Must use a COS-enabled OS (50GB boot disk. 10GB default runs out of space.)

--COS: Container-Optimized **OS**

Ssh is through the GCP console only.

Install GPU

> sudo cos-extensions install gpu

Check that drivers work

sudo mount --bind /var/lib/nvidia /var/lib/nvidia

sudo mount -o remount,exec /var/lib/nvidia

/var/lib/nvidia/bin/nvidia-smi

/var/lib/nvidia/bin/nvidia-smi

# make sure that docker works

docker run

\ --volume

/var/lib/nvidia/lib64:/usr/local/nvidia/lib64

\ --volume

/var/lib/nvidia/bin:/usr/local/nvidia/bin

\ --device

/dev/nvidia0:/dev/nvidia0

\ --device

/dev/nvidia-uvm:/dev/nvidia-uvm

\ --device

/dev/nvidiactl:/dev/nvidiactl

\ gcr.io/google\_containers/cuda-vector-add:v0.1

Cuda image

docker run -i

|  |
| --- |
| us-docker.pkg.dev/deeplearning-platform-release/gcr.io/base-cu122.py310 |
|  |

This container works for CUDA development with nvcc. gcr.io/google\_containers/cuda-vector-add:v0.1

AWS – launch g4dn.xlarge  with

Deep Learning Base OSS Nvidia Driver GPU AMI (Ubuntu 22.04) 20241025

Cuda works out of the box.

Docker run

docker run -it --gpus all nvidia/cuda:12.4.1-cudnn-devel-ubuntu22.04 bash

docker pull nvidia/cuda:12.9.1-cudnn-devel-ubuntu24.04

GKE: Google Kubernetes Engine

-

In GCP website:

Enable Compute Engine API and Kubernetes Engine API

-

Resources request request:

Resources unavailable. Requested to increase GPU quota then worked? Coincidental?

-

Do this each time when VM instance is stopped and started

Install NVIDIA drivers: sudo cos-extensions install gpu to install NVIDIA drivers

Make nvidia files executable: sudo mount --bind /var/lib/nvidia /var/lib/nvidia, sudo mount -o remount,exec /var/lib/nvidia

Test driver: /var/lib/nvidia/bin/nvidia-smi

If nvidia is not setup as runtime: nvidia-ctk –version, sudo nvidia-ctk runtime configure --runtime=docker, sudo systemctl restart docker

Find devices by sending output to yaml file: sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml to find devices

For port forwarding: Create firewall rule in GCP. Firewall access scope issue: Allow full access to all Cloud API in the VM instance configuration.

Run docker with port forwarding: docker run -it --runtime=nvidia -p 2222:22 nvidia/cuda:12.9.1-cudnn-devel-ubuntu24.04 bash

-

In the docker:

apt update

apt install openssh-server

service ssh start

apt install net-tools then check for port 22 SSH

apt install nano for editing

Setup root password: RUN echo "root:password" | chpasswd

RUN echo”PermitRootLogin Yes” >> /etc/ssh/sshd\_config

Mkdir cudadev

COPY authorized\_keys /root/.ssh/

Modifiy /etc/ssh/sshd\_config to have PermitRootLogin Yes

EXPOSE 22

service restart ssh

-

From the connection computer to copy:

ssh-copy-id -i /Users/jungsangyoon/mykeys/remote-ssh-gcp1.pub -p 2222 [root@34.125.138.217](mailto:root@34.125.138.217)

-

Some of the useful gcloud commands:

gcloud auth login to initiate access to gcp

gcloud compute ssh --zone "us-west4-b" "instance-20250715-195630" --project "adept-storm-466013-f6" to connect to the project from terminal.

-

Virtual environment in container for ydf build

apt install pip

apt install python3.12-venv

python3 -m venv . for the current directory

source bin/activate

deactivate

pip install ydf

install python in ssh vs code

select python interpreter for venv

Dockerfile

docker build -t myimage:latest .

docker run -it --runtime=nvidia -p 2222:22 cudadev bash

-

Remove ssh key from known\_hosts

ssh-keygen -R "[34.125.138.217]:2222" -f /Users/jungsangyoon/.ssh

-

In the VM,

bash setup.sh

-

After VM set

1. In the VM,
   1. Run setup.sh in the VM home directory: sh setup.sh
   2. Build docker using Docker file docker build -t cudadev:latest .
   3. Connect to container with VM port 2222 forwarding to container port 22: docker run -it --runtime=nvidia -p 2222:22 cudadev bash
2. In the connecting computer
   1. Remove ssh key from known\_hosts: ssh-keygen -R "[34.125.138.217]:2222" -f /Users/jungsangyoon/.ssh
   2. In VS Code, Remote-SSH: Connection to Host of remote-container1
   3. Open the folder to /root/venv
   4. Check the IP of VM instance, it changes after being stopped
   5. Install Python and C extension pack in the VS Code Remote
   6. Install the Nsight Visual Studio Code Edition extension.
3. In the container
   1. Activate venv: source or . bin/activate /root/venv

