AWS Public Cloud Setup

Make sure to setup an account with Amazon AWS by visiting:
http://console.aws.amazon.com

We will be using AWS free tier services. See what is available in free tier:
http://aws.amazon.com/free/

Make sure the lab setup is completed before attempting the CLI portion of this lab.

AWS requires a credit card, but it won't be charged if you continue to use free tier. Remember, the most expensive part of aws services is EC2 instances (Virtual Machines). If you limit yourself to t2.micro instances, then the monthly cost of using other aws resources will be nominal, a few dollars a month.

Launching Instance in AWS Cloud (aws console)

Type a URL in web browser: http://console.aws.amazon.com

Login to AWS console using your account login/password.

Follow steps below:
Step 1: Click **Launch Instance**

Step 2: Select AMI (Amazon Machine Image) to launch an instance. I have custom built AMI: name: **UCSC-BIONIC**. AMI is only available in **us-east-1 (N.Virginia)** region. Click "**Community AMIs**" and search for "**UCSC-BIONIC**" and then select. Choose instance type: **t2.micro** *(Free tier eligible)* to avoid getting charge.

Step 3: Click **Next:Configure Instance Details**. Accept defaults.

- This will launch one t2.micro instance in a default VPC in any AZ of us-east-1 region
- It will use root IAM credentials. Root IAM credential has full access.
- Instance will be launched on a shared hardware (multi-tenancy)

Step 4: Click **Next:Add Storage**. Root volume is on network storage EBS. If you need to configure additional storage/volume then you specify heer. Configure root volume 20GB . Later, if we decide to additional volumes, we can add new volume and attach to a running instance.

- *Some instance types come with additional direct attached storage, called ephemeral storage or **instance volume**. Instance cost includes: EBS root volume and any direct attached storage if supported on the instance (Not all instances have instance storage).*

Step 5: Click **Next:Tag instance**. You can tag the resource using some descriptive key (value is optional) like *prod, test, sales, marketing* etc.” to AWS resources. and thus can use it for filtering when searching for resources.
Step 6: Click **Next:Configure Security Group.** This configures firewall rule at the instance that controls ingress network traffic. Select: *“Create a new security group”,* “*Security group name: “**cloudperf-yourname”***. There is a ssh rule already provided for login into the instance. Click ***“Add Rule”*** to allow ingress traffic from **Source: Anywhere** to list of network ports:

- *ssh*
- *http*
- *https*
- *Custom TCP Rule: 7400 - 7500 (port range)*
- *Custom TCP Rule: 44300 - 44400 (port range)*

NOTE: We will use ports within range of 7400-7500 and 44300 - 44400 to test a few network services running on ec2 instances.

Step 6: Click **Review and Launch**. Review all the settings for instance launch.
Step 7: Click **Launch**. You will be prompted: “Create a new key pair” and “Key pair name”. Name the key pair “***mysshkey***” and download the public/private keypair (.pem file) on your local machine. Once the instance state shows “ running”, it is ready to ssh. Find the public hostname of the instance or IP address and then ssh/login the instance from vagrant VM running on your desktop.

```
  $ ssh -i mysshkey.pem ubuntu@ec2.compute.amazonaws.com
```

NOTE: Without keypair, you cannot ssh (login) to instance. You may have to install ssh (putty software) on your laptop

Find the instance name or ip address from aws console and test network access to services running on cloud instance:
http://ec2-xxxx.compute-1.amazonaws.com:44323/

Launching instance with IAM Role Instance Profile (aws console)

Applications running on EC2 cloud instances may need access to aws resources such as: S3 bucket to download or upload a file for processing. To allow EC2 instances to access cloud resources you must create an IAM Instance Profile and attach it to EC2 instances. Think of instance profile containing an IAM role that can be passed to EC2 instances.

One option is to bake your *AWS access/secret* keys into an AMI when launching EC2 instances but that is a security risk. IAM ***Instance Profile*** allows EC2 instances to assume a role that provides access to a subset of aws resources without storing aws access/secret keys.This way applications running on EC2 instances are able to access AWS resources. IAM instance profile contains:

- **Trust** information that contains policy to allows application running on EC2 instance to assume role (AssumeRole) by using temporary aws security keys
- Type of **permissions** or action application running on AWS instance can perform on aws resources. For example, you can allow applications to only list and download files from your s3 buckets, but not upload or delete files in the s3 bucket.
- There are ready-to-use policies available. We can also can create custom policy

Step 1: Login to aws console and select **IAM** in Services Tab.

