[High Level Design] Using only Customer Container Registry

1. PDD

ADS Distributed Operators PDD

2. Dependency

- Customer has environment to build docker images
- Customer has access to publish docker image in ocir container registry
- ML Jobs support BYOD
- ML Jobs supports for Mounted file system (Required for Horovod)
- ML Jobs Firewall does not block the required port numbers

3. Prerequisite

- User will create a regional private subnet with access to NAT GW or Service GW
- User will update the security list to open ports as required by the distributed framework - The port list will be published
- User will setup resource principal policy such that
    - ML Jobs can pull the container
    - ML Jobs can create new Jobs on the same subnet and compartment
    - ML Jobs can access/use user provided subnet
    - ML Jobs can access object storage (optional, this is required if user has to download/access data from object storage during training time. This might also be useful if user wants to save the model in the object storage)

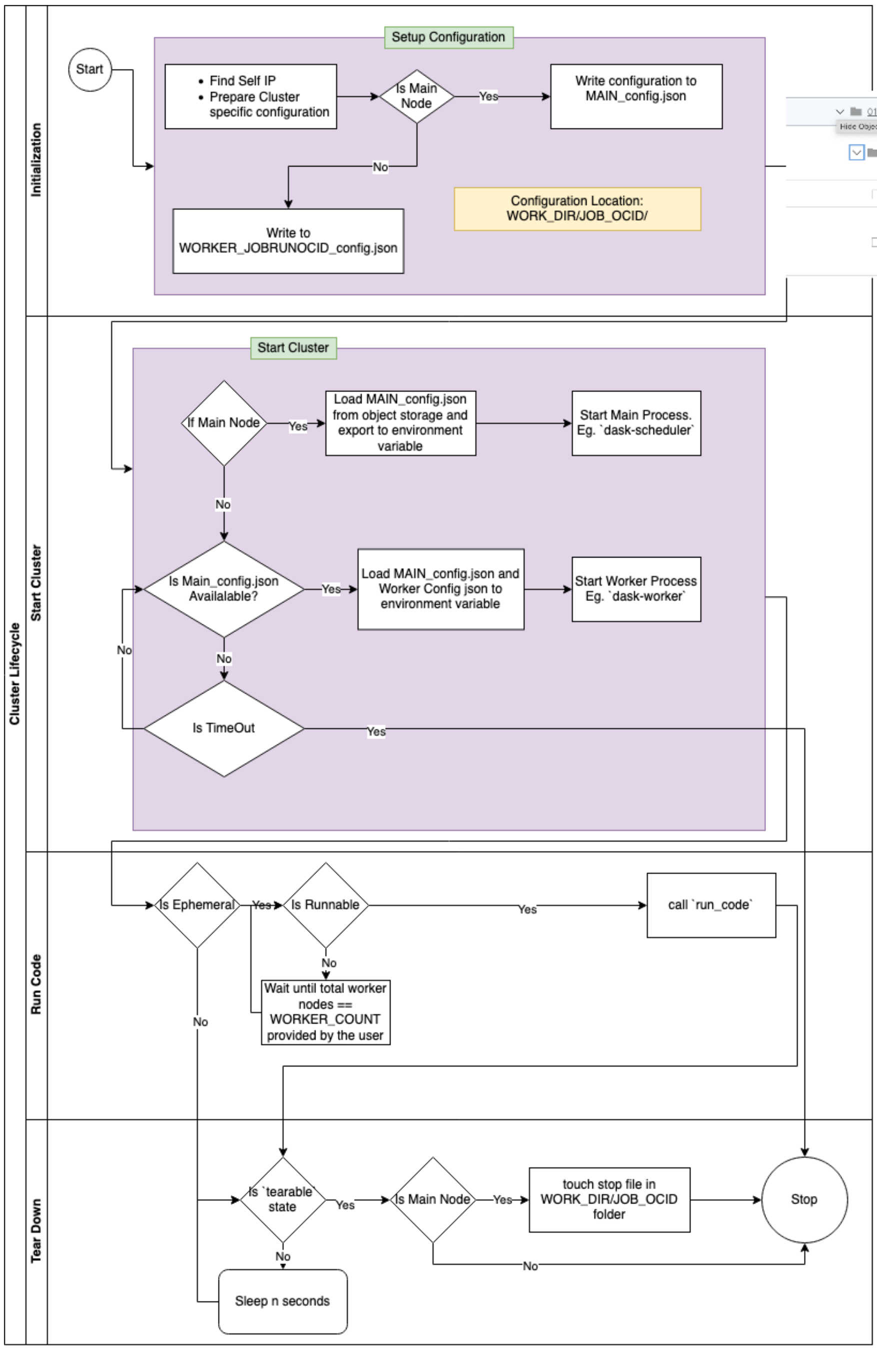
4. Caveat

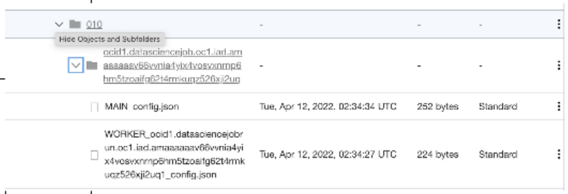
The main goal of this exercise is to allow users to run distributed training using Jobs. The exact nature of that interaction is less important that the ability of the user to be able to run distributed training jobs. While this design document aims to capture possible UX, we should **not** consider to stick to the design, but allow continuous evolution as we research, experiment and gather feedback. From that sense this document will be continuously updated as we change our approach based on our research.

5. High Level Design

5.1. Workflow

https://bitbucket.oci.oraclecorp.com/projects/ODSC/repos/ads-misc/browse/dsc_distributed?at=distributed


Cluster Lifecycle
Initialization
Start Cluster
Run Code
Tear Down
Start
Setup Configuration
• Find Self IP
• Prepare Cluster specific configuration
Is Main Node
Yes
No
Write configuration to MAIN_config.json
Write to WORKER_JOBRUNOCID_config.json
Configuration Location: WORK_DIR/JOB_OCID/
If Main Node
Load MAIN_config.json from object storage and export to environment variable
Start Main Process. Eg. `dask-scheduler`
Is Main_config.json Availalable?
Load MAIN_config.json and Worker Config json to environment variable
Start Worker Process Eg. `dask-worker`
Is TimeOut
Is Ephemeral
Is Runnable
call `run_code`
Wait until total worker nodes == WORKER_COUNT provided by the user
Is `tearable` state
Is Main Node
touch stop file in WORK_DIR/JOB_OCID folder
Stop
Sleep n seconds



010	-	-	-
ocid1.datasciencejob.oc1.iad.amaaaaaav66vvnia4yix4vosvxnmp6hm5tzoaifg62t4mkuqz526xji2uq	-	-	-
MAIN_config.json	Tue, Apr 12, 2022, 02:34:34 UTC	252 bytes	Standard
WORKER_ocid1.datasciencejobrun.oc1.iad.amaaaaaav66vvnia4yix4vosvxnmp6hm5tzoaifg62t4mkuqz526xji2uq1_config.json	Tue, Apr 12, 2022, 02:34:27 UTC	224 bytes	Standard

