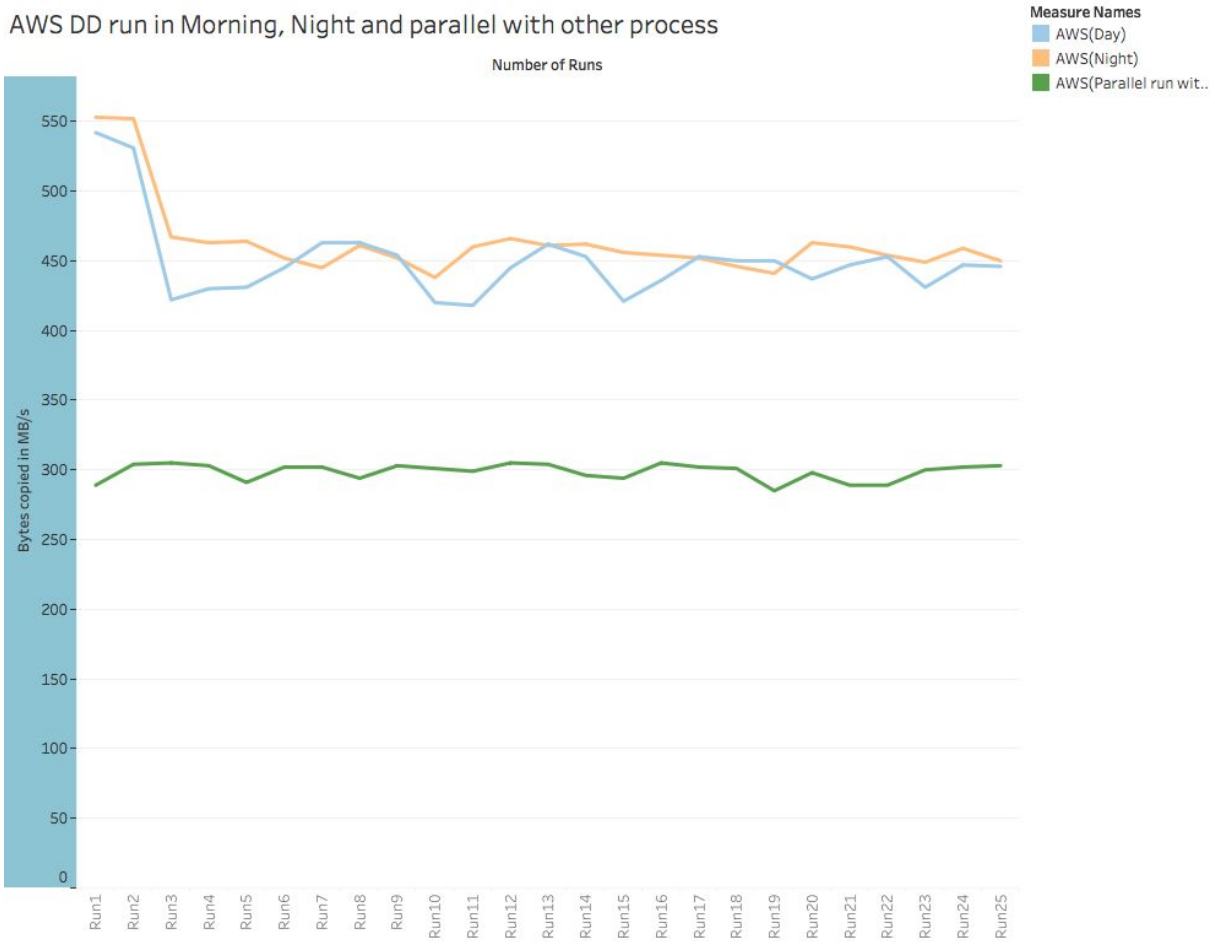
```
1  #!/bin/bash
2
3  READ_DEV=/dev/vda1
4  echo "Reading 4.3G of data from $READ_DEV (must run as root)"
5  dd if=$READ_DEV of=/dev/null bs=4096 count=1M
```



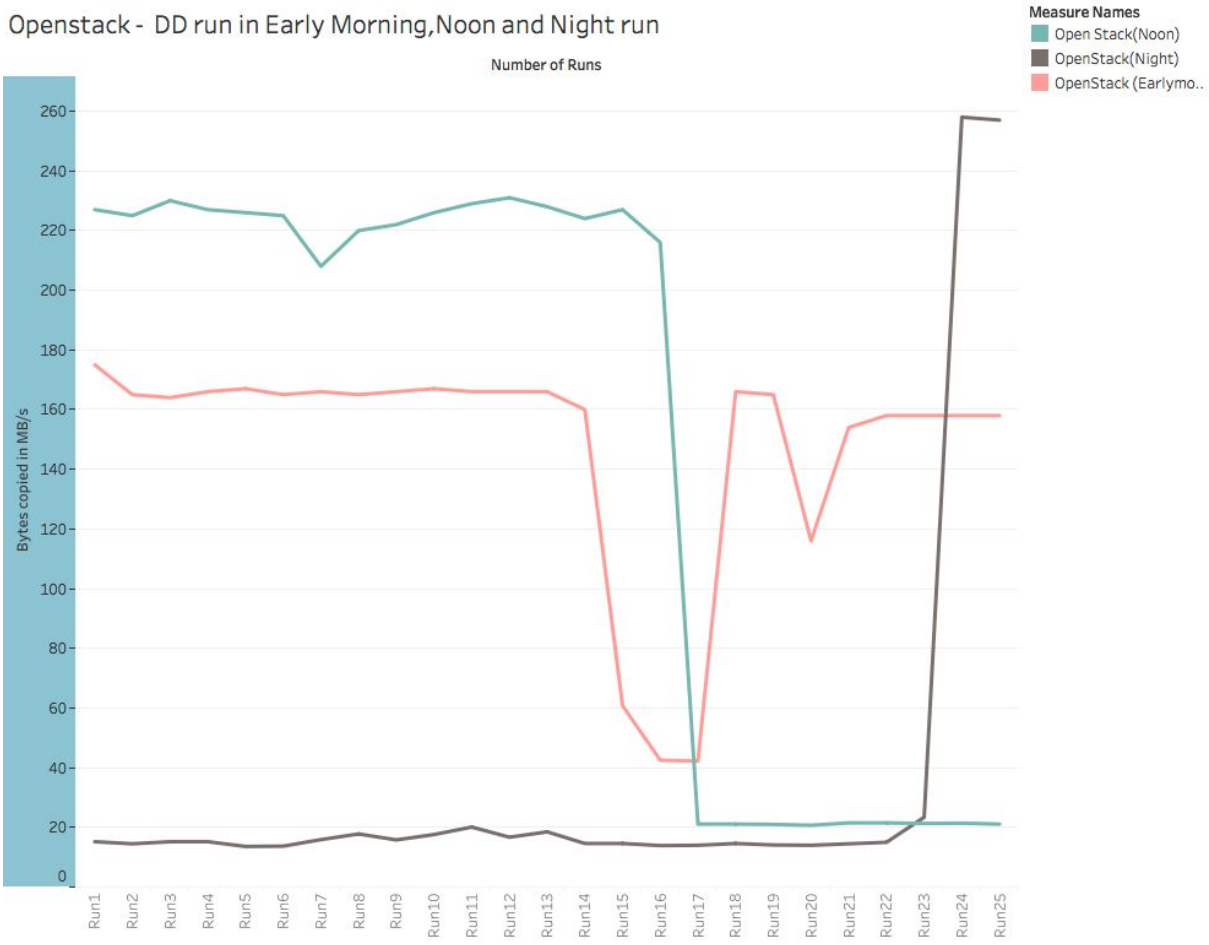
Average speed in local machine: ~75 MB/s
Average speed in OpenStack instance: ~225 MB/s(Dropped values are taken from a separate run)
Average speed in AWS instance: ~460MB/s

We observe almost 200% better performance by the AWS instance in comparison to the Openstack instance. Our local machine performs even poorer. AWS EC2 instance has SSD and way bigger RAM

than Openstack instance which has HDD and just 512MB of RAM. This has hugely impacted the dd benchmarking results in those instances. The local machine where dd was run has HDD and 4GB of RAM. It is still underperforming probably due to parallel processes being run at the same time as the benchmarking and also could be because of slower memory bus.



We observe the decrease in dd performance when it is run in parallel with other processes like running another dd parallely. This decrease when run in parallel is due to single core nature of the instance. It has vCPU 1 which says it's single core i.e. 1 CPU and hence multiple processes affect the benchmark result.



The cause for the difference in openstack performance during different time of the day is unknown.
In the reading taken during night time, there was a console output process(a heavy process) running in parallel with dd and hence there is such a low performance and when the console output was aborted, the performance shoot up!
Note: We used CRONTAB to schedule the script run

1. Look at the disk measurements. Are they consistent with your expectations. If not, what could be the reason?

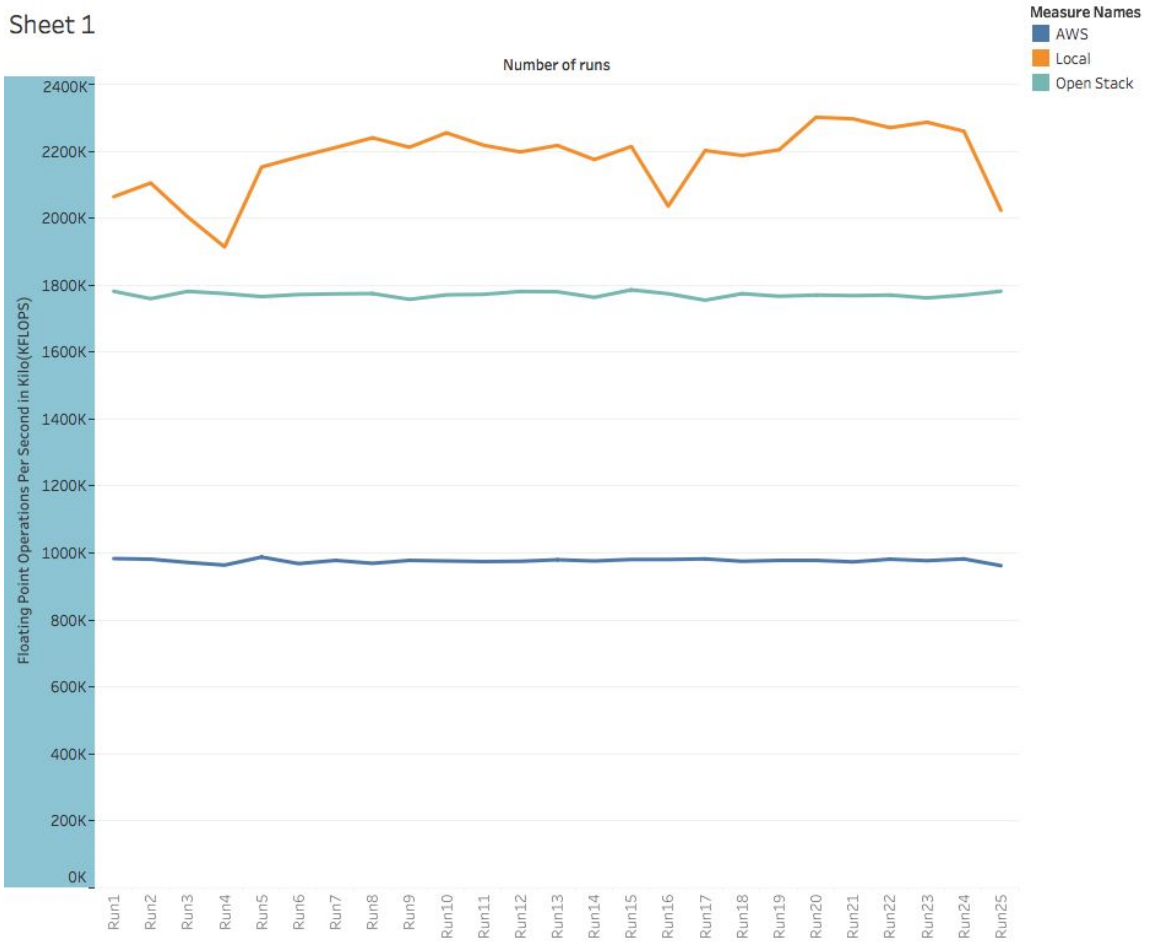
Among the virtualized instances on two different platforms viz., AWS and OpenStack we see an increasing disk read performance as they get more powerful as expected(AWS with RAM size 30GB and OpenStack with RAM size 512 MB) and the harddisk type one being SSD and the other being HDD. The difference between the novirt value and the virtualized one might be due to more powerful hard disks and faster memory bus.

2. Based on the comparison with the measurements on your local hard drive, what kind of storage solutions do you think the two clouds use?

Based on the observation, the virtual machines are still performing better than the local machien (because of hardware differences). The chosen AWS EC2 instance shows better performance compared to OpenStack instance. The reason being type of hard disk which is SSD in case of AWS and HDD in case of OpenStack. So, we think it is always better to adopt SSD type hard disk for better dd performance and also for durability reason.

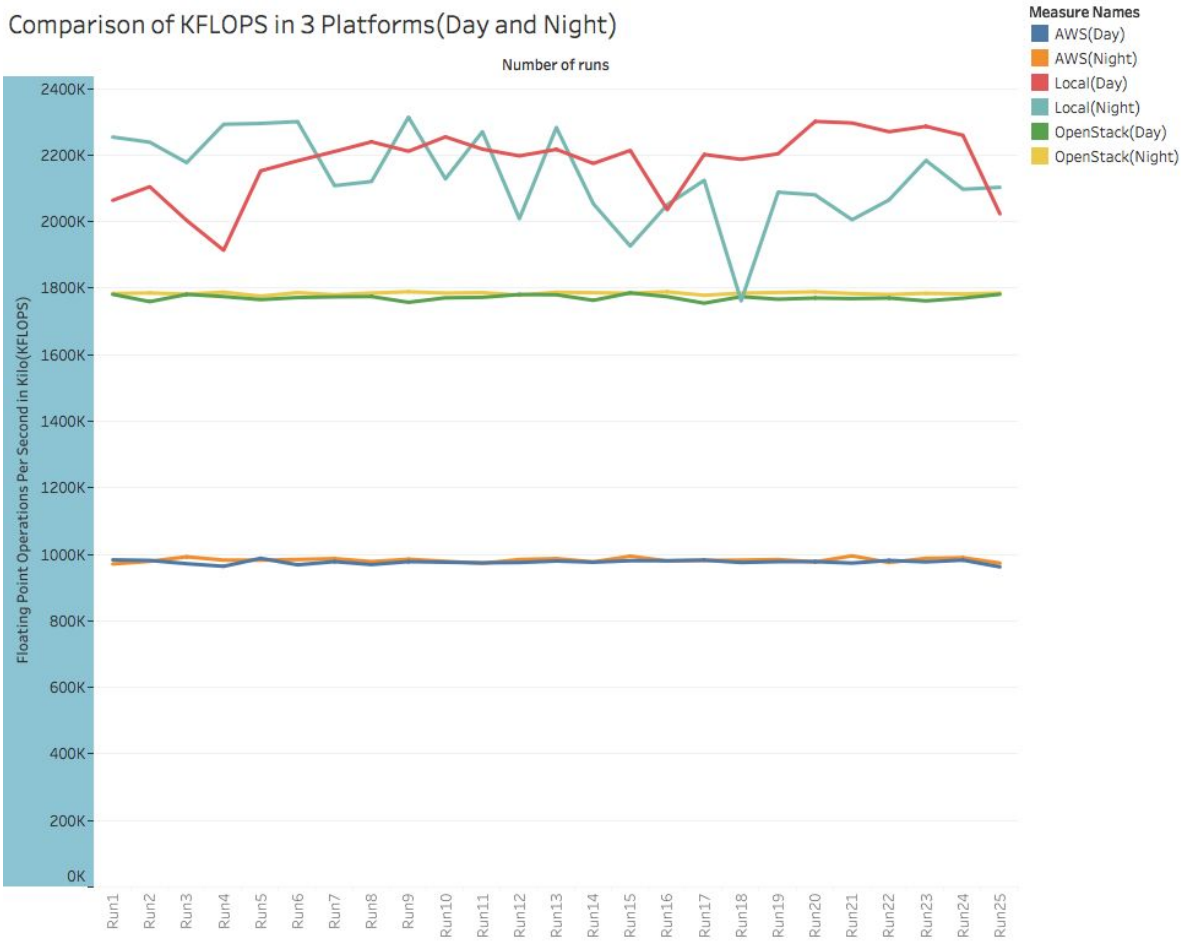
CPU benchmark (linpack.sh)

By running LINPACK, we are executing a huge amount of floating point operations, which are unprivileged instructions, and calculate the average KFLOPS (kilo floating point operations per second) as a measure of computing power.



KFLOPS in local machine: 2100K on average
KFLOPS in OpenStack instance: ±1800K
KFLOPS in AWS instance: ±1000K

We observe a 80% better performance by the OpenStack instance in comparison to the AWS instance. Our local machine performs even better with an average KFLOPS result of 2100K, resulting in a 110% increase in computing power compared to AWS EC2. The variation in the performance on our local machine is due to parallel processes being run at the same time as the benchmarking.



For our AWS and OpenStack instance, apart from their consistency, we don't observe any significant difference between their day and night performance. Even though our local machine takes different and irregular values and shows during the day or at night, the average remains the same. Hence, we can conclude that the time of benchmarking does not affect the performance of the machine.

1. Look at linpack.sh and linpack.c and shortly describe how the benchmark works.

linpack.sh compiles *linpack.c* using the gcc compiler and writes the result to the standard output. This benchmark consists of a linear algebra problem to rate the floating point performance of a computer. We create a random dense matrix A of size 1000 (*arsize= 1000*), a vector B which is equal to the product of A and a vector X consisting of only 1's. The computer will be required to

1. compute an LU factorization of A;
2. use this LU factorization above to solve A*X=B.

Rolled as well as unrolled loops are being used to compare their performance. Unrolled loops will perform better as their loops are done with an increment greater than 1.

FLOPS are calculated using the formula: *sockets * (cores / socket) * (cycles / second) * (FLOPS / cycle)*.

The number of KFLOPS (kilo floating point operations per second) is calculated as the average of operations / (CPU * 1000) for both rolled and unrolled loops.

2. Find out what the LINPACK benchmark measures (try Google). Would you expect paravirtualization to affect the LINPACK benchmark? Why?

As per the internet sources, the LINPACK benchmark measures the speed of a computer solving linear algebra calculations, LU factorization and a system of linear equations.

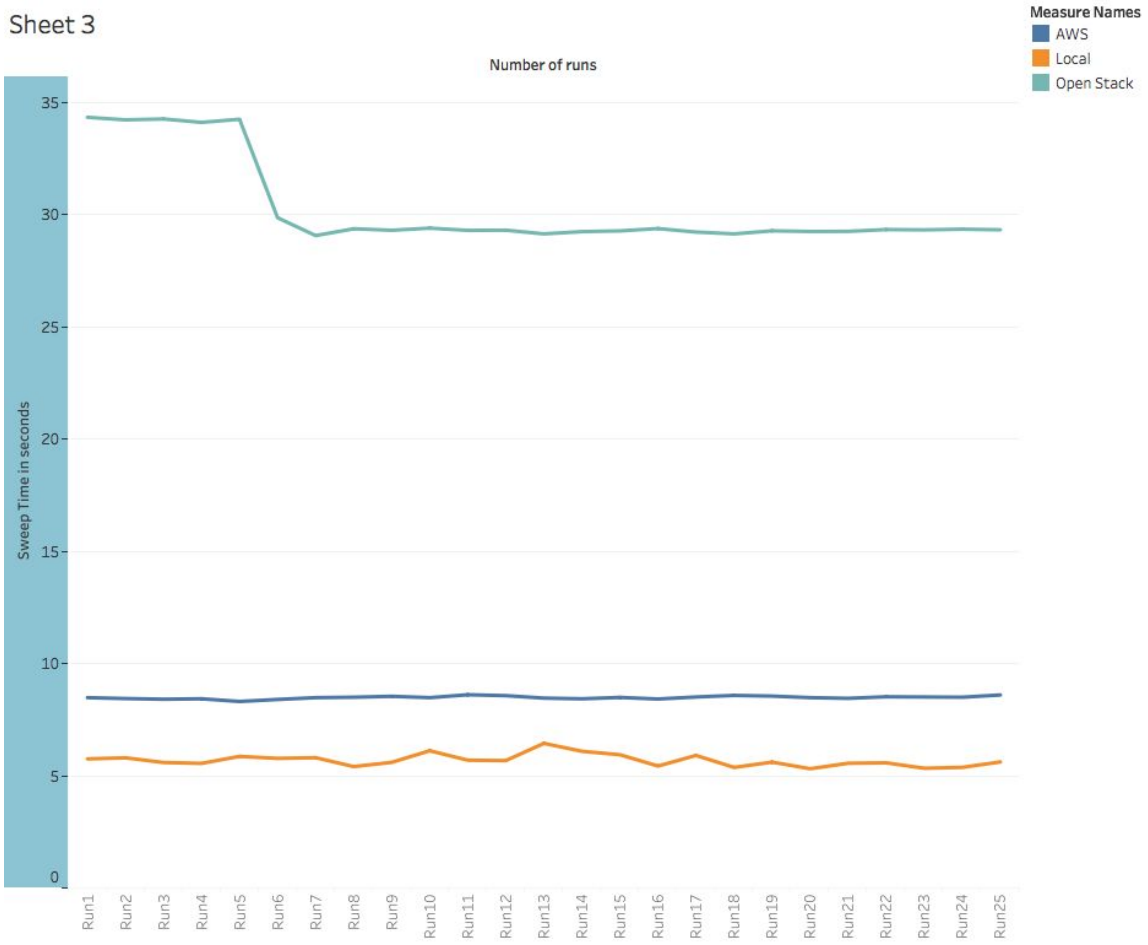
We don't expect paravirtualization to affect our LINPACK benchmark results, assuming that the VM has its dedicated access to the host's hardware, because the LINPACK measurement only involves unpriveleged instructions which are all executed on the CPU.

3. Look at your LINPACK measurements. Are they consistent with your expectations? If not, what could be the reason?

We observe that both cloud instances have a vCPU equal to 1, indicating single core but it ould nto affect LINPACK as it is not multithreaded. However, the instance type of AWS is m3.medium, which is for general purpose and not compute optimized. We assume that our OpenStack instance is compute optimized and therefore performing better. Our local machine on the other hand, has outperformed both virtual machines because of its higher RAM-size of 4GB.

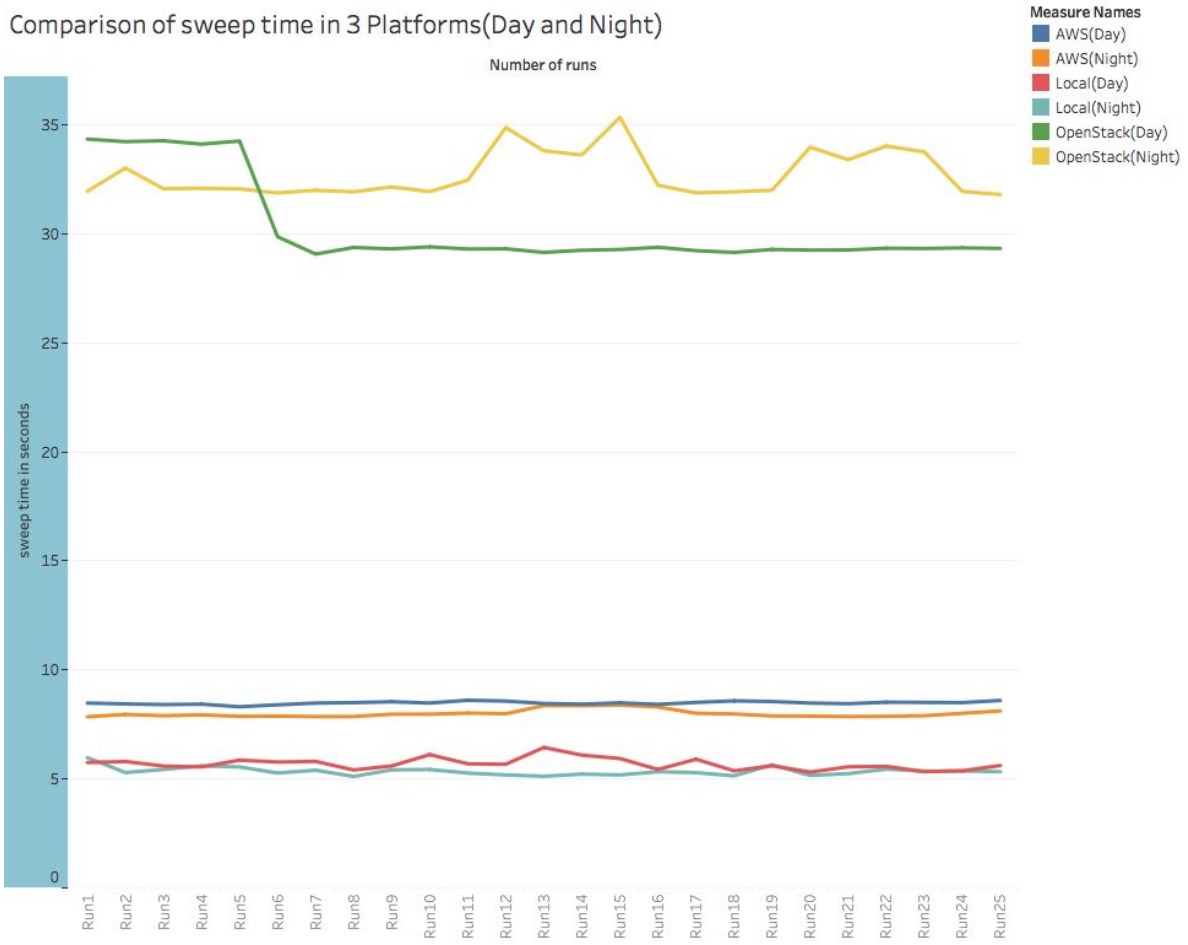
Memory benchmark (memsweep.sh)

By running MEMSWEEP, we allocate a big array (*ARR_SIZE (8096 * 4096)*) in heap memory, access it at different, random locations and measure the time in seconds needed to do this as a benchmark of memory access performance.



sweep time (s) in OpenStack instance: ±34 for the first 5 runs and ±29 for the remaining ones
sweep time (s) in AWS instance: ±8
sweep time (s) in local machine: ±6

All the sweep time values remain consistent. The AWS instance's sweep time is about 30% slower than our local machine's. Even slower, is the OpenStack instance with a sweep time between 29 and 34 seconds, compared to AWS's average of 8 seconds and our local machine's average time of 6 seconds. Note that the fall in values after the 5th run is caused because we executed the first 5 and the last 20 runs seperately.



For our AWS instance and our local machine, we only observe a neglectable difference between their day and night performance and they both remain consistent. The OpenStack instance takes more consistent memsweep time values during the day - at night, they are more irregular. Note that the fall in values after the 5th run is caused because we executed the first 5 and the last 20 runs separately. The average sweep time of day and night respectively remains the same for our OpenStack instance.

1. Find out how the memsweep benchmark works by looking at the shell script and the C code. Would you expect virtualization to affect the memsweep benchmark? Why?

The MEMSWEEP benchmark measures how much time (in seconds) it takes to access heap memory at several locations. A cache miss will occur and the data will have to be loaded from memory, as we set the step width accordingly. As for the VMs, we expect it to take more time than our local machine, as we need to wait for a validation of the write requests by their hypervisors.

2. Look at your memsweep measurements. Are they consistent with your expectations. If not, what could be the reason?

As expected, the local machine and AWS show near results and both outperform Openstack instance, the reason being the local machine's RAM size of 4GB and the OpenStack machine's RAM size of only 512MB.

Best Practices Followed

1. Usage of GIT to share scripts across the machines
2. Writing single script which executes all the benchmarking scripts for a specified number of times and outputs the results to respective files
3. Using NOHUP to let the script run in background even when the CLI gets disconnected
4. Using CRONTAB to schedule script-run