- Click **Roles**.
- Click **Create role**
- Click **EC2** under "*Choose the service that will use this role"*
- Click **Next:permissions**
- Filter: Policy type: **AmazonS3FullAccess.** Select the policy
- Click **Next: Review**
- Give a name to this new role: **EC2toS3Access**
- Click: **Create role**
- New role should be available for use

We launched the instance earlier without specifying any IAM role. This time, we will associate the IAM role to EC2 instances as Instance Profile. We will follow the same steps for launching instance except in step 3:

Step 3: Click **Next:Configure Instance Details**.

- We will select IAM role: EC2toS3Access

NOTE: All other steps are the same. When prompted for a security group and key pair, use the existing one created when the first instance was launched. No need to create another set.

vector: helps you to fetch instance level performance metrics: cpu, memory, io, network from the instance on demand and displays it in your browser. Vector is a client side tool (javascript) and runs in browser.

http://ec2-xxxx.compute-1.amazonaws.com:44323/

NOTE: You may need to start pmwebd in order for vector to work. First ssh into cloud instance and run command on the cloud instance:

$ sudo service pmwebd start

It will download javascript, images and css files into your browser and execute them. Once loaded, you specify your cloud instance in the **"Hostname"** field of the vector dashboard. Vector will start fetching live system metrics at 2 second granularity from the cloud instance or Vagrant VM. Learn more about vector at: http://vectoross.io/docs/getting-started.html

Abyss:
For abyss to work, you need to login into cloud instance and then run:

```
$ cd /usr/share/grafana/bin
$ sudo nohup ./grafana-server start &
```

Abyss dashboard: http://ec2-xxxx.compute-1.amazonaws.com:7410/

Create S3 bucket

Search for S3 service in Services tab in aws console
Click **Create bucket**. Give a bucket name, call it **openworkshop**
Click **openworkshop**
Click **Upload** to upload a file from your laptop. Select any file on your laptop and upload it. You should see the file listed.

Now open the **ssh** window to both instances that you have launched.

List all S3 buckets in your account

```
$aws s3 ls
```

List **openworkshop** bucket

```
$ aws s3 ls openworkshop
```

Download a file from **openworkshop** bucket

```
$aws s3 cp s3://openworkshop/<filename> --region us-east-1 .
```

Note: "." means download file into a current directory

Upload a file to **openworkshop** bucket:

```
$aws s3 cp /var/log/syslog s3://openworkshop
```

Confirm it:

```
$ aws s3 ls openworkshop
```

USING AWS CLI

Launching Instance in AWS Cloud (CLI):

You need a vagrant/virtualbox setup working on your laptop to complete this lab.

To launch an ec2 instance in AWS Cloud using CLI, login to Vagrant VM and install aws cli tools (already installed).
Run ***"aws configure"***. You need to have root **aws access/secret** or IAM aws access/secret (Recommended) keys

```
$ aws configure
AWS Access Key ID [None]: <YOUR_AWS-ACCESS-KEY>
AWS Secret Access Key [None]: <YOUR-AWS-SECURITY-KEY>
Default region name [None]: us-east-1
Default output format [None]: text
```

Check if AWS can successfully authenticate your request

```
$ aws ec2 describe-regions
```

Sample output:

```
REGIONeu-west-1         ec2.eu-west-1.amazonaws.com
REGIONap-southeast-1    ec2.ap-southeast-1.amazonaws.com
REGIONap-southeast-2    ec2.ap-southeast-2.amazonaws.com
REGIONeu-central-1      ec2.eu-central-1.amazonaws.com
REGIONap-northeast-1    ec2.ap-northeast-1.amazonaws.com
REGIONus-east-1         ec2.us-east-1.amazonaws.com
REGIONsa-east-1         ec2.sa-east-1.amazonaws.com
REGIONus-west-1         ec2.us-west-1.amazonaws.com
REGIONus-west-2         ec2.us-west-2.amazonaws.com
```

We will use AMI UCSC-BIONIC to launch an instance. Command “ec2-run-instances” support number of options. We will use:

--key-name Your private/public key. Do not add the pem extension when specifying key with ec2-run-instances command
--security-groups   security-group-name
--count   number of instances
--instance-type instance type
--region ***us-east-1***
--image-id ami-033a3273401a27142 **(***ami-name: UCSC-BIONIC: ami-033a3273401a27142* **)**

```
$ aws ec2 run-instances --key-name cloudperf-netflix --security-groups UCSC --count 1
--instance-type t2.micro --region us-east-1 --image-id ami-033a3273401a27142
```

Sample output

```
442122186855        r-0f9bcedf0e1bd1dbd
INSTANCES           0           x86_64            False      xen          ami-033a3273401a27142
i-09974b9e3f56c20c7             t2.micro   cloudperf-netflix      2018-10-16T21:55:09.000Z     ip-172-31-7-89.ec2.internal
172.31.7.89                     /dev/sda1 ebs     True                    subnet-a241b6fb     hvm         vpc-eb69dd8e
MONITORING          disabled
NETWORKINTERFACES                          0e:c3:c3:81:e3:be     eni-08daf1a0bd736dbf4         442122186855
ip-172-31-7-89.ec2.internal     172.31.7.89       True       in-use       subnet-a241b6fb     vpc-eb69dd8e
ATTACHMENT          2018-10-16T21:55:09.000Z      eni-attach-0b6323bc8e5350e42               True        attaching
GROUPS sg-75f1ec0a              UCSC
PRIVATEIPADDRESSES              True        ip-172-31-7-89.ec2.internal      172.31.7.89
PLACEMENT           us-east-1c                    default
SECURITYGROUPS                  sg-75f1ec0a       UCSC
STATE     0         pending
STATEREASON         pending     pending
```

Now you are ready to ssh into the cloud instance. Find the hostname assigned by AWS. It will be similar to: **ec2-xxxx.us-east-1.compute.amazonaws.com.**

```
$ ssh -i your.pem ubuntu@ec2-xxxx.us-east-1.compute.amazonaws.com
```

Find the instance name or ip address from aws console and test network access to services running on cloud instance.
I have installed two services, both open source softwares: vector and abyss in UCSC-BIONIC images (aws, vagrant). Vectors require pcp and abyss require grafana, apache, graphite packages installed on the system that you are interested in monitoring. Cloud images for Vagrant box and AWS ami both have these required packages installed.

vector: helps you to fetch instance level performance metrics: cpu, memory, io, network from the instance on demand and displays it in your browser. Vector is a client side tool (javascript) and runs in browser.
http://ec2-xxxx.compute-1.amazonaws.com:44323/

NOTE: You may need to start pmwebd in order for vector to work. First ssh into cloud instance and run command on the cloud instance:
$ sudo service pmwebd start

It will download javascript, images and css files into your browser and execute them. Once loaded, you specify your cloud instance in the **"Hostname"** field of the vector dashboard. Vector will start fetching live system metrics at 2 second granularity from the cloud instance or Vagrant VM. Learn more about vector at: http://vectoross.io/docs/getting-started.html

Abyss:
For abyss to work, you need to login into cloud instance and then run:

```
$ cd /usr/share/grafana/bin
$ sudo nohup ./grafana-server start &
```

Abyss dashboard: http://ec2-xxxx.compute-1.amazonaws.com:7410/

Launching instance with IAM Role Instance Profile *(CLI)*

Applications running on EC2 cloud instances may need access to aws resources such as: S3 bucket to download or upload a file for processing. To allow EC2 instances to access these cloud resources you must create IAM Instance Profile and attach it to EC2 instances. Think of instance profile containers for an IAM role, that can be passed to EC2 instances.

One option is to bake your *AWS access/secret* keys into an AMI when launching EC2 instances but that is a security risk. IAM ***Instance Profile*** allows EC2 instances to assume role that provides access to subset of aws resources without storing aws access/secret keys.This way applications running on EC2 instances can able to access AWS resources. IAM instance profile contains:

- **Trust** information that contains policy to allows application running on EC2 instance to assume role (AssumeRole) by using temporary aws security keys
- Type of **permissions** or action application running on AWS instance can perform on aws resources. For example, you can allow applications to only list and download files from your s3 buckets, but not upload or delete files in s3 bucket.
- There are ready-to-use policies available. In this CLI exercise, we will create a new Role and attach a custom policy to it.

- Create two files in the home directory after login into Vagrant VM: *S3Bucket-EC2-Trust.json* and *S3Bucket-EC2-Permission.json*.
    - **S3Bucket-EC2-Trust.json:** You specify a trusted entity that is allowed access to AWS resource. In this case EC2 instance.
    - **S3Bucket-EC2-Permission.json:** Inline policy that can be associated to role. It lists fine grained permissions to AWS resources that a role is allowed to perform. In this case, we are allowing application to list and download S3 buckets in our account.

S3Bucket-EC2-Trust.json

```
       {
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
    "Service": "ec2.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

S3Bucket-EC2-Permission.json

```
     {
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Effect": "Allow",
        "Resource": "*"
      }
   ]
}
```

Create an Instance profile

```
$aws iam create-instance-profile --instance-profile-name cloudperf-S3-Profile
Sample output:
INSTANCEPROFILE          arn:aws:iam::442122186855:instance-profile/cloudperf-S3-Profile          2018-10-16T22:16:13Z
AIPAI5NH6JCKXYC7RISA6    cloudperf-S3-Profile /
```

Create IAM role and attach an assume role policy in file S3Bucket-EC2-Trust.json

```
$ aws iam create-role --role-name cloudperf-S3-Profile --assume-role-policy-document file://S3Bucket-EC2-Trust.json
Sample output:
ROLE      arn:aws:iam::442122186855:role/cloudperf-S3-Profile           2018-10-16T22:21:17Z
AROAIO4GMK73DRL3O3MYO            cloudperf-S3-Profile
ASSUMEROLEPOLICYDOCUMENT         2012-10-17
STATEMENT          sts:AssumeRole         Allow
PRINCIPAL          ec2.amazonaws.com
```

Give IAM role permission to perform tasks to your S3 bucket as specified in file S3Bucket-EC-Permission.json

```
$aws iam put-role-policy --role-name cloudperf-S3-Profile --policy-name cloudperf-S3-Profile --policy-document
file://S3Bucket-EC2-Permission
<no output>
```

Now attach it to an instance profile

```
$aws iam add-role-to-instance-profile --instance-profile-name cloudperf-S3-Profile --role-name cloudperf-S3-Profile
<no output>
```

Check if the instance profile *S3Bucket-EC2-Instance-Profile* is created successfully

```
$aws iam list-instance-profiles-for-role --role-name cloudperf-S3-Profile --query "InstanceProfiles[0].InstanceProfileName" --output text
Sample output:
cloudperf-S3-Profile
```

Launch instance with instance profile S3Bucket-EC2-Instance-Profile attached

```
$aws ec2 run-instances --key-name cloudperf-netflix --security-groups UCSC --count 1 --instance-type t2.micro --region us-east-1
--image-id ami-033a3273401a27142 --iam-instance-profile Name=cloudperf-S3-Profile
Sample Output:

442122186855        r-0fefbaa58910d2ea6
INSTANCES           0            x86_64             False       xen         ami-033a3273401a27142
i-075e1ae4ad30942a0          t2.micro    cloudperf-netflix      2018-10-16T22:32:07.000Z     ip-172-31-0-87.ec2.internal
172.31.0.87                  /dev/sda1 ebs          True                    subnet-a241b6fb      hvm       vpc-eb69dd8e
```

```
IAMINSTANCEPROFILE      arn:aws:iam::442122186855:instance-profile/cloudperf-S3-Profile      AIPAI5NH6JCKXYC7RISA6
MONITORING      disabled
NETWORKINTERFACES               0e:6a:2b:1c:eb:d8   eni-0f92068a7c258e37f       442122186855
ip-172-31-0-87.ec2.internal     172.31.0.87         True        in-use      subnet-a241b6fb     vpc-eb69dd8e
ATTACHMENT      2018-10-16T22:32:07.000Z    eni-attach-0f8e4ad23752e4ff8 True       attaching
GROUPS sg-75f1ec0a      UCSC
PRIVATEIPADDRESSES      True        ip-172-31-0-87.ec2.internal     172.31.0.87
PLACEMENT       us-east-1c                  default
SECURITYGROUPS          sg-75f1ec0a         UCSC
STATE   0       pending
STATEREASON     pending     pending
```

Create a bucket from your vagrant VM.
Make sure you have aws cli installed on your machine:
https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html
For Linux do the following:

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

```
$aws s3api create-bucket --bucket myfirstbucket.cloudperf.net --region us-east-1
```

List all buckets

```
$ aws s3 ls
```

List files in bucket (it should return empty list)

```
$aws s3 ls myfirstbucket.cloudperf.net
```

Upload a file:

```
$aws s3 cp /var/log/syslog s3://myfirstbucket.cloudperf.net/
```

List files in bucket:

```
$aws s3 ls myfirstbucket.cloudperf.net/
```

Download file from bucket:

```
$aws s3 cp s3://myfirstbucket.cloudperf.net/syslog --region us-east-1 .  << dot means current directory
```

Now ssh to the instance that you launched with the instance profile attached.