sudo docker rmi $(sudo docker images -f "dangling=true" -q) -f

sudo docker exec -it image bash

docker run -it --volume /var/lib/nvidia/lib64:/usr/local/nvidia/lib64 --volume /var/lib/nvidia/bin:/usr/local/nvidia/bin --device /dev/nvidia0:/dev/nvidia0 --device /dev/nvidia-uvm:/dev/nvidia-uvm --device /dev/nvidiactl:/dev/nvidiactl gcr.io/google\_containers/cuda-vector-add:v0.1

export PATH=/usr/local/cuda-12.9/bin${PATH:+:${PATH}}

export LD\_LIBRARY\_PATH=/usr/local/cuda-12.9/lib64{LD\_LIBRARY\_PATH:+:${LD\_LIBRARY\_PATH}}

Dockerfile

FROM nvidia/cuda:12.9.1-devel-ubuntu24.04

ENV CUDA\_PATH=/root/cudadev

RUN apt-get update

RUN apt-get install openssh-server -y

RUN apt-get install net-tools

RUN apt-get install nano

RUN apt-get install git -y

RUN apt-get install cmake -y

RUN apt-get install pip -y

RUN apt-get install python3.12-venv -y

RUN mkdir $CUDA\_PATH

RUN python3 -m venv $CUDA\_PATH

RUN . /root/cudadev/bin/activate && pip install pandas && pip install ydf && deactivate

RUN echo "root:passwd" | chpasswd

RUN echo "PermitRootLogin Yes" >> /etc/ssh/sshd\_config

COPY authorized\_keys /root/.ssh

COPY entrypoint.sh $CUDA\_PATH

RUN chmod +x $CUDA\_PATH/entrypoint.sh

RUN git clone <https://github.com/NVIDIA/cuda-samples> $CUDA\_PATH/cuda-samples

ENTRYPOINT ["/root/cudadev/entrypoint.sh"]

EXPOSE 22

WORKDIR $CUDA\_PATH

CMD ["/bin/bash"]

Setup.sh

#!/bin/sh

echo "Starting setup..."

sudo cos-extensions install gpu

sudo mount --bind /var/lib/nvidia /var/lib/nvidia

sudo mount -o remount,exec /var/lib/nvidia

/var/lib/nvidia/bin/nvidia-smi

sudo nvidia-ctk runtime configure --runtime=docker

sudo systemctl restart docker

sudo nvidia-ctk cdi generate --output=/etc/cdi/nvidia.yaml

echo "Finishing setup..."

entrypoint.sh

#!/bin/sh

echo "Starting container ..."

service ssh restart

exec "$@"

authorized\_keys

ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC6obyOZHE8hl+251chih81Io6Egejx1jRpZZQelKy9leGyc6BiMY1to/3WOdumZCZepGhrS3wr7m/DDrzwfsHoN8oGHizC4ZgttPTLBfMhzFTh7GhfGHe5Ou1W/mWEL8ilozOSLbdjCbKHbKbcTg0NFEkFyHBWXaEK+UEQG0u52u9BkBjsM2qY7i7se1JF5+u3b0OBOGYqYEouhYNynrkxthzz9XXhs8RLLYk4c/a0+FZ7pQHHnrRGJFOWDx8HW3dee6uCl4OVXMGNVcQpzG31ImRk+UcT9TWB883/ODUjCPxuDYaZ4u1/e61nMyYC5mh5zk8QkNsxAvAeSOaqlg+R topyoon2025

mkdir build && cd build

cmake ..

make -j$(nproc)

Test1 cu file

#include <stdio.h>

// Kernel function

\_\_global\_\_ void helloFromGPU() {

printf("Hello from GPU!\n");

}

int main() {

// Host code (runs on CPU)

printf("Hello from CPU!\n");

//Launch the kernel on the GPU

helloFromGPU<<<1, 1>>>(); // Launch one thread block with one thread

// Wait for the kernel to finish

cudaDeviceSynchronize();

printf("Done!\n");

return 0;

}

Or nvcc -o hello hello.cu

./hello

For test files,

Create/copy a folder to work from in cuda-samples then modify CmakeLists.txt for the right cu code and executable names.

In the VS code setting,

Add extension items for Remote-SSH.

import ydf

import pandas as pd

import subprocess

print(ydf.\_\_version\_\_)

ds\_path = "<https://raw.githubusercontent.com/google/yggdrasil-decision-forests/main/yggdrasil_decision_forests/test_data/dataset>"

train\_ds = pd.read\_csv(f"{ds\_path}/iris.csv")

label = "class"

model = ydf.RandomForestLearner(label=label, num\_trees=10).train(train\_ds)

model.describe()

with open("ydf\_codereview\_model.h", "w") as f:

f.write(model.to\_cpp(key="ydf\_codereview"))

subprocess.run(['cat', 'