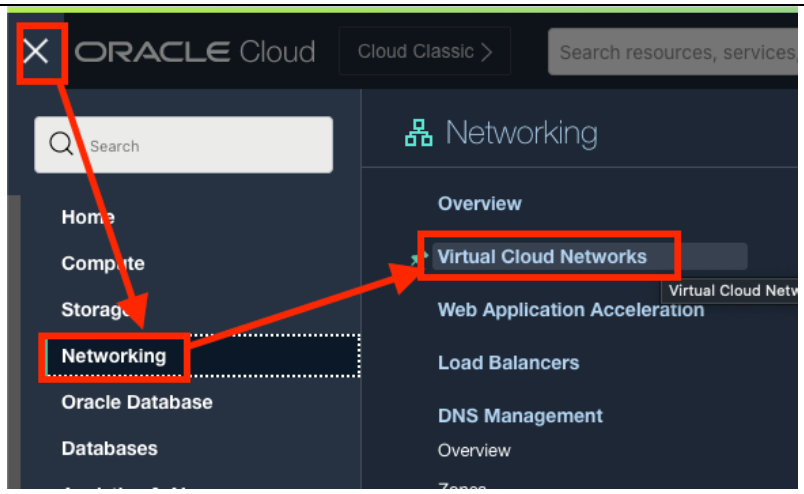
Distributed Training with OCI DS Jobs

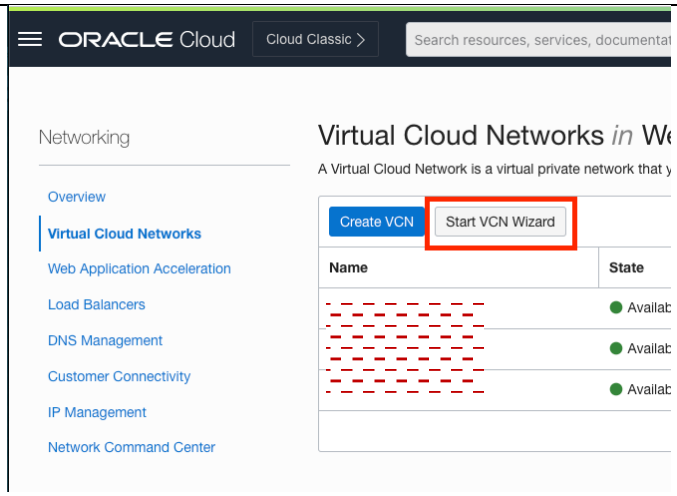
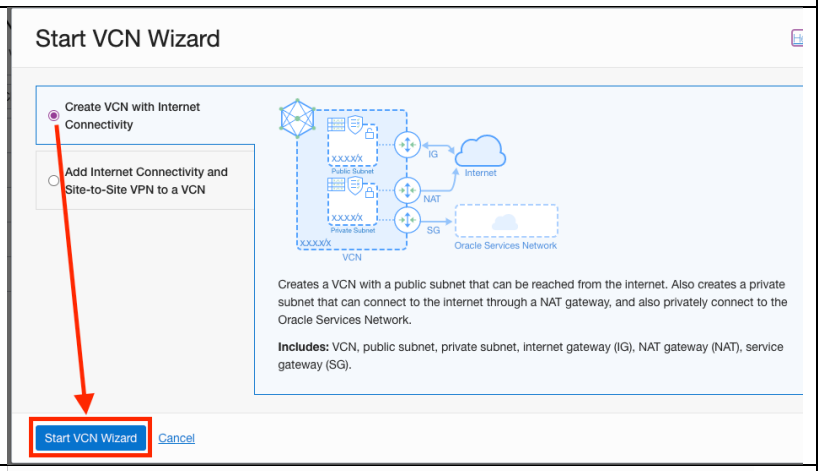
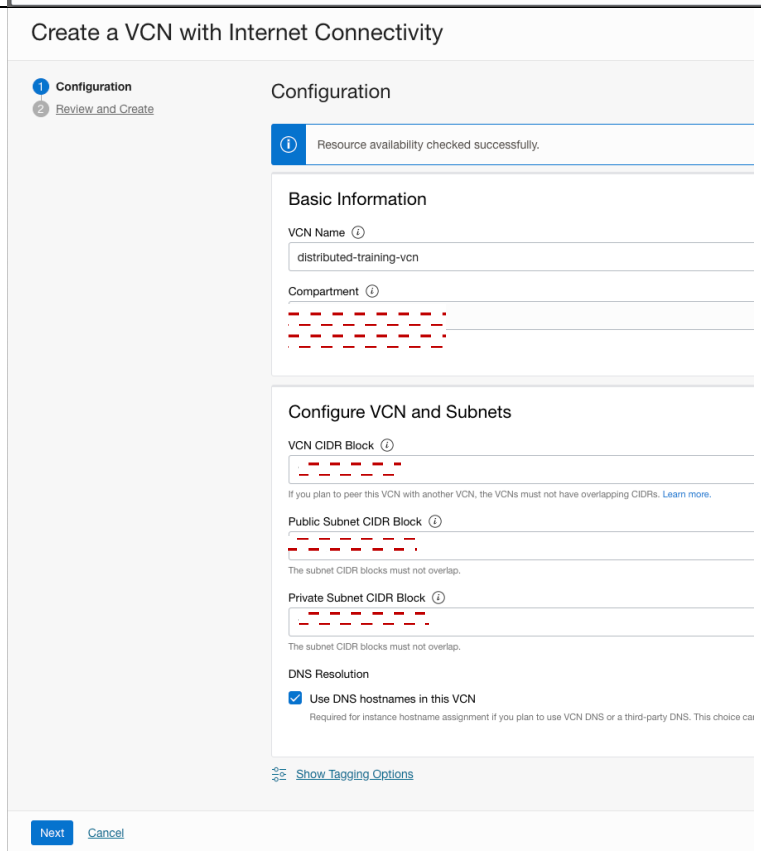
Referenced Documentation: https://github.com/oracle-samples/oci-data-science-ai-samples/tree/master/distributed_training/notebooks

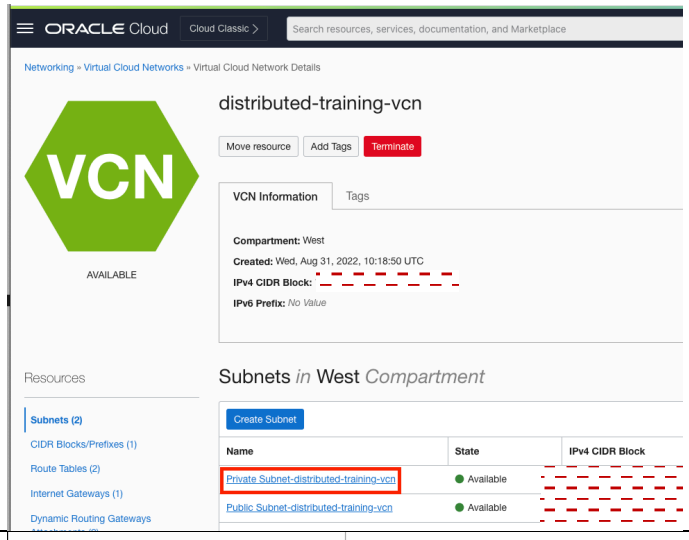
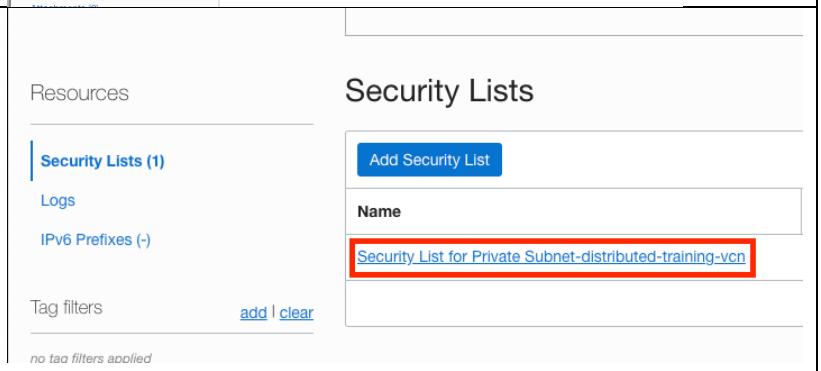
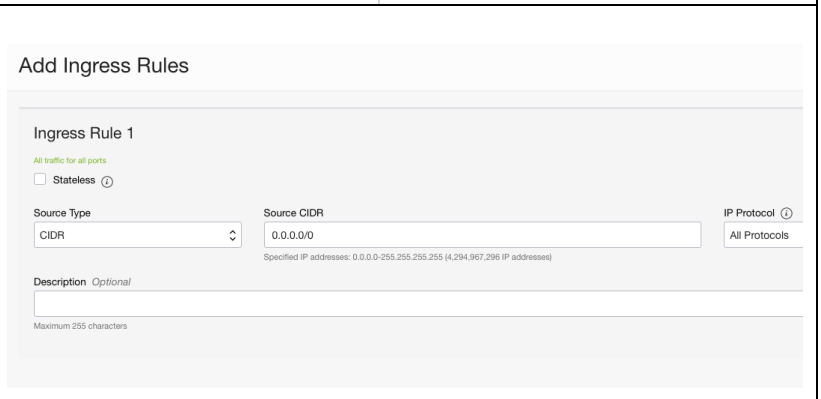
In this document, we will only be looking at the Horovod TensorFlow Distributed Training Example.

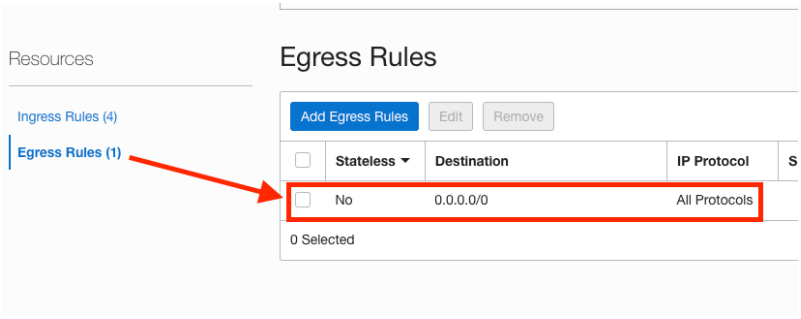
Here are the key pre-requisites that you would need to execute before you can proceed to building a distributed workload docker image and execute it using ***odsc-jobs***.

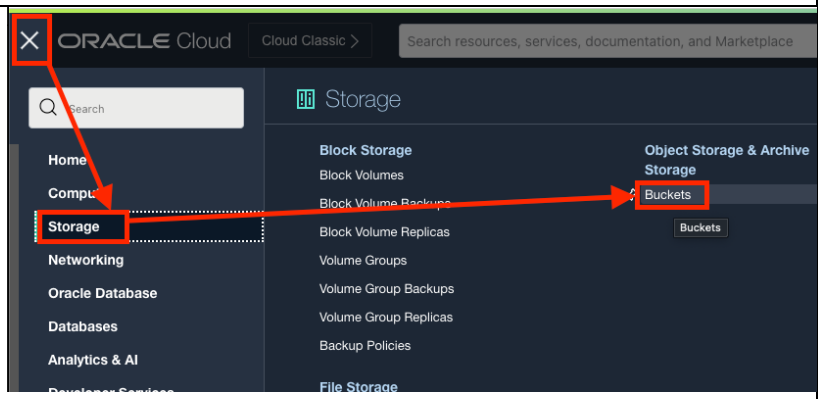
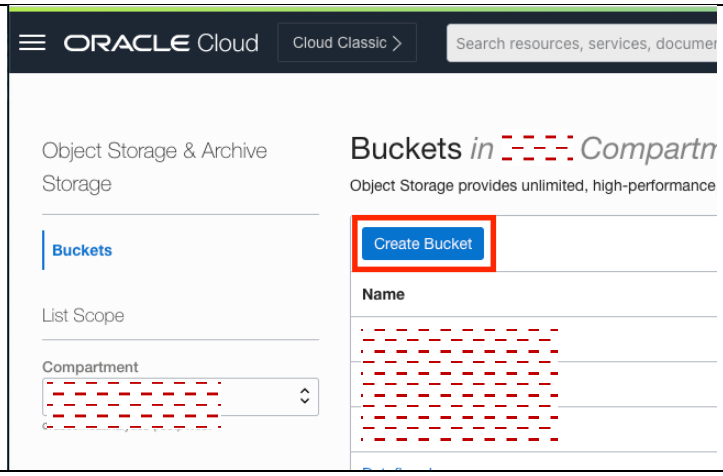
1. Configure Network. Required for the inter node communication.
2. OCI policies. Required for accessing OCI services by the distributed job.
3. Install ads-opctl. Required for packaging training script and launching odsc distributed job.

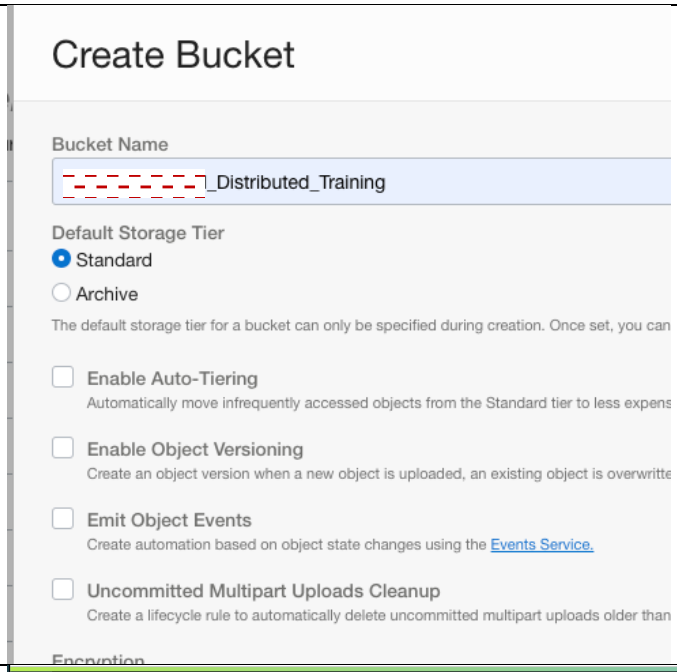
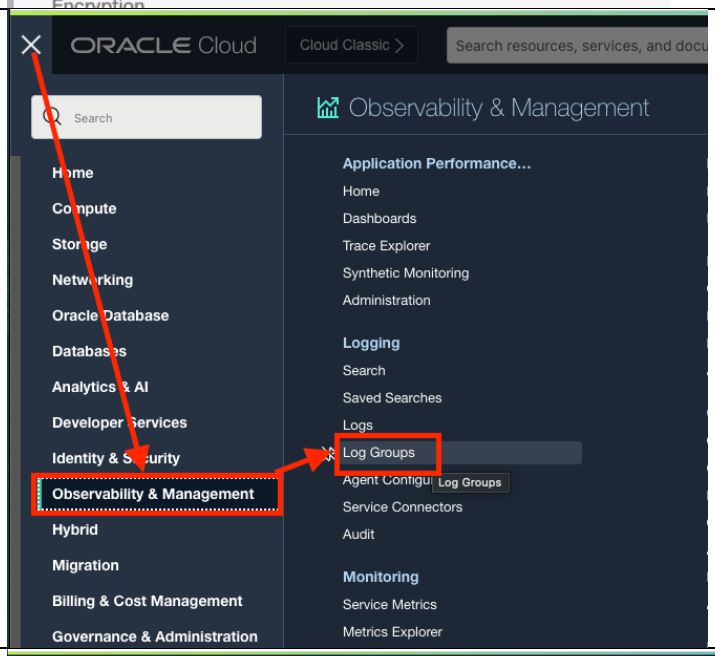
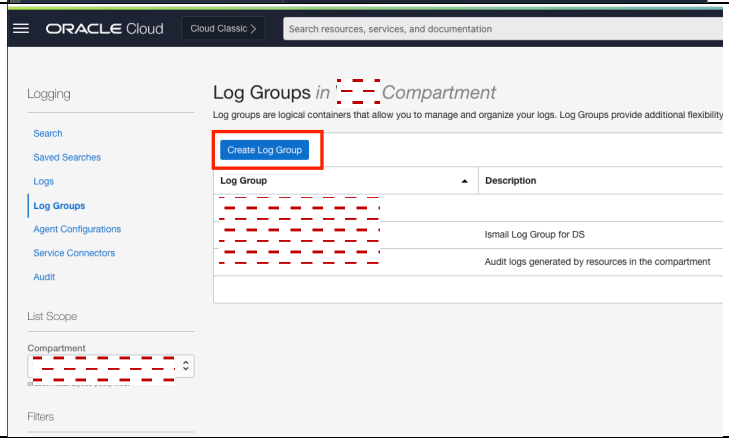
Networking Pre-Requisite	
Step	**Screenshot**
Create a new VCN with a Private Subnet, that allows all traffic within the subnet. This will allow the nodes to all communicate with each other. ***OCI Console Menu > Networking > Virtual Cloud Networks.***	

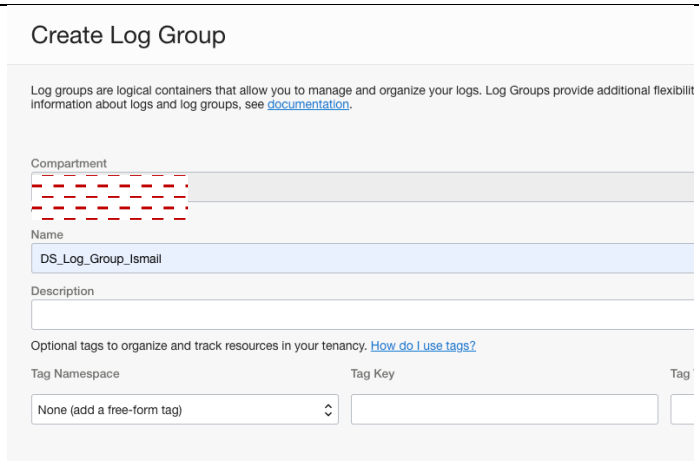
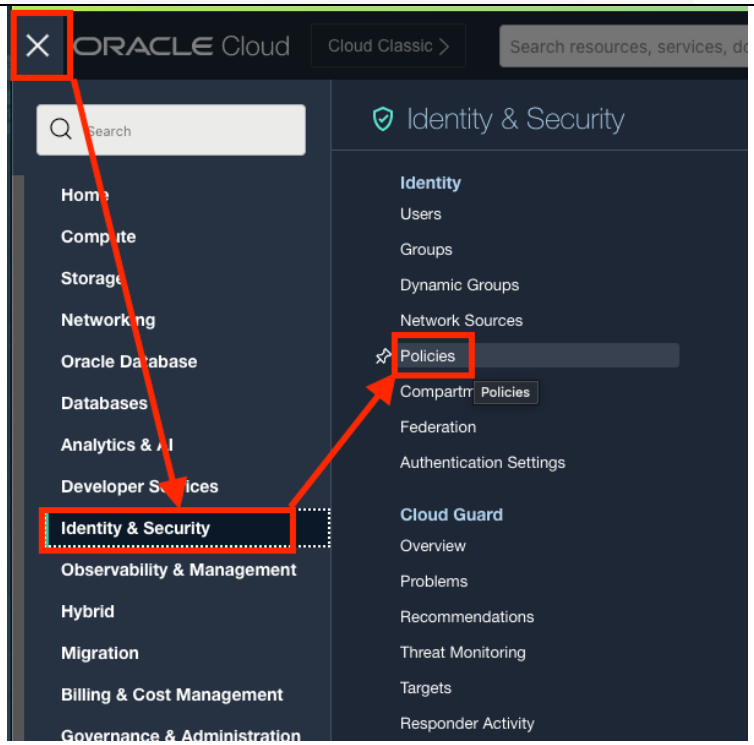
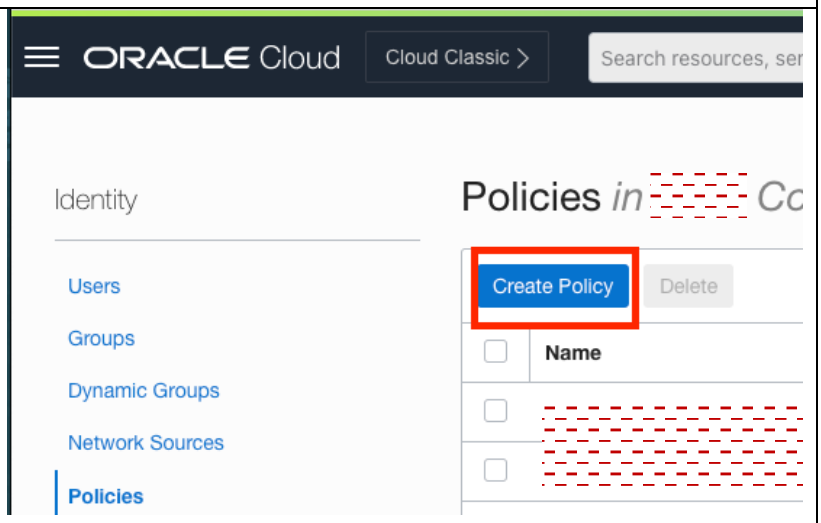
Click on ***Start VCN Wizard***.	
Select ***Create VCN with Internet Connectivity***. Click on ***Start VCN Wizard***.	
Enter a ***Name***. Leave defaults or enter CIDR Block ranges for the VCN, Public Subnet and Private Subnet. Click ***Next***. Review Summary Page. Click ***Create***.	

We need to now allow all communication between the nodes in the Subnet. ***Click*** on the ***Private Subnet*** that has just been created.	
Select the ***Private Subnet Security List***.	
We will add a new Ingress Rule to Allow communication between all IPs and Ports within the Subnet. Click ***Add Ingress Rule***.	
Add a Rule with ***Source CIDR as 0.0.0.0/0*** (This can also be locked down to the Subnet itself, however for simplicity we will leave it as such). Set IP Protocol as ***All Protocols.*** Click ***Add Ingress Rules***.	

Double Check the Egress Rules to ensure the same Rule is set up here (Destination 0.0.0.0/0 All Protocols). This should have been created by default anyhow.	

Set Up OCI Services & Configure OCI Policies	
Steps	**Screenshot**
We will create an Object Storage Bucket as a Staging area and to store outputs in when the job runs. ***OCI Menu > Storage Buckets.***	
Click ***Create Bucket***	

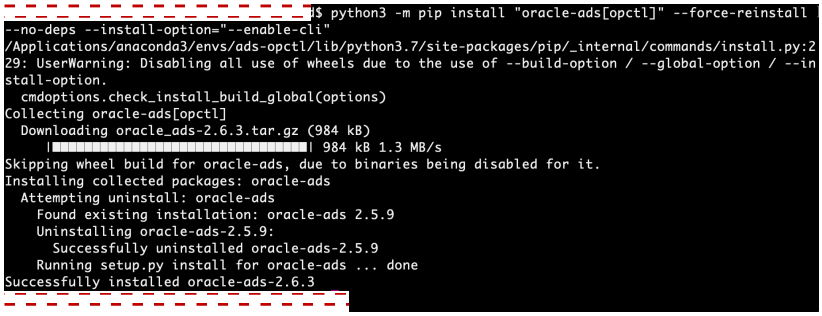
Enter a ***Bucket Name*** and Click on ***Create***.	
When we come to later running jobs within OCI Data Science Service, it is a good idea to set up logging in order to understand how the job has run. From the OCI Console ***visit Menu > Observability & Management > Log Groups***	
Click on ***'Create Log Group'*** to create a new log group to store our logs in.	

Enter a name for your Log Group and click ***'Create Log Group'*** down at the bottom.	
We will create multiple policies to allow users and our Data Science Dynamic Group access to Networking Components, Data Science Components, OCI Container Registry, Logs, Metrics and Object Storage. ***OCI Menu > Identity & Security > Policies.***	
Click on ***Create Policy.***	

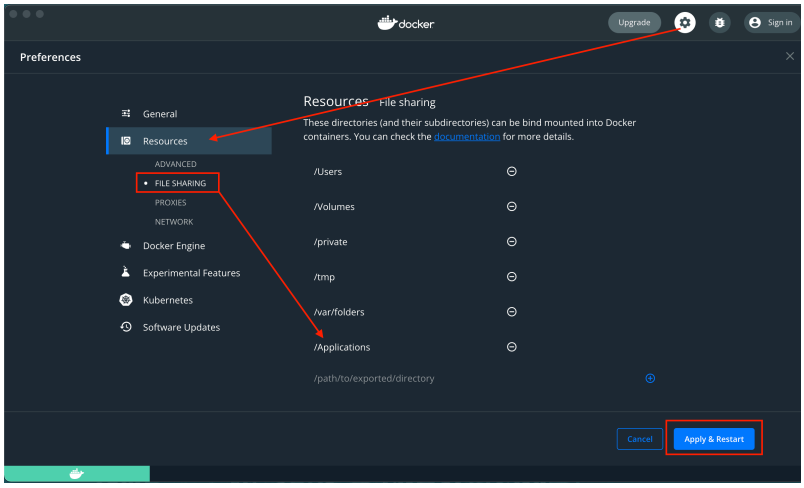
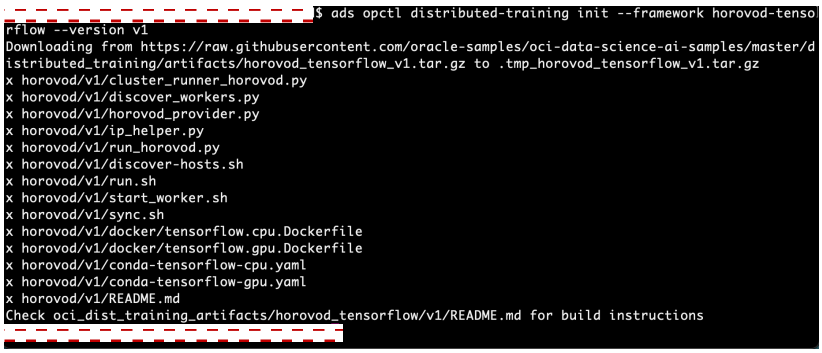
Give the Policy a ***Name***, ***Description*** and enter the following policies, under ***Manual Editor***. **NOTE:** This assumes you have already created a ***Group*** of Users, have a ***Dynamic Group*** set up for your Data Science Resources Click ***Create***.	***Allow group <group> to manage repos in compartment <compartment>*** ***Allow group <group> to manage data-science-jobs in compartment <compartment>*** ***Allow group <group> to manage data-science-job-runs in compartment <compartment>*** ***Allow group <group> to use virtual-network-family in compartment <compartment>*** ***Allow group <group> to manage log-groups in compartment <compartment>*** ***Allow group <group> to use logging-family in compartment <compartment>*** ***Allow group <group> to use metrics in compartment <compartment>*** ***Allow dynamic-group <dynamic-group> to read repos in compartment <compartment>*** ***Allow dynamic-group <dynamic-group> to use data-science-family in compartment <compartment>*** ***Allow dynamic-group <dynamic-group> to use virtual-network-family in compartment <compartment>*** ***Allow dynamic-group <dynamic-group> to use log-groups in compartment <compartment>*** ***Allow dynamic-group <dynamic-group> to use logging-family in compartment <compartment>*** ***Allow group <group> to manage objects in compartment <compartment> where all {target.bucket.name=<object_storage_bucket_name>}*** ***Allow dynamic-group <dynamic-group> to manage objects in compartment <compartment> where all {target.bucket.name=<object_storage_bucket_name>}*** ***Allow dynamic-group <dynamic-group> to manage buckets in compartment <compartment> where all {target.bucket.name=<object_storage_bucket_name>}***