A cluster is read for tear down when any of the following condition is satisfied -

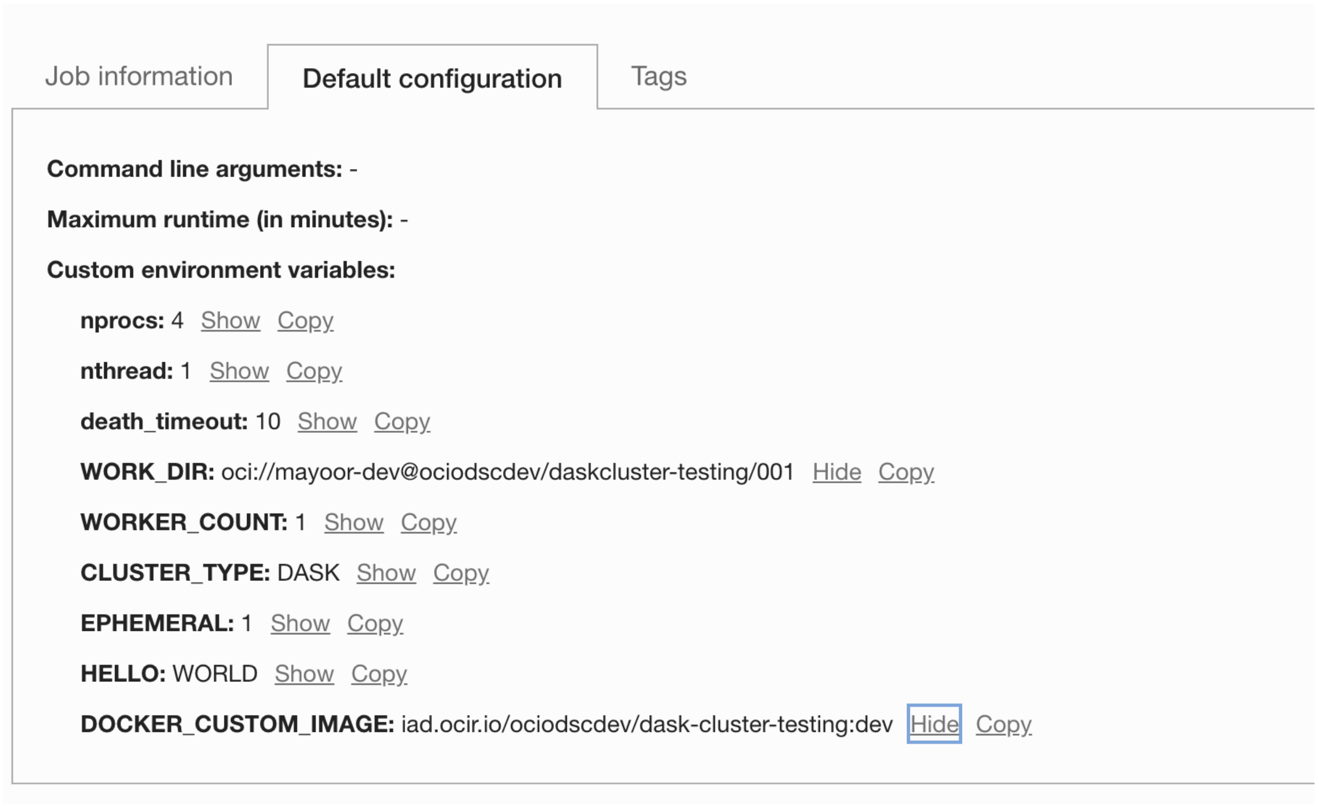
- A cluster is type Ephemeral and execution of ENTRY_POINT is complete
- A stop file is found in WORK_DIR/JOB_OCID directory. stop file acts as a control file to stop cluster immediately.
- A cluster is not Ephemeral but TTL is reached
- Worker times out waiting for MAIN_config.json. The default timeout is 600 seconds. Can be configured by the user using environment variable TIME_OUT
- Ready State is not attained within TIME_OUT. This could be because one or more of workers could not join master within configured time.

5.2. Key Configuration Parameters

Parameter	DEFAULT	REQUIRED	Description

WORK_DIR	NONE	Y	fsspec supported filesystem - oci objectstorage, s3, gitfs, etc
CLUSTER_TYPE	NONE	Y	The bootstrap process uses this information to delegate the cluster initialization process to the ClusterProvider implementation corresponding to CLUSTER_TYPE
WORKER_COUNT	1	N	Number of expected worker nodes. Can be used to check if the cluster is ready for execution
EPHEMERAL	1	N	If 1, then cluster will run the code provided in `ENTRY_SCRIPT` and stops cluster. If 0, the cluster is expected to be long lived.
ENTRY_POINT	NONE	N	The code to run when launched as Ephemeral Cluster
MODE	NONE	Y	The value here is either MAIN or WORKER. Used to identify the behavior of the Node. If set as MAIN, this node becomes the scheduler. If set as WORKER, behaves as worker.
TIMEOUT	600 Seconds	N	TIME_OUT for the cluster to get into ready state or Wait time for Main_config.json

5.2.1. Configuration Example In Job



Command line arguments: -

Maximum runtime (in minutes): -

Custom environment variables:

nprocs: 4 Show Copy

nthread: 1 Show Copy

death_timeout: 10 Show Copy

WORK_DIR: oci://mayoor-dev@ociodscdev/daskcluster-testing/001 Hide Copy

WORKER_COUNT: 1 Show Copy

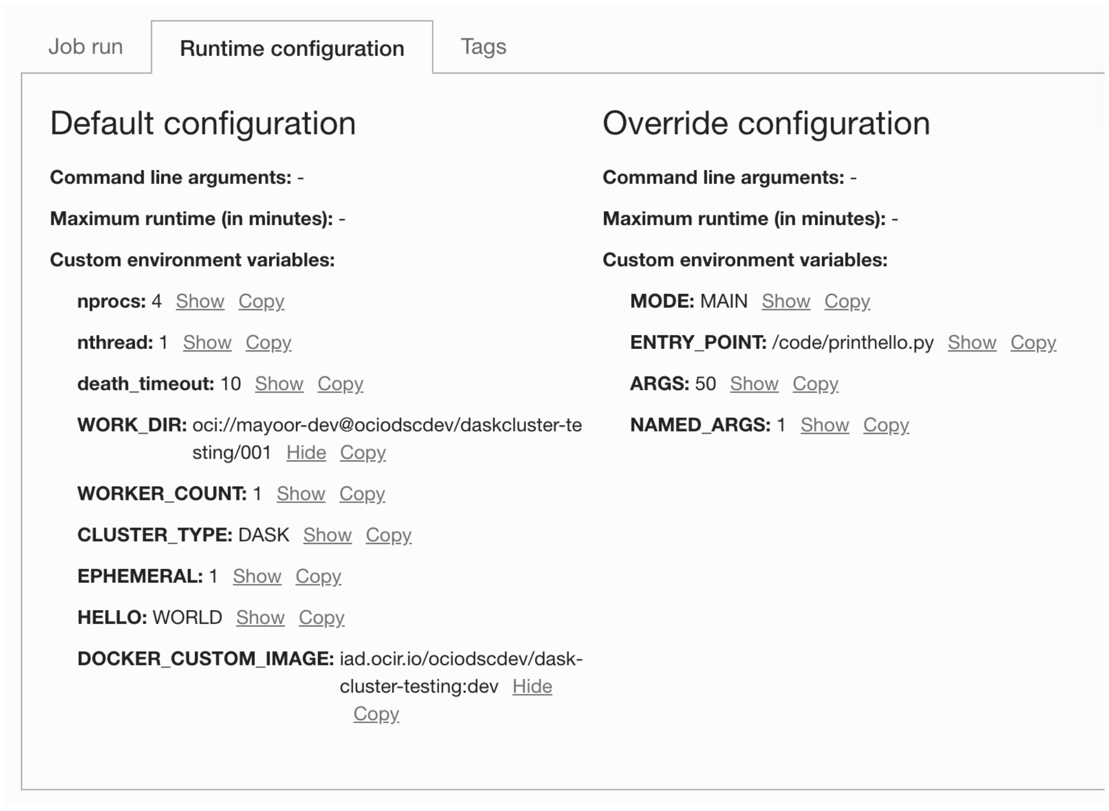
CLUSTER_TYPE: DASK Show Copy

EPHEMERAL: 1 Show Copy

HELLO: WORLD Show Copy

DOCKER_CUSTOM_IMAGE: iad.ocir.io/ociodscdev/dask-cluster-testing:dev Hide Copy

5.2.2. Configuration Example in Job Run



Default configuration

Command line arguments: -

Maximum runtime (in minutes): -

Custom environment variables:

- **nprocs:** 4
- **nthread:** 1
- **death_timeout:** 10
- **WORK_DIR:** oci://mayoor-dev@ociodscdev/daskcluster-testing/001
- **WORKER_COUNT:** 1
- **CLUSTER_TYPE:** DASK
- **EPHEMERAL:** 1
- **HELLO:** WORLD
- **DOCKER_CUSTOM_IMAGE:** iad.ocir.io/ociodscdev/dask-cluster-testing:dev

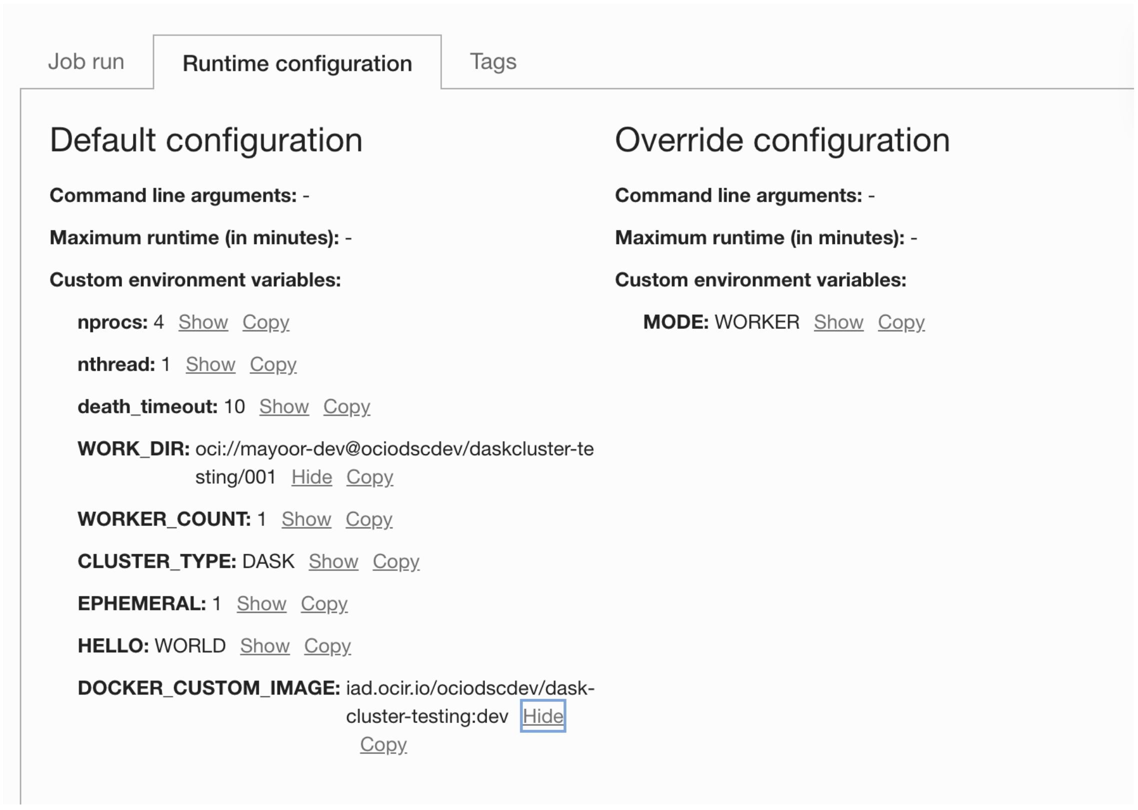
Override configuration

Command line arguments: -

Maximum runtime (in minutes): -

Custom environment variables:

- **MODE:** MAIN
- **ENTRY_POINT:** /code/printhello.py
- **ARGS:** 50
- **NAMED_ARGS:** 1



Default configuration

Command line arguments: -

Maximum runtime (in minutes): -

Custom environment variables:

- **nprocs:** 4
- **nthread:** 1
- **death_timeout:** 10
- **WORK_DIR:** oci://mayoor-dev@ociodscdev/daskcluster-testing/001
- **WORKER_COUNT:** 1
- **CLUSTER_TYPE:** DASK
- **EPHEMERAL:** 1
- **HELLO:** WORLD
- **DOCKER_CUSTOM_IMAGE:** iad.ocir.io/ociodscdev/dask-cluster-testing:dev

Override configuration

Command line arguments: -

Maximum runtime (in minutes): -

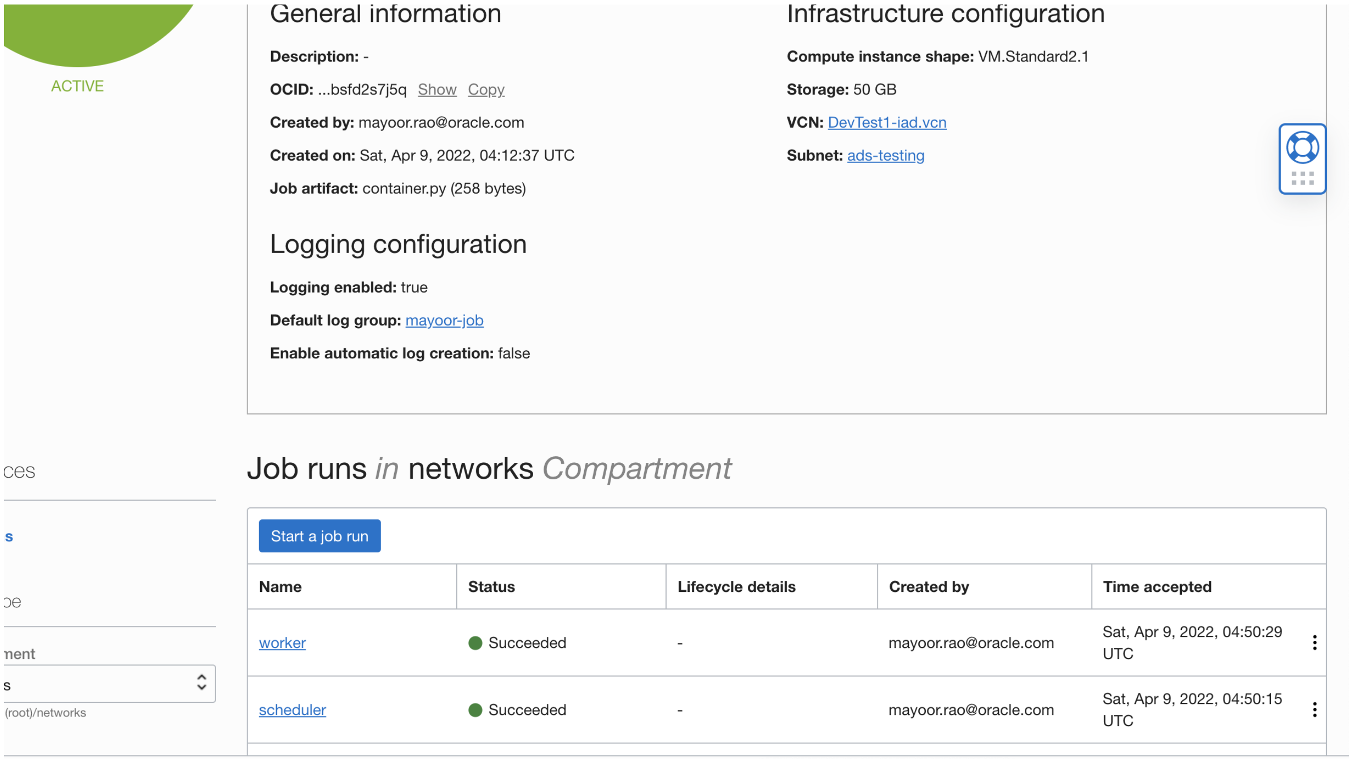
Custom environment variables:

- **MODE:** WORKER

5.3. Cluster to Job Mapping

A cluster is defined by Job and the running instances are Job Runs for that Job Definition. A Job is defined everytime a new cluster is created. Though technically, the job definition can be used to lauch new clusters, the management can get complex as there could be old configurations that are there which can interfere with the new cluster. This design proposes that we have a new Job definition for every new cluster launch.

Here is an example of Dask Cluster. Here we have mulitple job runs under the same job definition. The worker/scheduler is decided by the MODE = MAIN or WORKER provided in the environment variable.



ACTIVE

ces

s

ɔe

nent

s

(root)/networks

General information

Description: -

OCID: ...bsfd2s7j5q Show Copy

Created by: mayoor.rao@oracle.com

Created on: Sat, Apr 9, 2022, 04:12:37 UTC

Job artifact: container.py (258 bytes)

Infrastructure configuration

Compute instance shape: VM.Standard2.1

Storage: 50 GB

VCN: DevTest1-iad.vcn

Subnet: ads-testing

Logging configuration

Logging enabled: true

Default log group: mayoor-job

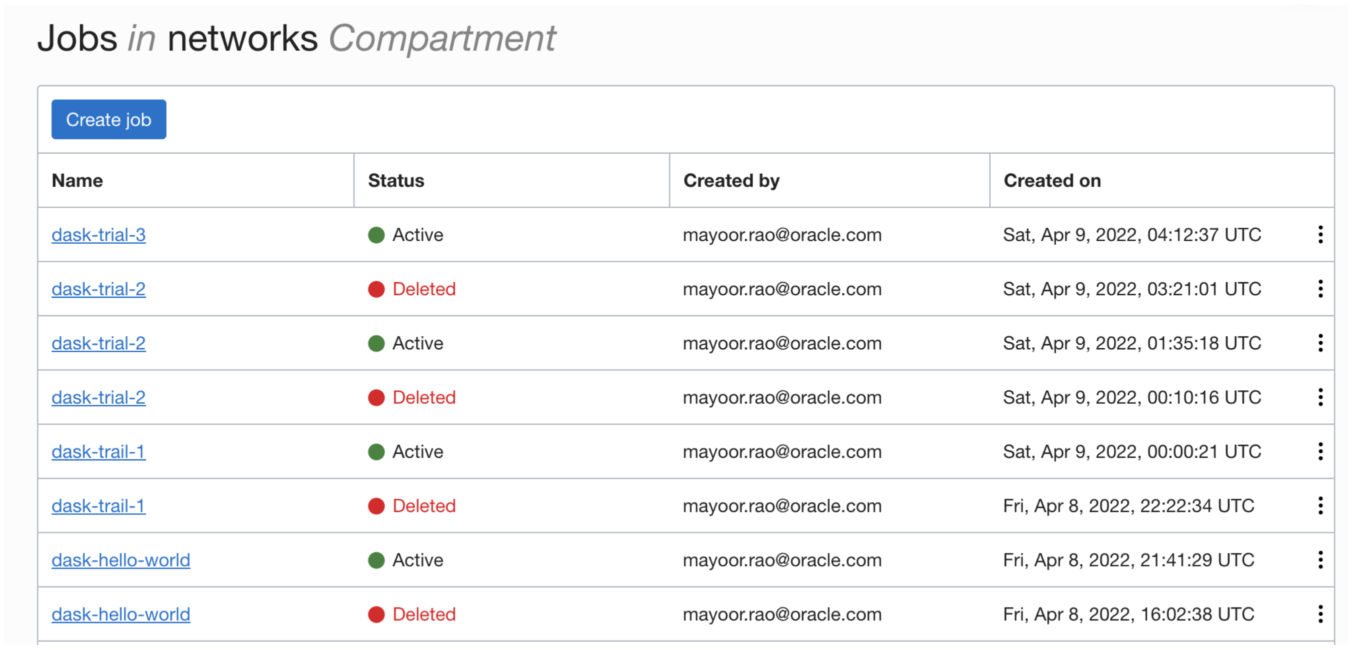
Enable automatic log creation: false

Job runs *in* networks *Compartment*

Start a job run

Name	Status	Lifecycle details	Created by	Time accepted
worker	● Succeeded	-	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 04:50:29 UTC
scheduler	● Succeeded	-	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 04:50:15 UTC

Here is view of multiple cluster launches -



Jobs *in* networks *Compartment*

Create job

Name	Status	Created by	Created on
dask-trial-3	● Active	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 04:12:37 UTC
dask-trial-2	● Deleted	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 03:21:01 UTC
dask-trial-2	● Active	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 01:35:18 UTC
dask-trial-2	● Deleted	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 00:10:16 UTC
dask-trail-1	● Active	mayoor.rao@oracle.com	Sat, Apr 9, 2022, 00:00:21 UTC
dask-trail-1	● Deleted	mayoor.rao@oracle.com	Fri, Apr 8, 2022, 22:22:34 UTC
dask-hello-world	● Active	mayoor.rao@oracle.com	Fri, Apr 8, 2022, 21:41:29 UTC
dask-hello-world	● Deleted	mayoor.rao@oracle.com	Fri, Apr 8, 2022, 16:02:38 UTC

WORK_DIR can be reused by the user across the Cluster creation. The runtime configurations will be isolated using the JOB_OCID folder within WORK_DIR provided by the user.

5.4. Dask

5.4.1.1. Building Images

ADS will contain the Docker recipe to help build the docker image required for Starting a dask cluster. These recipes could be either bundled inside ads wheel or it could be published inside ADS Github. The Docker recipe will include -

The Source below is only for illustration purpose so that we get a sense of what pieces will go into the deliverable. The actual content and exact position of the orchestration will evolve as we develop and experiement

- **Dockerfile Source**

Dockerfile

```
FROM ghcr.io/oracle/oraclelinux7-instantclient:19 AS base

RUN rm -rf /var/cache/yum/* && yum clean all && yum install -y gcc mesa-libGL vim iproute net-tools && rm -rf
/var/cache/yum/*
RUN curl -L https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh >> miniconda.sh
RUN bash ./miniconda.sh -b -p /miniconda; rm ./miniconda.sh;
ENV PATH="/miniconda/bin:$PATH"

ENV HOME /home/datascience
RUN mkdir -p /etc/datascience
WORKDIR /etc/datascience

COPY env.yaml /opt/env.yaml
RUN conda env create -f /opt/env.yaml --name daskenv && conda clean -afy
ENV PATH="/miniconda/envs/daskenv/bin:$PATH"

RUN /bin/bash -c "source activate daskenv"
RUN conda init bash && source ~/.bashrc && conda activate daskenv
COPY run.py /etc/datascience/run.py
COPY cluster_helper.py /etc/datascience/cluster_helper.py

RUN mkdir -p /etc/datascience/dask
COPY start-master.sh /etc/datascience/dask/start-master.sh
COPY start-worker.sh /etc/datascience/dask/start-worker.sh
COPY run.sh /etc/datascience/run.sh

RUN chmod u+x /etc/datascience/run.sh
RUN chmod u+x /etc/datascience/dask/start-master.sh
RUN chmod u+x /etc/datascience/dask/start-worker.sh

ENTRYPOINT ["/etc/datascience/run.sh"]

ARG CODE_DIR
COPY ${CODE_DIR} /code

EXPOSE 3000-3100
EXPOSE 8700-8800
```

- **Scripts to start the cluster**

start-master.sh

```
#!/bin/bash
set -m -e -o pipefail
ifconfig
ip addr
#export MASTER_IP=$(ifconfig ens5 |grep -Po 't \K[\d.]+' | head -1)
#export MASTER_IP=$(ip addr show ens5 | grep -Po 'inet \K[\d.]+')
#export MASTER_IP=$(hostname -i)

#echo $MASTER_IP
#echo $CONFIG_FILE_PATH
export PYTHONPATH=/code
dask-scheduler --host 0.0.0.0 --port 8786 --pid-file main_pid &
export MAIN_PID=$(<main_pid)
echo "main pid $MAIN_PID"
python /code/$ENTRY_SCRIPT
```

start-worker.sh

```
#!/bin/bash
set -m -e -o pipefail

export PYTHONPATH=/code

echo "num procs"
echo $nprocs
echo "death timeout $death_timeout"
dask-worker $SCHEDULER_IP:8786 --worker-port=8700:8800 --nanny-port=3000:3100 --nworkers $nprocs --nthreads 1
--death-timeout $death_timeout &
```

entryscript: run.py

```
https://bitbucket.oci.oraclecorp.com/projects/ODSC/repos/ads-misc/browse/dsc_distributed/run.py?at=distributed
```

- **Dependencies**

Base Dependency

```
channels:
  - conda-forge
  - nodefaults
dependencies:
  - conda-forge::dask
  - conda-forge::python=3.8
  - conda-forge::pip
  - pip:
        - oracle-ads
        - supervisor
```

Finding IP

```
from ads.jobs import Job, DataScienceJob, DataScienceJobRun
from ads.common import auth as authutil
import oci
import ads
import ipaddress
import psutil

def what_is_my_ip(jdef_ocid):


    ads.set_auth("resource_principal")

    jobDef = Job.from_datascience_job(jdef_ocid)

    core_client = oci.core.VirtualNetworkClient(**authutil.resource_principal())

    cidr = core_client.get_subnet(jobDef.infrastructure.subnet_id).data.cidr_block
    ipset = [snics[0].address for interface, snics in psutil.net_if_addrs().items()]
    for ip in ipset:
        if ipaddress.ip_address(ip) in ipaddress.ip_network(cidr):
            return ip
    return None
```

User can build images using -

```
ads opctl build-image --backend distributed-job --type dask --image-name <image name> --source <folder> --
entrypoint <starting script>
```

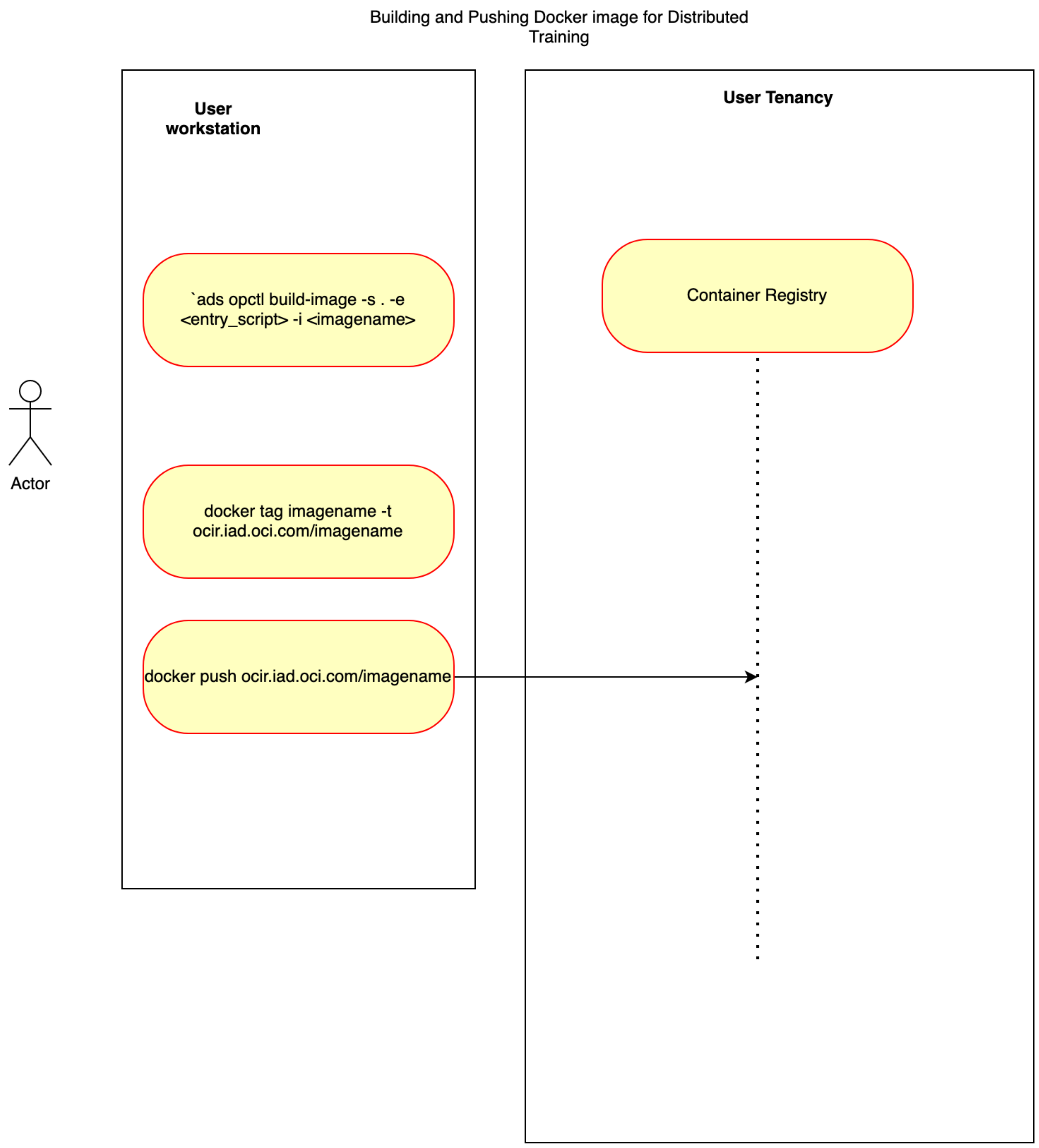
This will build the image locally in the users' workstation. The user can then push the image using `docker push` command to ocir

To add dependencies, user will create conda yaml style dependency file and can provide it as additional input in `ads opctl build-image` command.

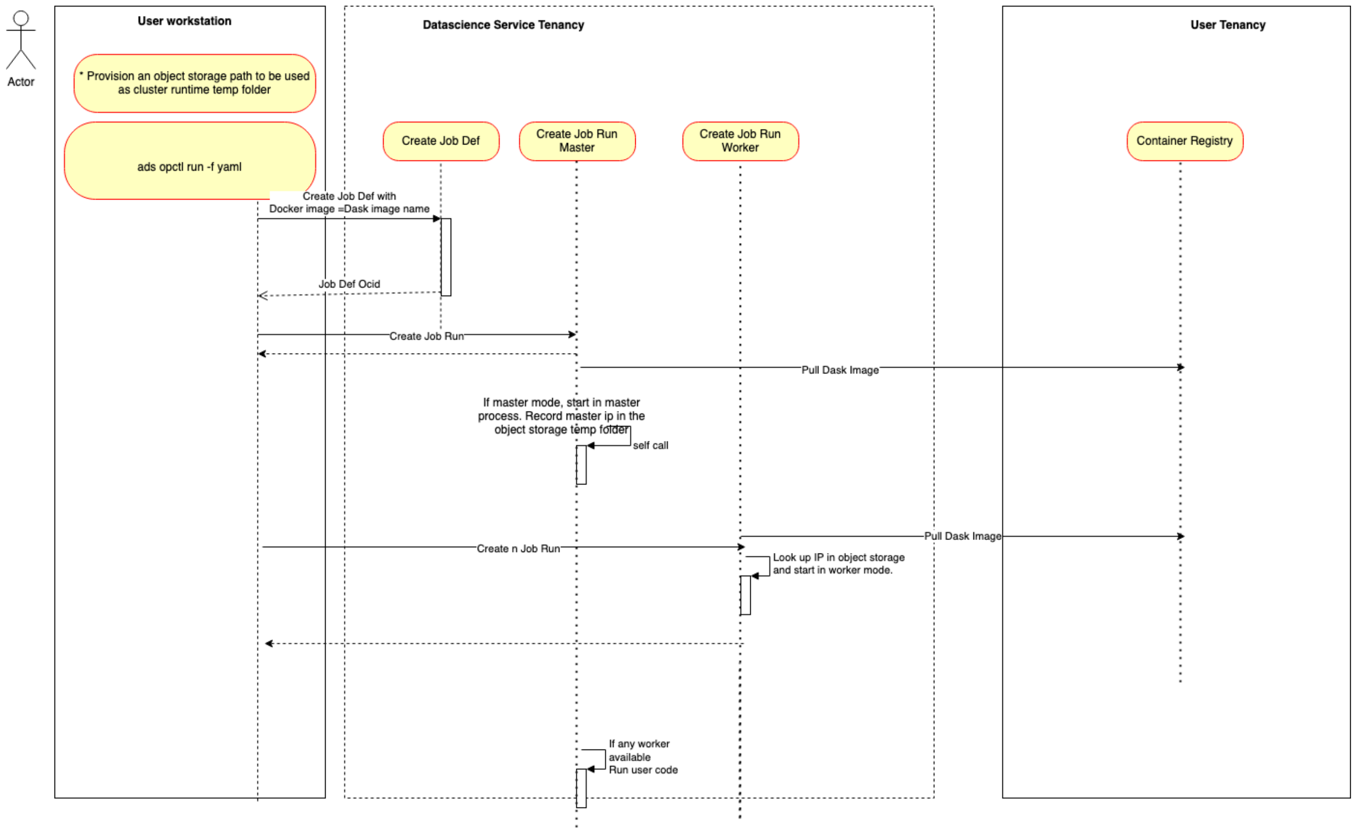
Eg. User who wants to add lightgbm could provide additional dependency by creating a yaml file such as -

```
channels:
  - conda-forge
  - nodefaults
dependencies:
  - conda-forge::lightgbm
```

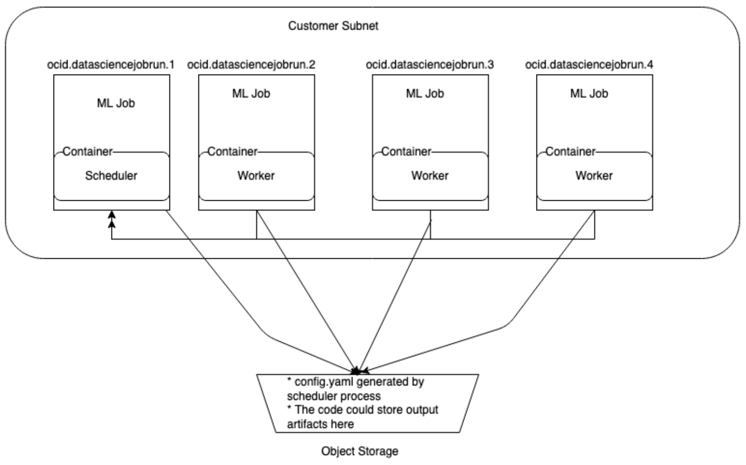
The docker image that is built in local is then pushed to ocir registry of the user tenancy


Building and Pushing Docker image for Distributed Training
User workstation
User Tenancy
`ads opctl build-image -s . -e <entry_script> -i <imagename>
docker tag imagename -t ocir.iad.oci.com/imagename
docker push ocir.iad.oci.com/imagename
Container Registry
Actor

5.4.1.1.1. **Launching Cluster**


Actor
User workstation
* Provision an object storage path to be used as cluster runtime temp folder
ads opctl run -f yaml
Datascience Service Tenancy
Create Job Def
Create Job Run Master
Create Job Run Worker
User Tenancy
Container Registry
Create Job Def with Docker image =Dask image name
Job Def Ocid
Create Job Run
Pull Dask Image
If master mode, start in master process. Record master ip in the object storage temp folder
self call
Create n Job Run
Pull Dask Image
Look up IP in object storage and start in worker mode.
If any worker available Run user code

View of Dask Cluster with 3 workers and 1 scheduler



5.4.1.2. Alternate designs (Designs that were considered but not chosen)

5.4.1.2.1. - With a Driver image

We can build a driver docker image which will launch n jobs with user provided docker command. The driver image will also take ownership of shutting down the cluster based on stopping condition. The ads docker recipe will be restricted to driver code. The user builds and publishes the driver docker image. User will separately build another docker image which contains the code. Such a docker image should reasonably work in the service provided distributed training infrastructure as well. This design is easier to implement in case of horvod than dask. We can probably maintain both approaches. Key advantage of using driver image will be separation of user code and cluster launching logic. When we have service provided distributed training, user can continue to use image corresponding to the code and ignore driver. The driver will not not be directly initiated by the user ever. ADS will lanch the driver and driver will internally lauch the user docker with appropriate configuation.

In case of horvod the worker nodes can automatically discover master node. We could expect user to use horovod public image as their base image.

In case of Dask, the user will have to build the docker image for code from Dockerfile template that we provide. This is because, there is a scheduler node and work node and they are different scripts. So we need some logic to dictate in which mode the image should run.

5.4.1.2.2. Starting workers directly from ADS opctl

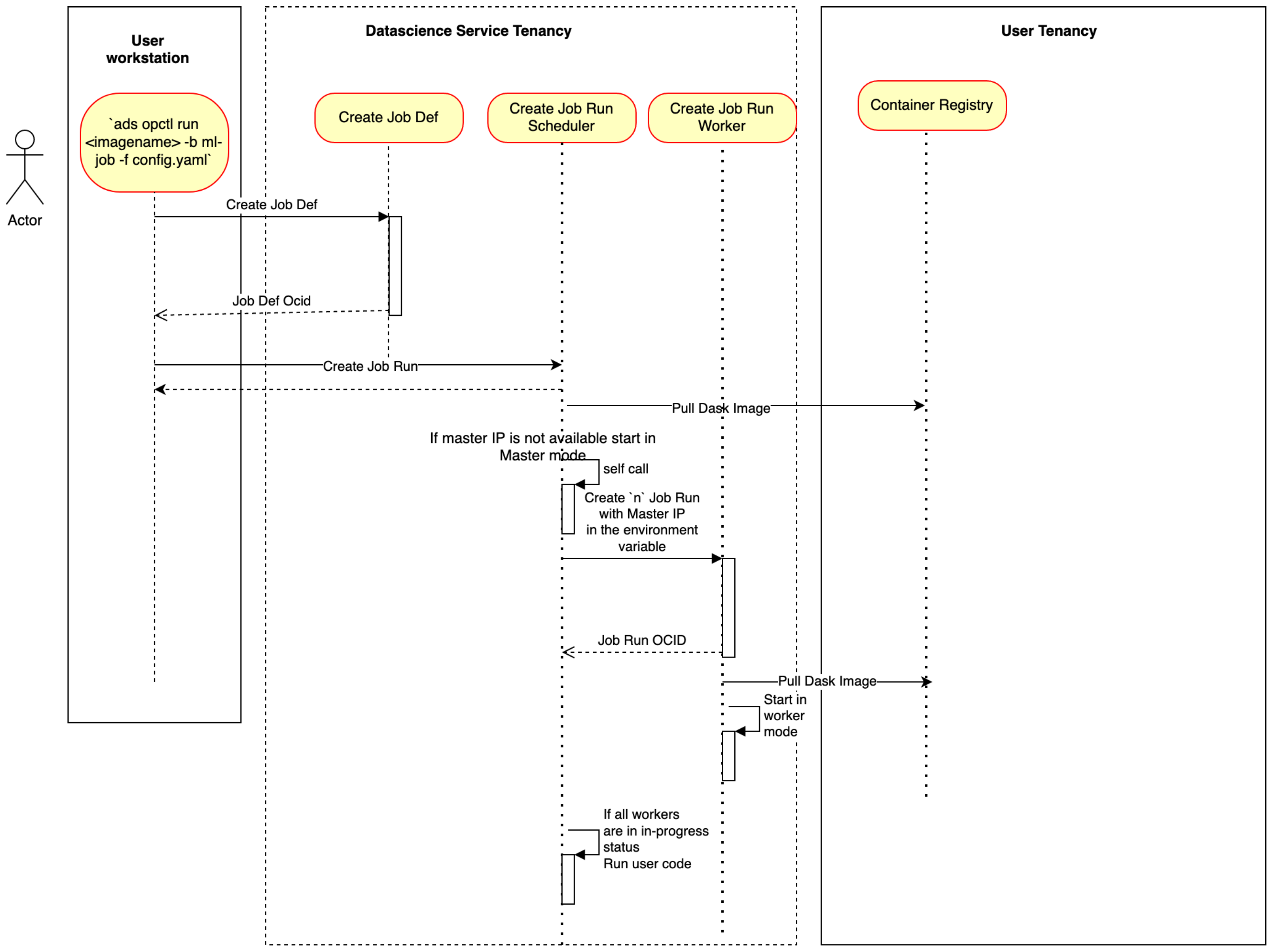
In this approach we start all the workers from the client code (ADS opctl) itself.

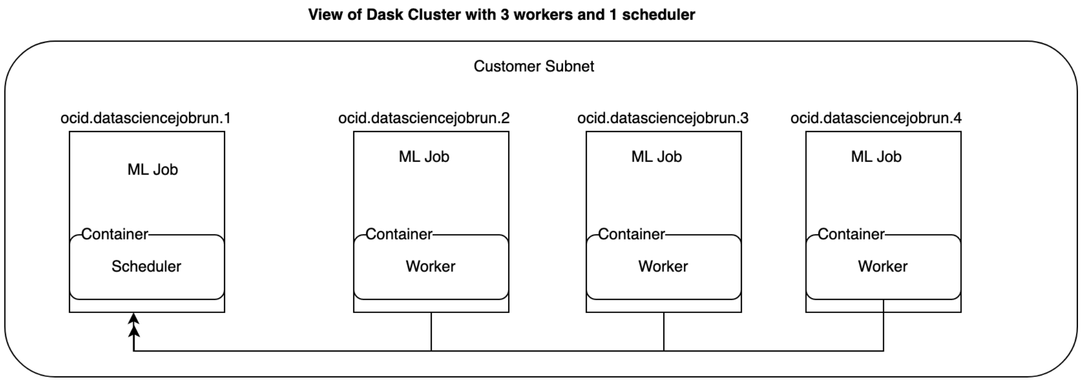
5.4.1.2.3. Provide Launch scripts.

