Distributed ML on AWS Cloud: Computing with CPUs and GPUs

Xin Wang @ BDAL (08/24/2021)

Computation on Cloud

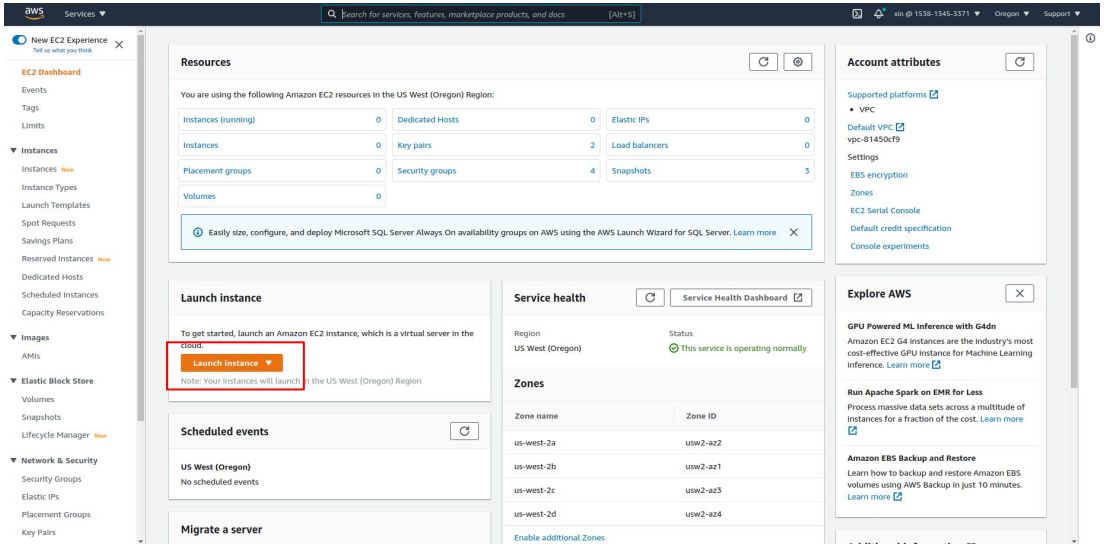
- Cloud Computing:
  - Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services (such as computing power, storage, and databases) from a cloud provider like Amazon Web Services (AWS), Microsoft Azure, Google Cloud, ...
- Benefits:
  - Agility
  - Elasticity
  - Cost savings
  - Deploy globally in minutes
  - ...

ML Computation on AWS Cloud

- Single machine computation:
  - CPU computation
  - GPU computation
- Distributed (Multiple) machine computation:
  - Multi-CPUs computation
  - Multi-GPUs computation on single virtual node (skipped)
  - Multi-GPUs computation on multiple virtual nodes
- Demo
  - Web based
  - Command line automation via Boto library (skipped)

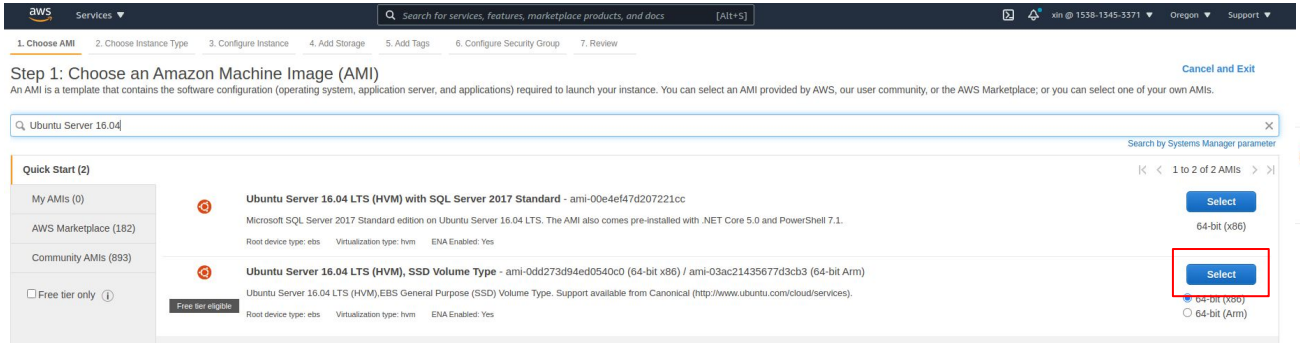
AWS Amazon Elastic Compute Cloud (EC2)

- Step 1: Launch instances on EC2: https://us-west-2.console.aws.amazon.com/ec2/v2/home



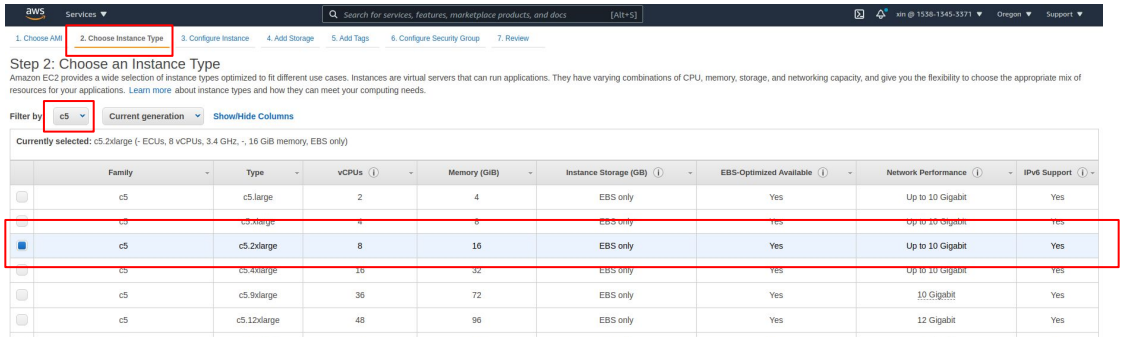
AWS Amazon Elastic Compute Cloud (EC2)

- Step 2: Choose an Amazon Machine Image (AMI)
  - An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance.
  - For CPU applications, we use Ubuntu Server 16.04 LTS (HVM), SSD Volume Type; for GPU case, we use Deep Learning Base AMI (Ubuntu 16.04) Version 40.0.



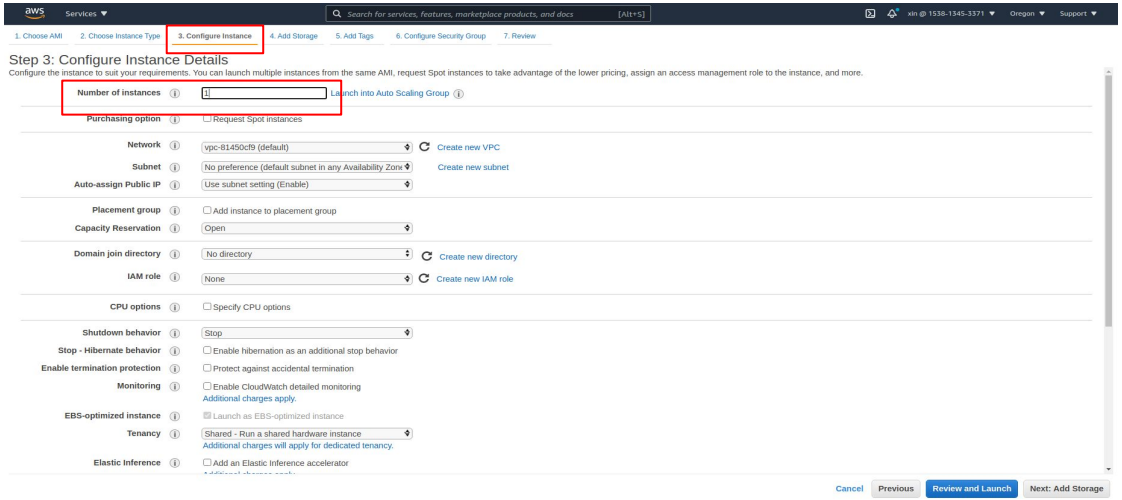
AWS Amazon Elastic Compute Cloud (EC2)

- Step 3: Choose an Instance Type
  - Based on your purpose, AWS provides various instance types on https://aws.amazon.com/ec2/instance-types/
    - For CPU application, we use c5.2xlarge instance in our demo;
    - For GPU application, we use p3.2xlarge instance in our demo.



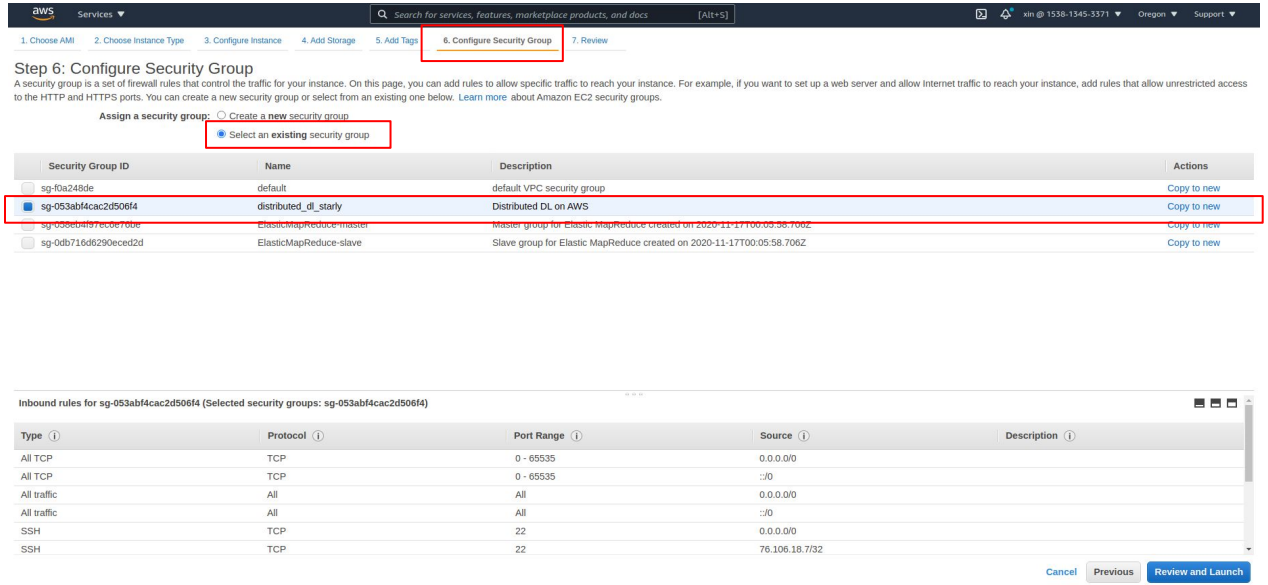
AWS Amazon Elastic Compute Cloud (EC2)

- Step 4: Configure Number of instances
  - We use 1 instance for single machine computation, and 2 instances for distributed computation.



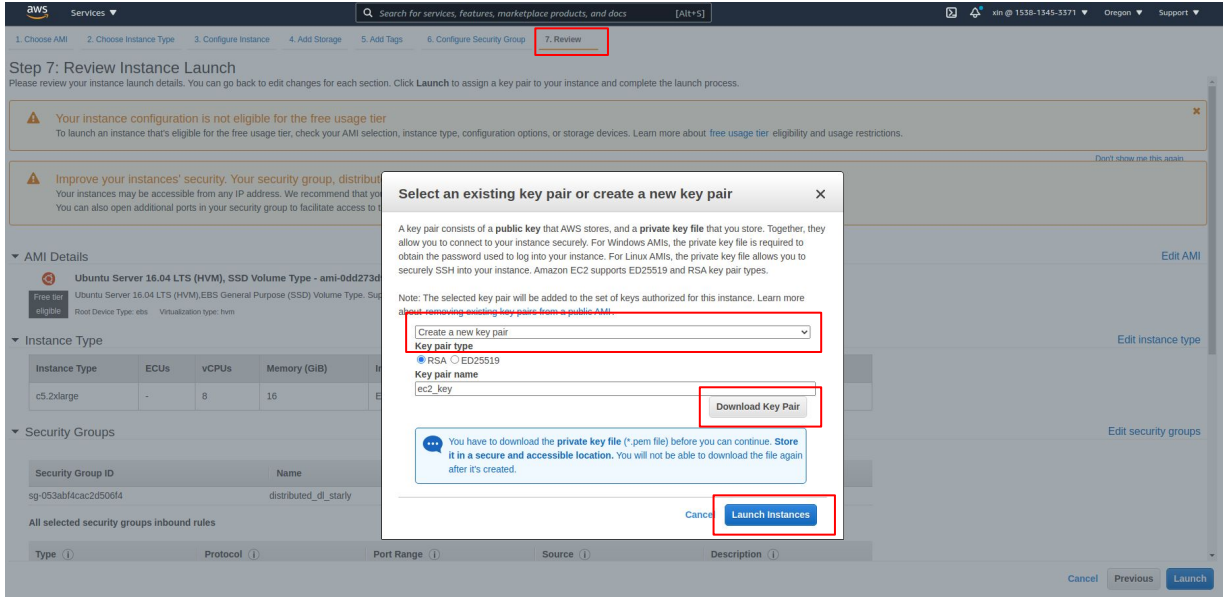
AWS Amazon Elastic Compute Cloud (EC2)

- Step 5: Configure Security Group



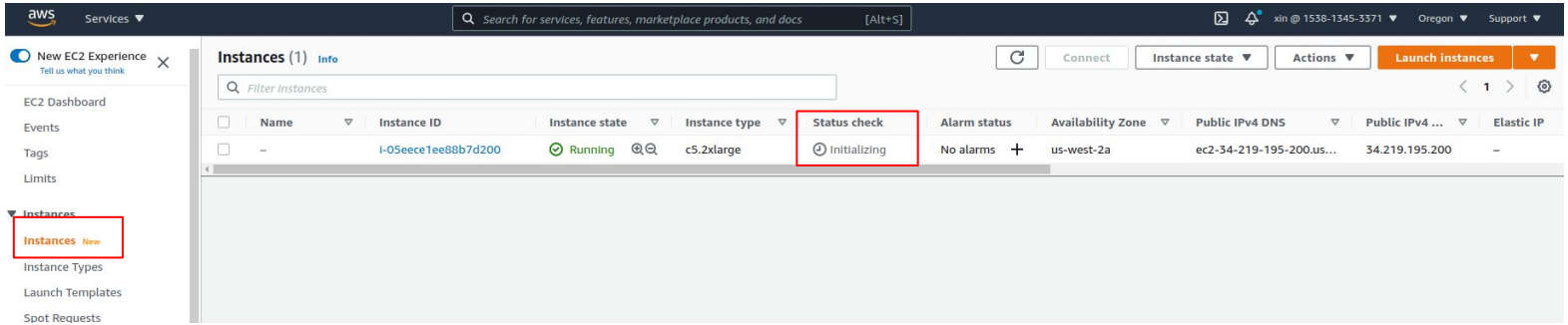
AWS Amazon Elastic Compute Cloud (EC2)

- Step 6: Review, Create your SSH key pair, and Launch



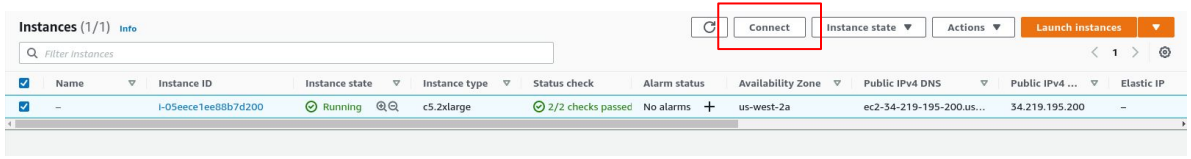
AWS Amazon Elastic Compute Cloud (EC2)

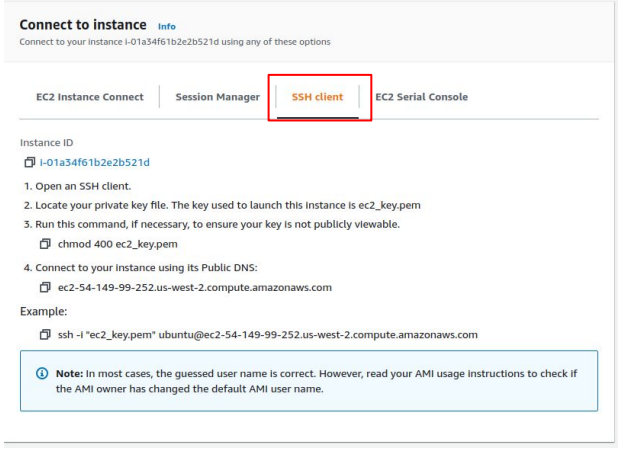
- Step 7: View your Instance and wait for Initialing