<table>
<tr><th colspan="2">Install ads-opctl and Docker</th></tr>
<tr><th>Steps</th><th>Screenshot</th></tr>
<tr><td>This assumes you already have Python3 installed locally on your machine.

I will install ADS within a Python3.7 Conda Environment I have already created.
python3 –m pip install "oracle–ads[opctl]"

python3 –m pip install "oracle–ads[opctl]" ––force–reinstall ––no–deps ––install–option="––enable–cli"</td><td> 

Need to run both commands on the left.</td></tr>
<tr><td>Download Docker and Run Installation from: https://docs.docker.com/get-docker</td><td> 
</td></tr>
</table>

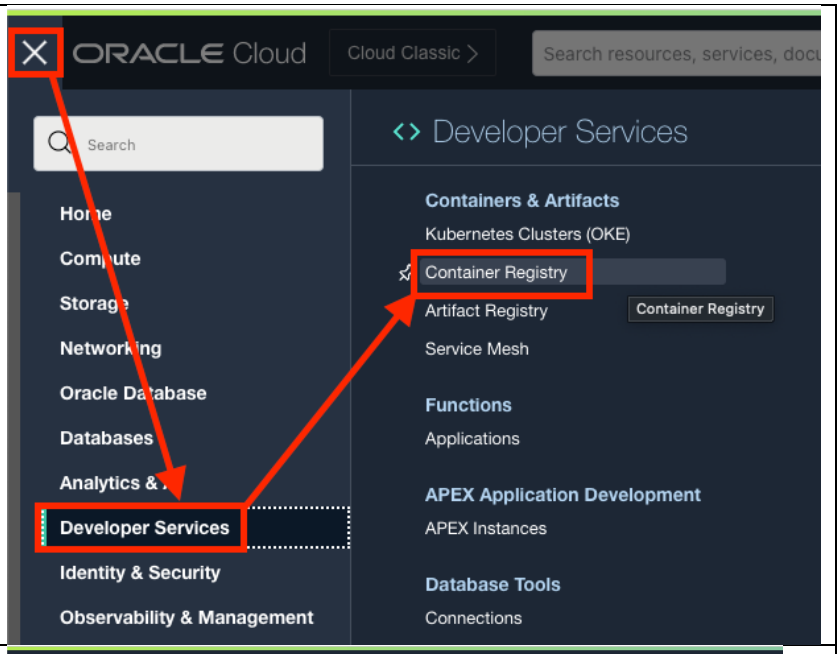
Creating, installing and publishing conda packs involves mounting folders to docker images. Please make sure that you give docker desktop access to file sharing of the parent folder for conda. ***Visit Settings > Resources > File Sharing*** **Add your parent folder for conda** in my case /Applications**.** ***Click Apply & Restart.***	 This is more a pre-requisite for Jobs and does not have to be done, but it is good practise.
Using the Terminal, navigate to a folder where you wish to download the related docker artifacts for distributed TF training. ***cd ~/Distributed_Training*** Using the ads-opctl, download and initialize the relevant distributed training framework. ***ads opctl distributed-training init --framework horovod-tensorflow --version v1***	

Prepare Docker Image	
Steps	**Screenshots**
All the docker image related artifacts are located under: ***oci_dist_training_artifacts/horovod/v1/***	Continued…
Horovod provides support for Pytorch and Tensorflow. Within these frameworks, there are two separate docker files, for cpu and gpu.	Upload your Tensorflow training script (***train.py***) within your current directory on your local machine as this will then be copied into the ***/code*** folder inside docker image once built.
Choose the docker file and conda environment files based on whether you are going to use Pytorch or Tensorflow with either cpu or gpu.	**Note**: Whenever you change the code, you have to build, tag and push the image to repo. If you change the tag, it needs to be updated inside the cluster definition yaml later on.
The instruction assumes that you are running this within the folder where you ran ***ads opctl distributed-training init --framework horovod-<pytorch|tensorflow>***	The required python dependencies are provided inside the conda environment file ***oci_dist_training_artifacts/horovod/v1/conda-<pytorch|tensorflow>-<cpu|gpu>.yaml.*** If your code requires additional dependency, update this file. **Note**: While updating ***conda-<pytorch|tensorflow>-<cpu|gpu>.yaml*** do not remove the existing libraries. You can append to the list.

We will now create an OCI Container Registry to store our Docker Image in when we push from our local repo to the cloud.

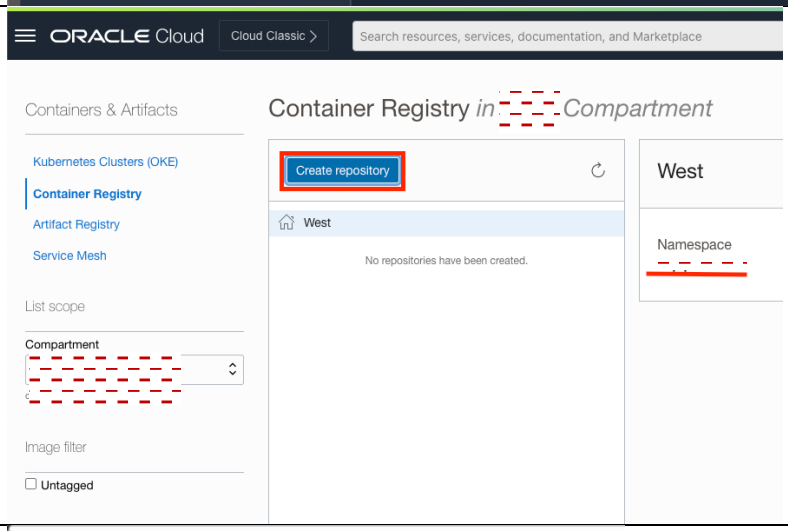
This pushed Docker Image will be used in the Job Run.

OCI Menu > Developer Services > Container Registry.

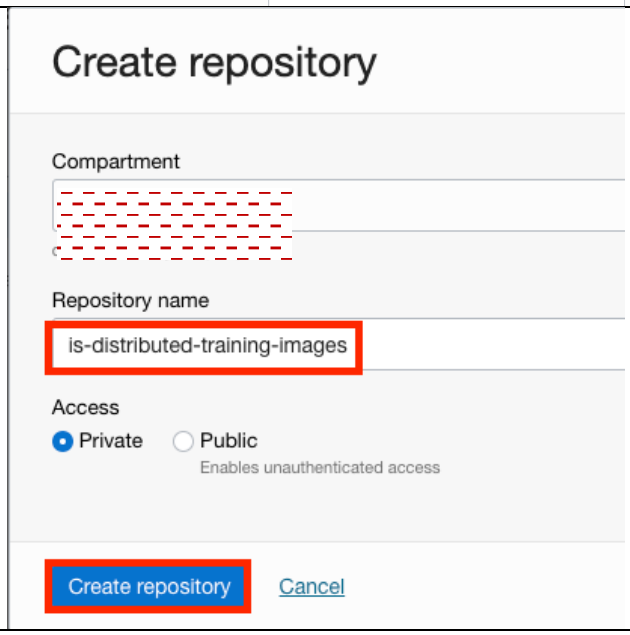


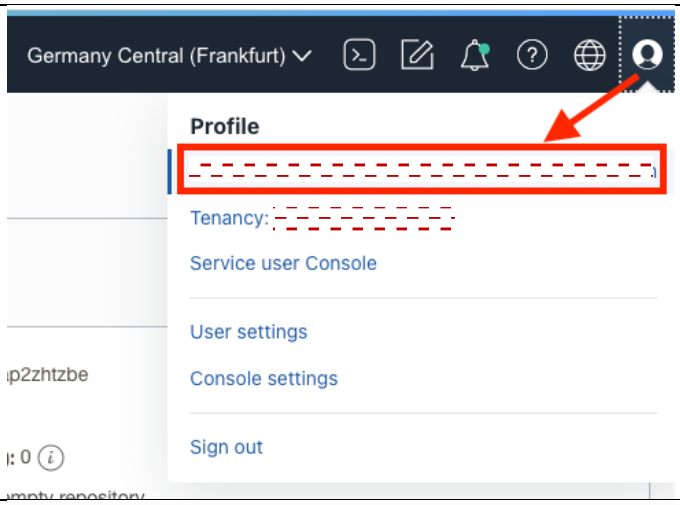
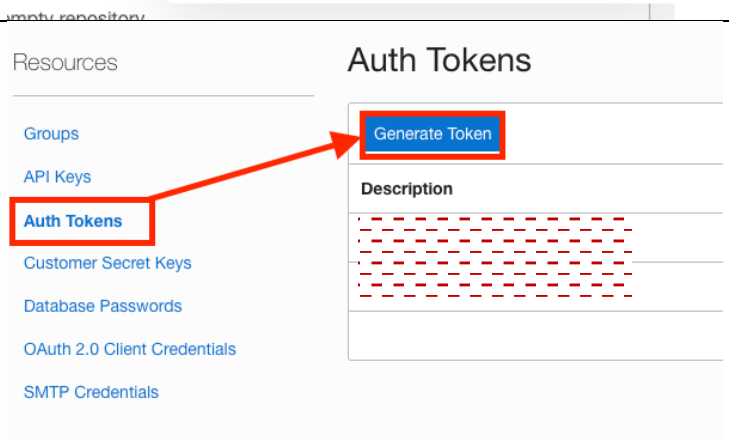
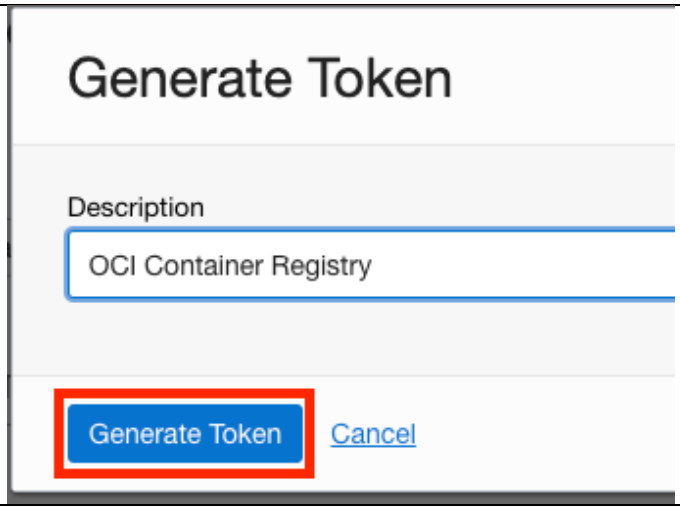
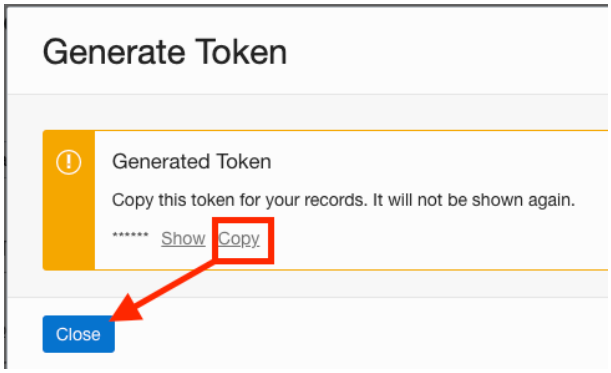
Make note of the Namespace to your right, you will need this later.

Click on ***Create Repository***.



Give it a ***Name*** and Click ***Create RepositoryV***.



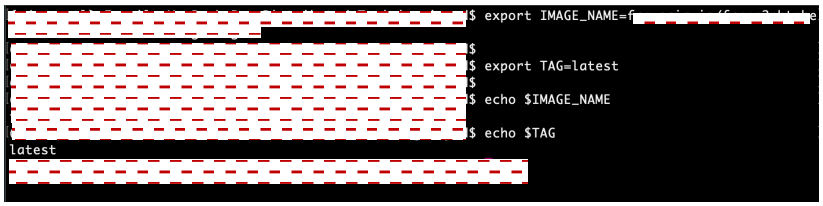
We then need to Generate Auth Token for Authentication from our Local Docker to Container Registry. Click on your ***Profile Icon*** then your ***Username***.	
Navigate to ***Auth Tokens*** and Click on ***Generate Token.***	
Give it a ***Description*** and Click on ***Generate Token***.	
Copy the ***Token*** as it will be needed late to authenticate. You can't view this again.	

Ensure your local Docker installation is running, before doing the following tasks. In a terminal window on the client machine running Docker, log in to Container Registry by entering ***docker login <region-key>.ocir.io, where <region-key>*** corresponds to the key for the Container Registry Region you're using.	 ***docker login fra.ocir.io for Frankfurt region***
When prompted for a username, enter your username in the format ***<tenancy-namespace>/<username>*** If your tenancy is federated with Oracle IDCS, use the format ***<tenancy-namespace>/oracleidentitycloudservice/<username>***	
When prompted for a password, **enter** the ***Auth Token*** you copied earlier.	

We will set some environment variables for the Docker Image Name and Tag

export IMAGE_NAME=fra.ocir.io/<name-space>/<repo-name>

export TAG=latest



Format of the image name must be ***<region-key>.ocir.io/<tenancy-namespace>/<repo-name>:<tag>*** where:

- ***<region-key>*** is the key for the Container Registry region you're using. For example, fra. See Availability by Region.
- ocir.io is the Container Registry name.
- ***<tenancy-namespace>*** is the auto-generated Object Storage namespace string of the tenancy that owns the repository to which you want to push the image (as shown on the **Tenancy Information** page).
- ***<repo-name>*** is the name of the target repository to which you want to push the image (for example, project01/acme-web-app).
- ***<tag>*** is an image tag you want to give the image in Container Registry (for example, version2.0.test).

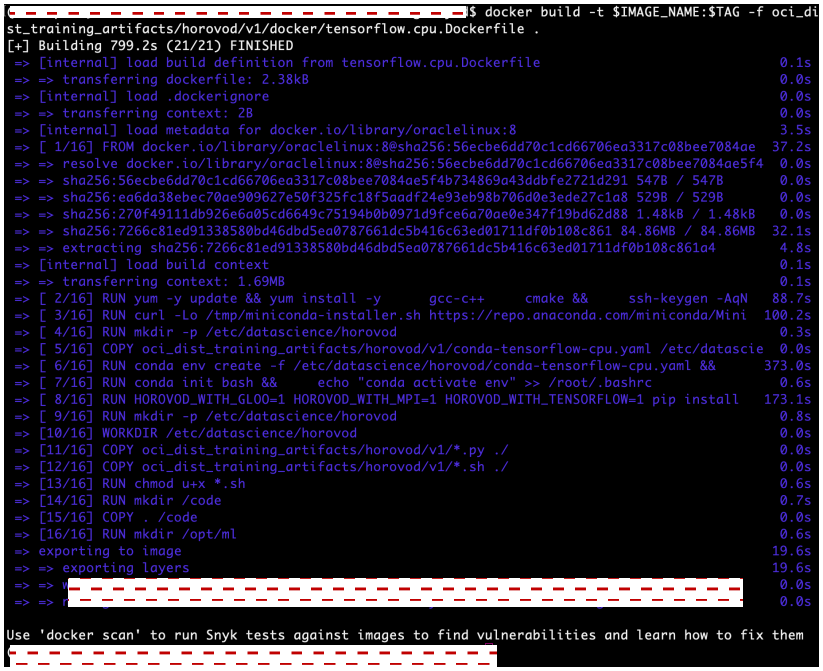
We can then build the Docker Image:

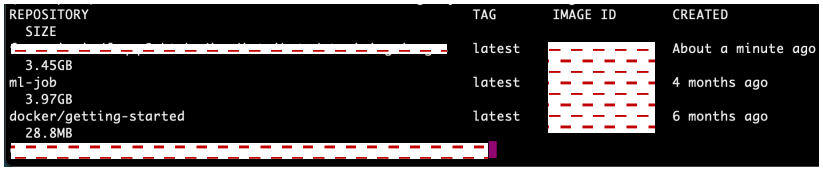
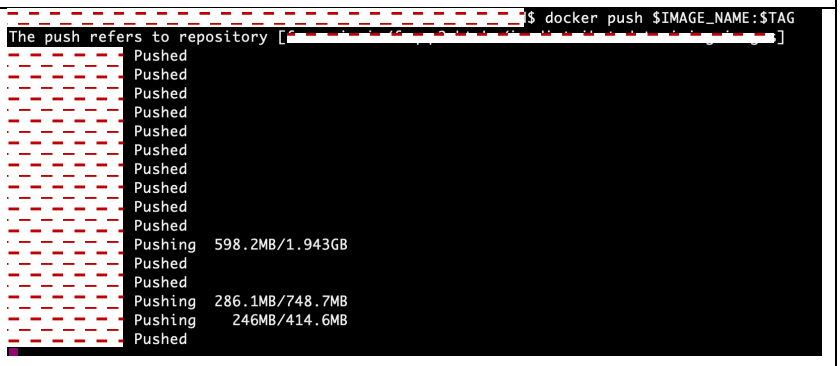
docker build -t $IMAGE_NAME:$TAG \ -f oci_dist_training_artifacts/horovod/v1/docker/tensorflow.cpu.Dockerfile .

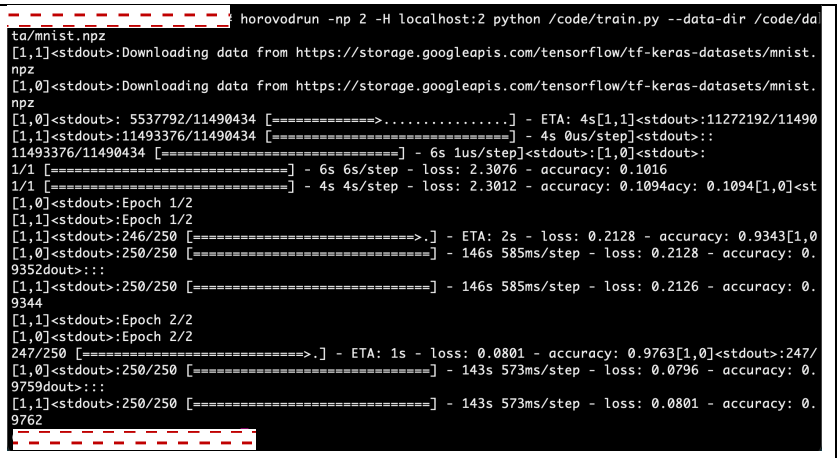
Include the (dot) at the end of the command.

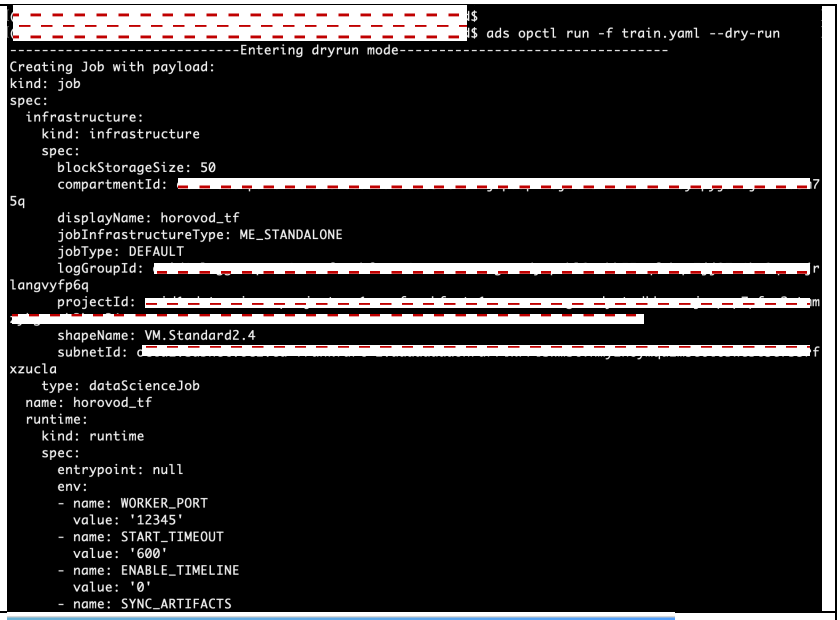
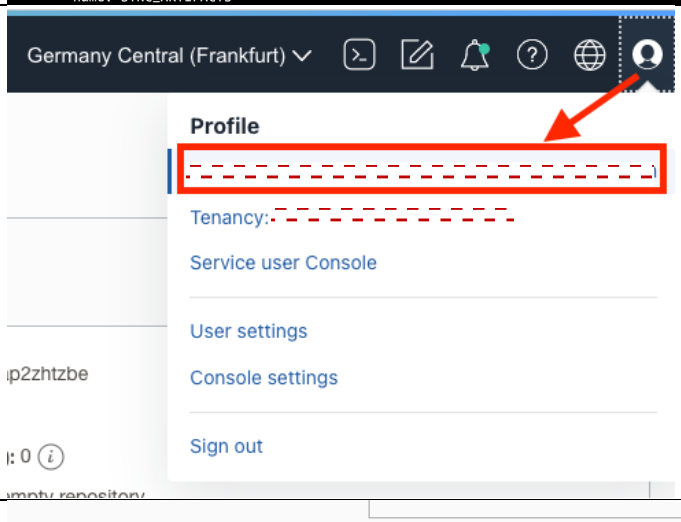
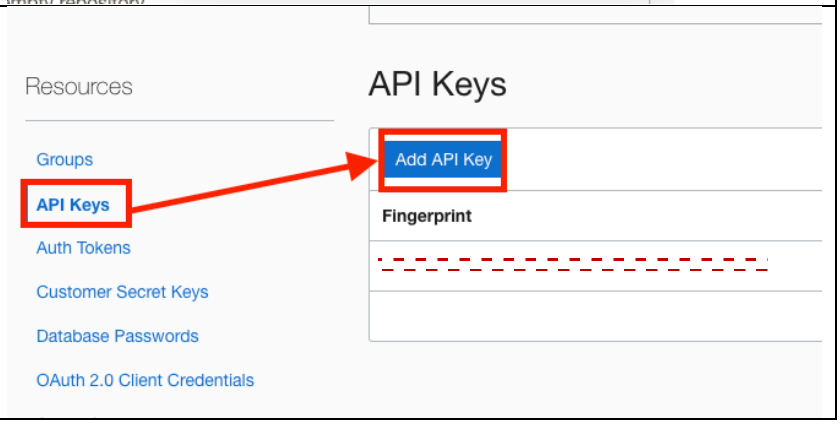
Ensure we are pointing to the correct Docker File dependant of CPU or GPU

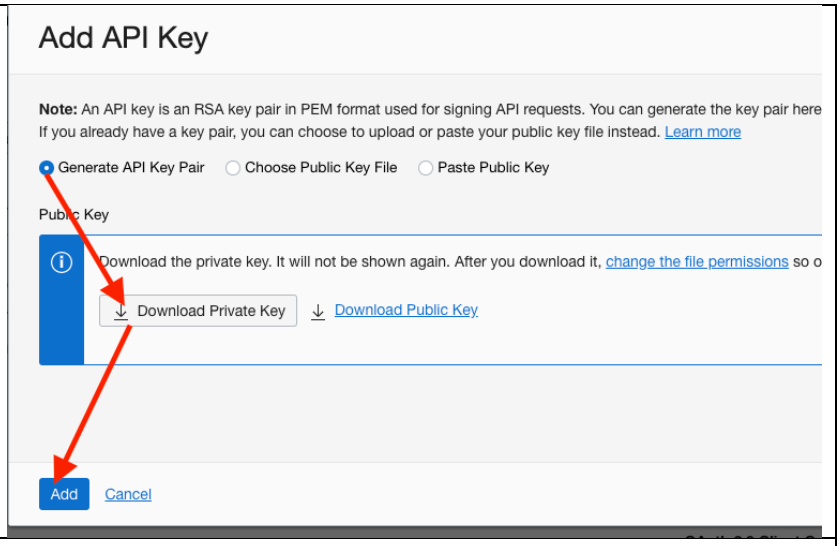
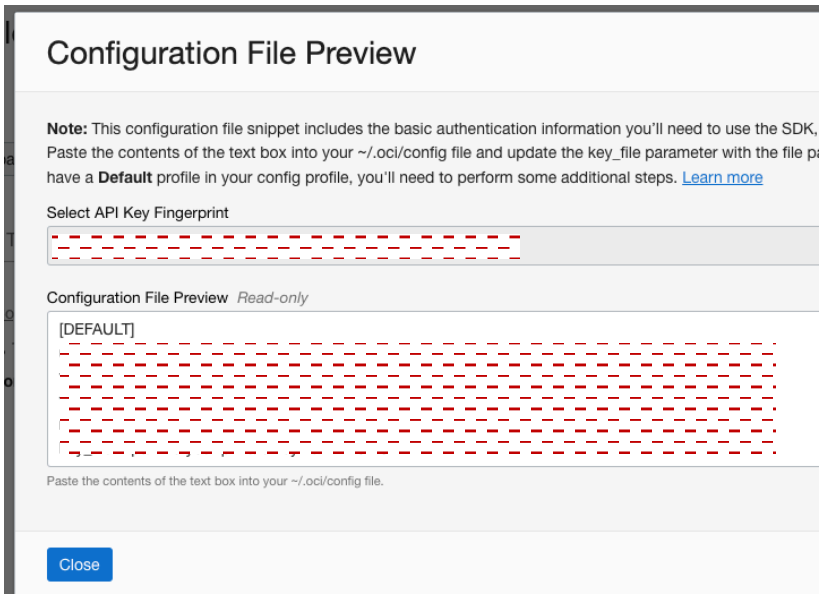
This will take a while to build.



Once complete, we can ***list*** our Docker Images ***docker image ls***	
Push the local Docker Image to the OCI Registry. ***docker push $IMAGE_NAME:$TAG*** Where ***IMAGE_NAME*** refers to the repo you want to push to.	
We can then view our Image in the OCI Registry.	
Before triggering the job run, you can test the docker image and verify the training code, dependencies etc. You can do this using a local stand-alone docker run For a stand-alone docker run, start a docker container with bash entry point. ***docker run --rm -it --privileged --entrypoint bash $IMAGE_NAME:$TAG***	

You have now entered your docker container. Now test your training script with the following command. ***horovodrun -np 2 -H localhost:2 python /code/train.py --data-dir /code/data/mnist.npz***	
Once done, exit the container with the exit command. ***exit***	
Create a ***train.yaml*** local file which will contain the job definition. Edit the file and adjust parameters as necessary. E.g. **Project ID, Compartment, Log Group, Docker Image, etc…** An example file is contained in the same directory as this Guide.	

This will print/validate the Job and Job run configuration without launching the actual job. ***ads opctl run -f train.yaml --dry-run***	
Before you submit the job to OCI DS Jobs, you need to add you Private Key File and Config file. Click on your ***Profile*** then ***Username***	
Navigate to ***API Keys > Add API Key***	

Generate API Key Pair*.** ***Download Private Key. ***Click Add.***	
Copy the config file and save locally along with the private key to ~/.oci. ***Ensure to update the path to the private key in the config file.***	

Next, we can set up the default ADS OPCTL config by running:

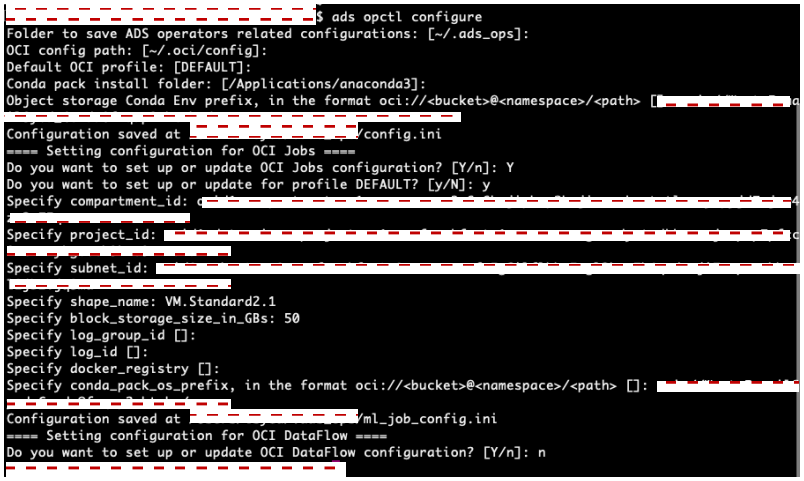
```
ads opctl configure
```

The above is normally done for submitting standard jobs so not all info is needed.

You will be prompted to enter the following info:

- OCI Config Path
- Default OCI Profile
- Conda Install Folder
- Conda Pack OS Prefix
- Compartment OCID
- Data Science Project OCID
- Data Science Subject OCID
- Job Run Shape
- Block Storage GB
- Log Group OCID
- Log OCID
- Docker Registry ID
- Conda Pack OS Prefix

Fill in the relevant information.



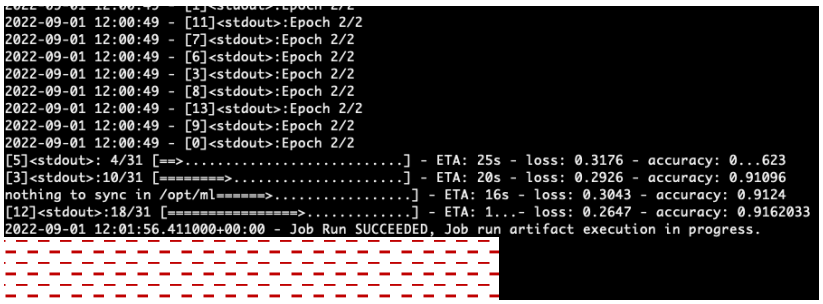
This will create two files in **/User/<username>/.ads_ops**

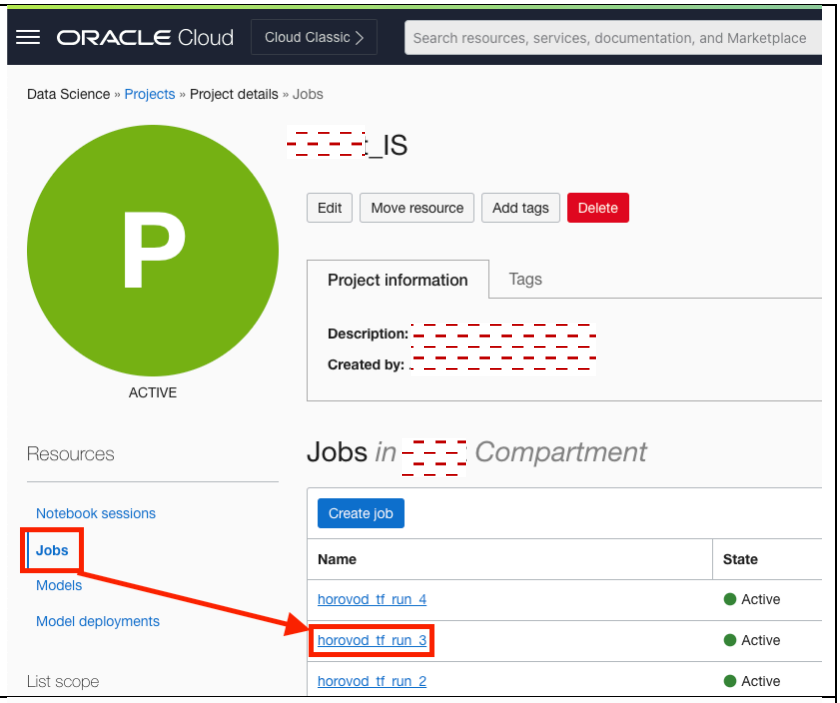
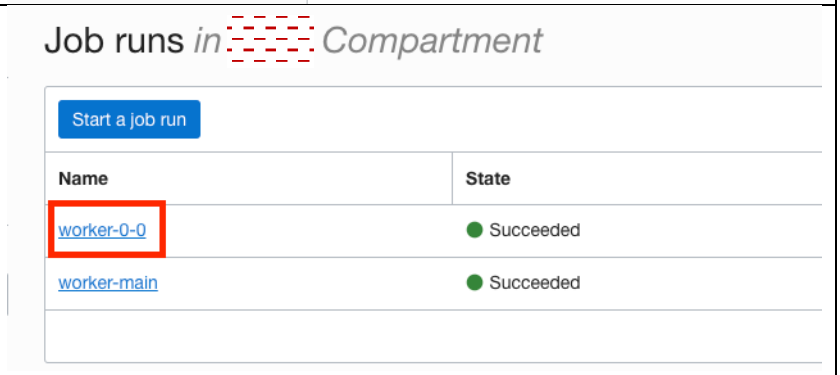
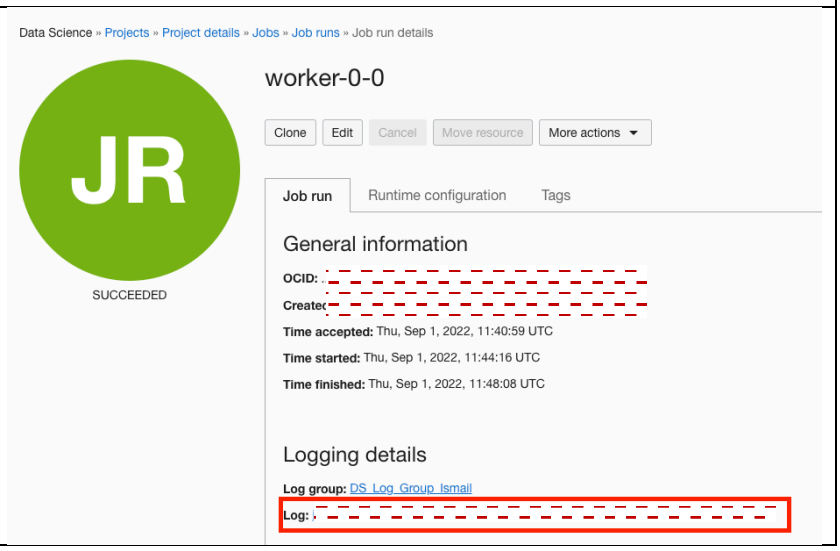
We can change directory to see the config files created.