In this approach, we dont tie cluster creation with ADS. We can have launch scripts distributed through our github repo. In this case user will git clone and cd into the distributed-training folder and then follows the Readme.md to build images, push it and then use the launch scripts to create cluster. We can possibly make it easier to dynamically add more workers to the cluster. Updating github is easier than adding features to ADS. Every change to ADS will result in version which may cause confusion to the SDK users who may not be concerned about the `opctl` command lline. Adding hacky code to ADS may lead to cause of concern for enterprise users.

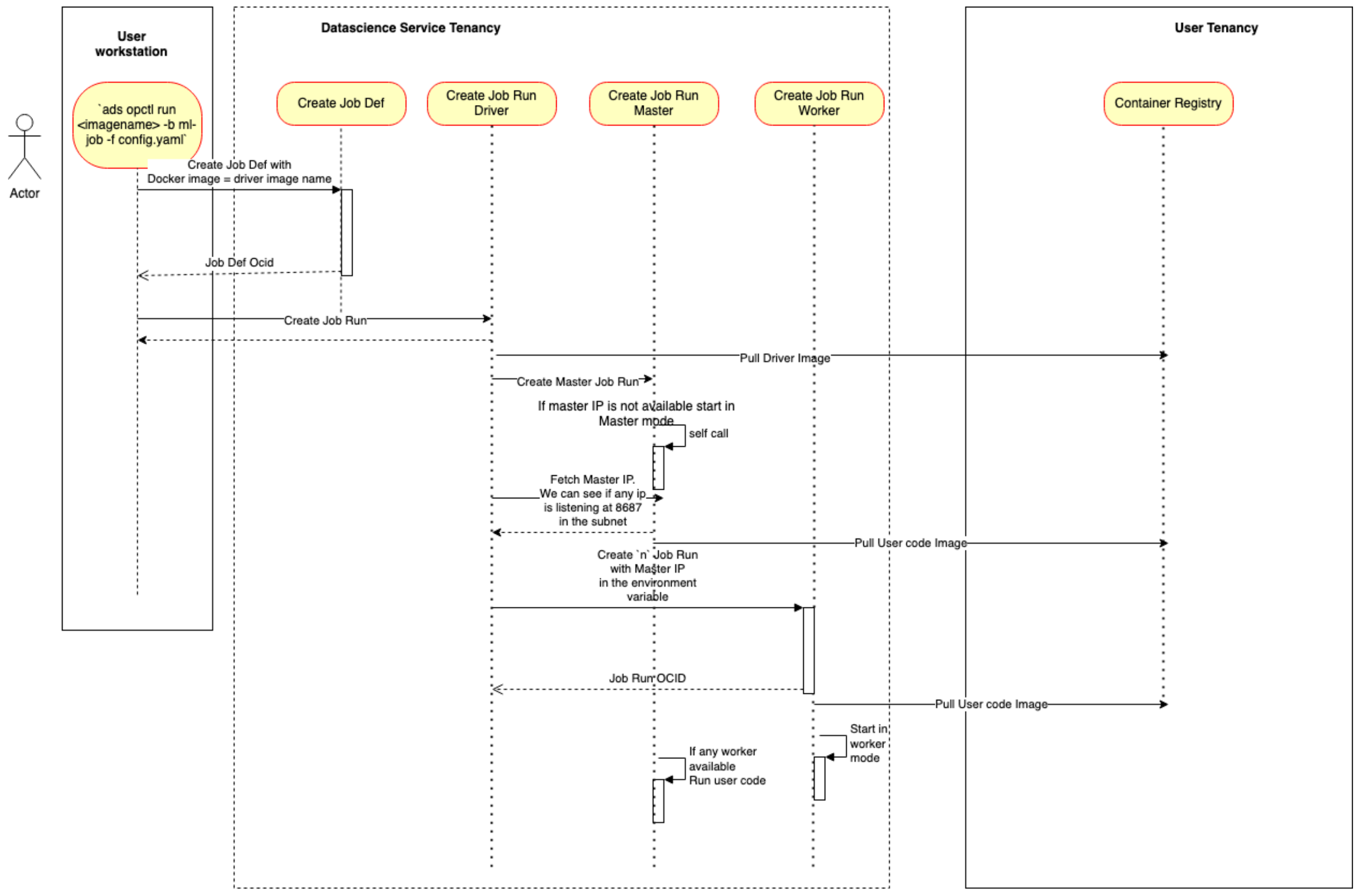
Adding temporary features or capabilities to ADS will give poor UX. Whatever we add to ADS should ideally be supported for long term.

5.4.1.2.4. Scheduler starting n workers.

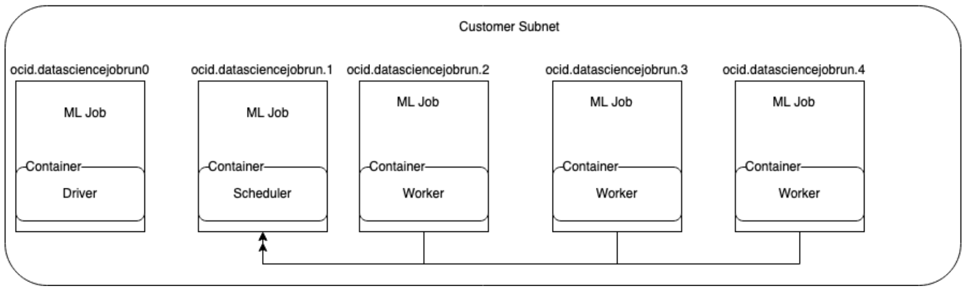

User workstation
Datascience Service Tenancy
User Tenancy
Actor
`ads opctl run <imagename> -b ml-job -f config.yaml`
Create Job Def
Create Job Run Scheduler
Create Job Run Worker
Container Registry
Create Job Def
Job Def Ocid
Create Job Run
Pull Dask Image
If master IP is not available start in Master mode
self call
Create `n` Job Run with Master IP in the environment variable
Job Run OCID
Pull Dask Image
Start in worker mode
If all workers are in in-progress status Run user code



Launching a cluster with Driver Node


User workstation
Datascience Service Tenancy
User Tenancy
Actor
`ads opctl run <imagename> -b ml-job -f config.yaml`
Create Job Def
Create Job Run Driver
Create Job Run Master
Create Job Run Worker
Container Registry
Create Job Def with Docker image = driver image name
Job Def Ocid
Create Job Run
Pull Driver Image
Create Master Job Run
If master IP is not available start in Master mode
self call
Fetch Master IP. We can see if any ip is listening at 8687 in the subnet
Pull User code Image
Create `n` Job Run with Master IP in the environment variable
Job Run OCID
Pull User code Image
Start in worker mode
If any worker available Run user code

View of Dask Cluster with 3 workers and 1 scheduler


Customer Subnet
ocid.datasciencejobrun0
ML Job
Container
Driver
ocid.datasciencejobrun.1
ML Job
Container
Scheduler
ocid.datasciencejobrun.2
ML Job
Container
Worker
ocid.datasciencejobrun.3
ML Job
Container
Worker
ocid.datasciencejobrun.4
ML Job
Container
Worker