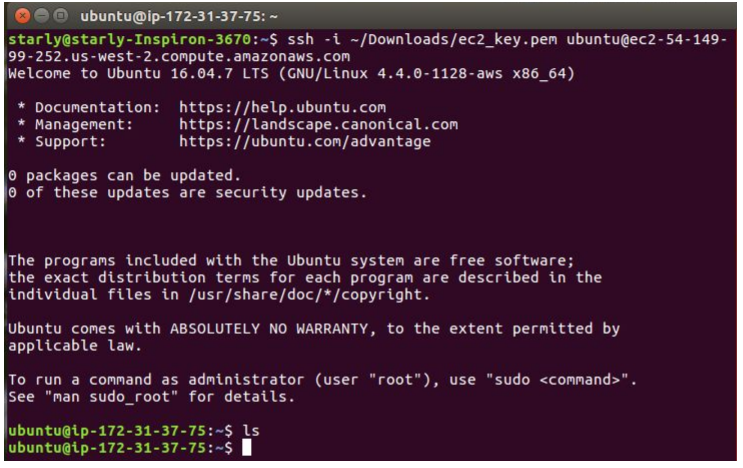


Accessing Virtual Machines on your Command Line

- Step 8: SSH into your instance







Install Docker and images on Virtual Machines

- Step 9: Install Docker
  - `curl -fsSL https://get.docker.com -o get-docker.sh`
  - `sudo sh get-docker.sh`
  - `sudo service docker start`
  - `sudo usermod -a -G docker ubuntu`
  - `sudo chmod 666 /var/run/docker.sock`

- Step 10: Download environment images (https://hub.docker.com/u/starlyxxx) or build images by Dockerfile (https://drive.google.com/drive/folders/1YZ2_ThqX46Yvy7dEt9KsG52pvQfhXBq1?usp=sharing)
  - CPU application environment image: `docker pull starlyxxx/dask-decision-tree-example`
  - GPU application environment image: `docker pull starlyxxx/horovod-pytorch-cuda10.1-cudnn7`
  - (Build from Dockerfile: `docker build -t <your-image-name> .`)

Download source data on Virtual Machines

- Step 11: Download ML applications and data.
  - For privacy, we store the application code and data on AWS S3
  - Install aws cli and set aws credentials (https://console.aws.amazon.com/iam/home?#security_credential):
    - `curl 'https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip' -o 'awscliv2.zip'`
    - `unzip awscliv2.zip`
    - `sudo ./aws/install`
    - `aws configure set aws_access_key_id <your-access-key>`
    - `aws configure set aws_secret_access_key <your-secret-key>`

Download source data on Virtual Machines

- Step 11: Download ML applications and data.
  - CPU application:
    - `aws s3 cp s3://kddworkshop/ML_based_Cloud_Retrieval_Use_Case.zip ./`
    - `(wget https://kddworkshop.s3.us-west-2.amazonaws.com/ML_based_Cloud_Retrieval_Use_Case.zip)`
    - `unzip ML_based_Cloud_Retrieval_Use_Case.zip`
  - GPU application:
    - `aws s3 cp s3://kddworkshop/MultiGpus-Domain-Adaptation-main.zip ./`
    - `(wget https://kddworkshop.s3.us-west-2.amazonaws.com/MultiGpus-Domain-Adaptation-main.zip)`
    - `aws s3 cp s3://kddworkshop/office31.tar.gz ./`
    - `(wget https://kddworkshop.s3.us-west-2.amazonaws.com/office31.tar.gz)`
    - `unzip MultiGpus-Domain-Adaptation-main.zip`
    - `tar -xzvf office31.tar.gz`

Running Docker Container for ML Computation

- Step 12: Run docker containers for CPU applications
  - Single CPU:
    - `docker run -it -v /home/ubuntu/ML_based_Cloud_Retrieval_Use_Case:/root/ML_based_Cloud_Retrieval_Use_Case starlyxxx/dask-decision-tree-example:latest /bin/bash`
  - Multi-CPUs:
    - `docker run -it --network host -v /home/ubuntu/ML_based_Cloud_Retrieval_Use_Case:/root/ML_based_Cloud_Retrieval_Use_Case starlyxxx/dask-decision-tree-example:latest /bin/bash`

-v: shared filesystems
-it /bin/bash: interactive processes with bash
--network host: container shares the host's networking namespace