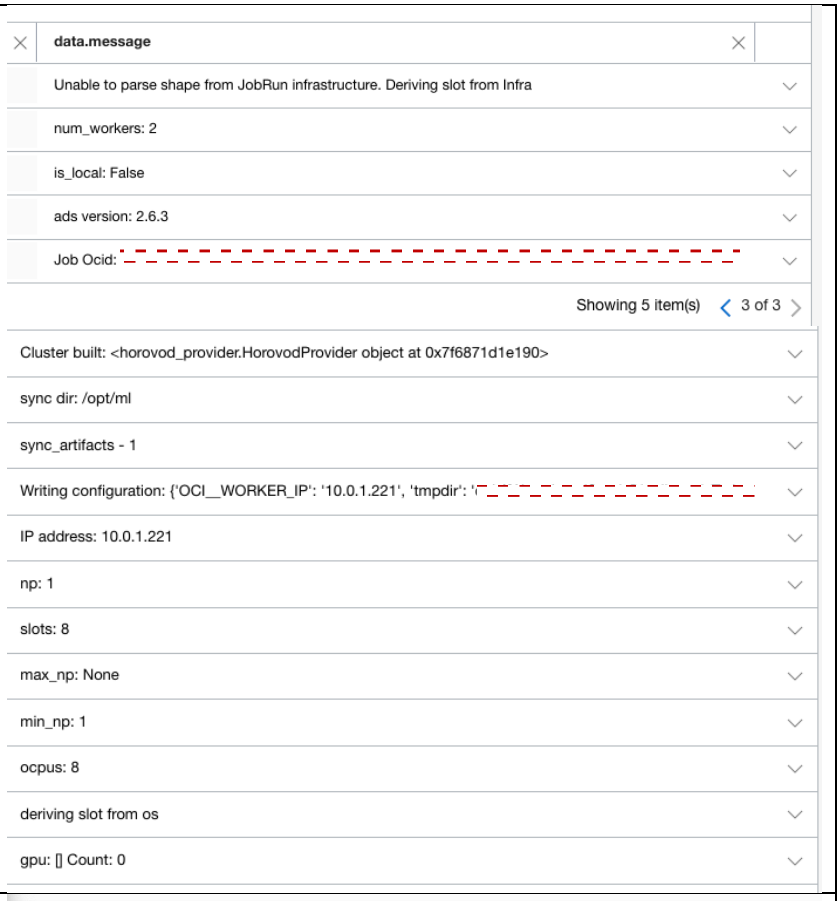
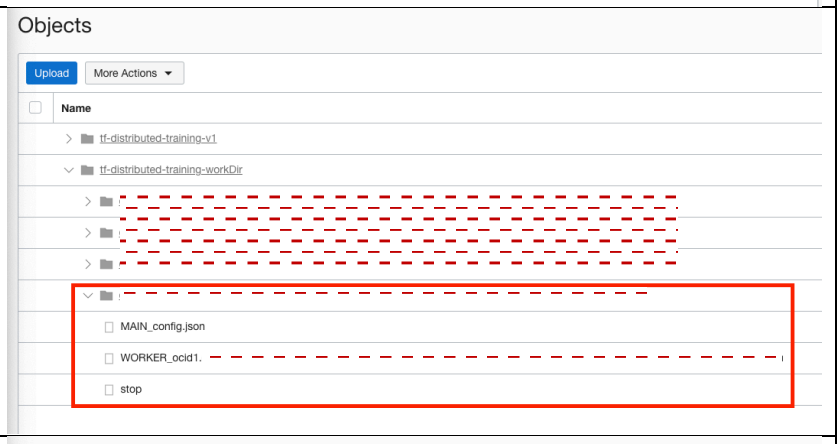
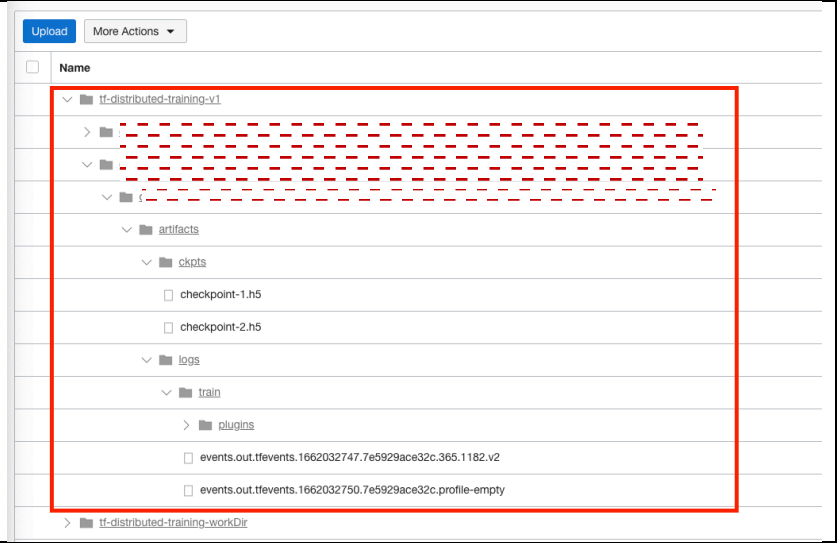
```
cd /Users/<username>/.ads_ops
```

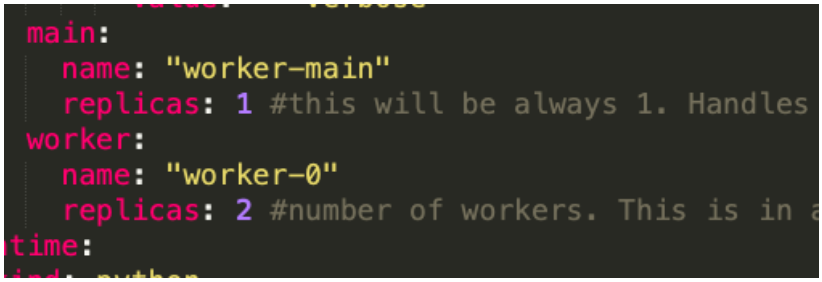
We can print out the first config file:

```
cat config.ini
```

We can now submit the training job to OCI DS Jobs and we can also save the output to ***info.yaml***. You could use this yaml for checking the runtime details of the cluster - scheduler ip ***ads opctl run -f train.yaml | tee info.yaml***	
This command will stream the log from logging infrastructure that you provided while defining the cluster inside train.yaml in the example above. ***ads opctl watch <job runid>*** The Job Status and Any Outputs will be displayed as the Job Runs.	 
Check runtime configurations ***ads opctl distributed-training show-config -f info.yaml***	

Have a look in the OCI console to see the jobs succeeded. ***OCI Menu > Analytics & AI > Data Science > Your Project > Jobs > Select your Distributed Job.***	
We can see the Job Runs for both the Scheduler and Worker Nodes. Both have Succeeded. Let's Click on the Worker Node – worker-0-0	
Let's view the Log created within OCI Logging.	

We can see information such as: • ADS Version • Num Nodes • Num OCPU • Location of Config Files	
We can also go view our Work Directory within Object Storage where the Config Files are: These can be downloaded.	
We can check our Output Logs and Model Checkpoints also stored within our Object Storage.	

If you want, you can edit the ***train.yaml*** file to increase the number of worker nodes. In this case I increased to 2 worker Nodes.	
You can then re-submit the job and see results in console. We now have 1 Scheduler Node and 2 Worker Nodes.