```
$ ssh -i <your.pem> ubuntu@ec2-instance-host-name
```

Check if you can list files, download and upload. Instance profile attached to the instance allows only listing and downloading files from S3 buckets. Upload should fail!

List files in bucket: myfirstbucket.cloudperf.net

```
$aws s3 ls s3://myfirstbucket.cloudperf.net/ --region us-east-1
```

Download file from bucket: myfirstbucket.cloudperf.net

```
$aws s3 cp s3://myfirstbucket.cloudperf.net/syslog --region us-east-1 . << Dot means current directory
```

Upload a new file to bucket

```
$ mv syslog mylog
$aws s3 cp mylog s3://myfirstbucket.cloudperf.net/ --region us-east-1
```

Clean up:

List all instances in the region

```
$ aws ec2 describe-instances
```

Terminate instance(s):

```
$aws ec2 terminate-instances --instance-ids <instance-name>
```

Example:

```
$aws ec2 terminate-instances --instance-ids i-075e1ae4ad30942a0 i-09974b9e3f56c20c7
```

Sample Output:

```
TERMINATINGINSTANCES   i-075e1ae4ad30942a0
CURRENTSTATE   32      shutting-down
PREVIOUSSTATE  16      running
TERMINATINGINSTANCES   i-09974b9e3f56c20c7
CURRENTSTATE   32      shutting-down
PREVIOUSSTATE  16      running
```

List all iam roles that you have created

```
$ aws iam list-roles
..
```

Remove role cloudperf-S3-Profile from cloudperf-S3-Profile instance profile

```
$aws iam remove-role-from-instance-profile --instance-profile-name cloudperf-S3-Profile --role-name cloudperf-S3-Profile
<no output>
```

Delete instance profile cloudperf-S3-Profile

```
$aws iam delete-instance-profile --instance-profile-name cloudperf-S3-Profile
<no output>
```

Remove the cloudperf-S3-Profile policy attached to role

```
$aws iam delete-role-policy --role-name cloudperf-S3-Profile --policy-name cloudperf-S3-Profile
<no output>
```

Remove the role cloudperf-S3-Profile

```
$aws iam delete-role --role-name cloudperf-S3-Profile
<no output>
```

NOTE:

*There are two types of policies that can be attached to Role: **Managed and Inline**. Managed policies are ready-to-use policies built to perform commonly used functions in AWS. Inline policy is a custom policy written to do a specific purpose, similar to one we have created in this exercise. You can delete Inline policy, but only can detach managed policy associated with the role. For example, to detach Managed policy "AdministratorAccess" from role AllAccess-EC2-Role, you specify the ARN path instead of policy name:*

$aws iam detach-role-policy --role-name AllAccess-EC2-Role --policy-arn arn:aws:iam::aws:policy/AdministratorAccess

Learn more IAM roles, instance profiles, permission etc..
http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html
http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2.html
http://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html
http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies.html
http://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html

AMI Customization (Optional):

You can build a customized AMI that can have all required packages installed and unnecessary packages removed. I built ***UCSC-BIONIC*** AMI using the same technique. Launch instance from the base vanila AMI available in AWS marketplace. Once the instance is launched and running, ssh into it and install/uninstall packages. Once the configuration meets the requirements, go to aws console, select the running instance and then choose *"**Action**"* and select *"**create image**"*. Give an image a new name: "Your Name". Instances will be stopped to take a snapshot of the root volume and a new root image (AMI) is created with "Your Name". You can now launch a new instance by selecting custom AMI.

you don't have to install all sorts of packages and bake it. You can always pass a user script, called "**--user-data"** at instance launch time (part of AWS metadata service), Script install required packages at instance launch time. This way you can continue using base AMI and install needed packages at server launch time instead of baking into base AMI. Only drawback is that it may take longer to bring up the instance when installing packages at instance launch time.