Running Docker Container for ML Computation

- Step 13: Run docker containers for GPU applications
  - Single GPU:
    - `nvidia-docker run -it -v /home/ubuntu/MultiGpus-Domain-Adaptation-main:/root/MultiGpus-Domain-Adaptation-main -v /home/ubuntu/office31:/root/office31 starlyxxx/horovod-pytorch-cuda10.1-cudnn7:latest /bin/bash`
  - Multi-GPUs:
    - Add primary worker's public key to all secondary workers' <~/.ssh/authorized_keys>
    - `sudo mkdir -p /mnt/share/ssh && sudo cp ~/.ssh/* /mnt/share/ssh`
    - Primary worker VM: `nvidia-docker run -it --network=host -v /mnt/share/ssh:/root/.ssh -v /home/ubuntu/MultiGpus-Domain-Adaptation-main:/root/MultiGpus-Domain-Adaptation-main -v /home/ubuntu/office31:/root/office31 starlyxxx/horovod-pytorch-cuda10.1-cudnn7:latest /bin/bash`
    - Secondary workers VM: `nvidia-docker run -it --network=host -v /mnt/share/ssh:/root/.ssh -v /home/ubuntu/MultiGpus-Domain-Adaptation-main:/root/MultiGpus-Domain-Adaptation-main -v /home/ubuntu/office31:/root/office31 starlyxxx/horovod-pytorch-cuda10.1-cudnn7:latest bash -c "/usr/sbin/sshd -p 12345; sleep infinity"`

ML Computation on Docker Container

- Step 14: Run ML CPU application:
  - Single CPU:
    - `cd ML_based_Cloud_Retrieval_Use_Case/Code && /usr/bin/python3.6 ml_based_cloud_retrieval_with_data_preprocessing.py`
  - Multi-CPUs:
    - Run dask cluster on both VMs in background:
      - VM 1: `dask-scheduler &` `dask-worker <your-dask-scheduler-address> &`
      - VM 2: `dask-worker <your-dask-scheduler-address> &`
    - One of VMs:
      - `cd ML_based_Cloud_Retrieval_Use_Case/Code && /usr/bin/python3.6 dask_ml_based_cloud_retrieval_with_data_preprocessing.py <your-dask-scheduler-address>`

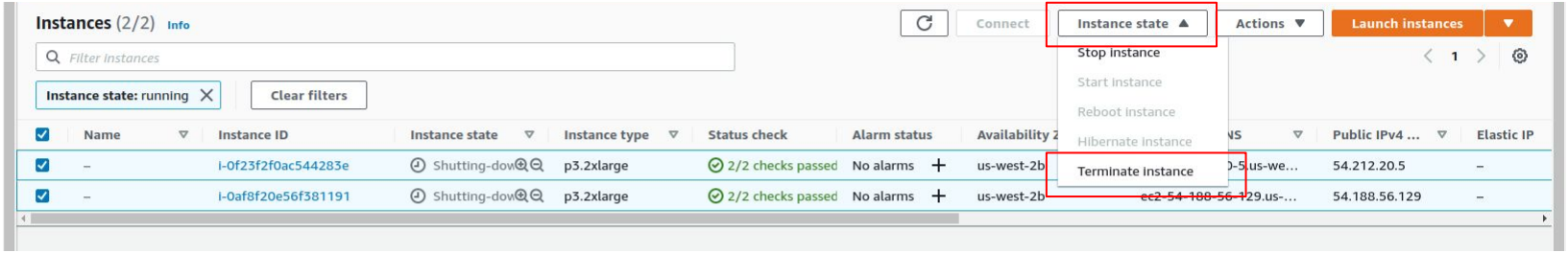
ML Computation on Docker Container

-np: number of processes
-H: list of hosts
-p: ssh port on all the hosts

- Step 15: Running ML GPU application:
  - Single GPU:
    - `cd MultiGpus-Domain-Adaptation-main`
    - `horovodrun --verbose -np 1 -H localhost:1 /usr/bin/python3.6 main.py --config DeepCoral/DeepCoral.yaml --data_dir ../office31 --src_domain webcam --tgt_domain amazon`
  - Multi-GPUs:
    - Primary worker VM:
      - `cd MultiGpus-Domain-Adaptation-main`
      - `horovodrun --verbose -np 2 -H <machine1-address>:1,<machine2-address>:1 -p 12345 /usr/bin/python3.6 main.py --config DeepCoral/DeepCoral.yaml --data_dir ../office31 --src_domain webcam --tgt_domain amazon`

Terminate Virtual Machines

- Step 16: Terminate all VMs



Distributed ML on AWS Cloud: Demo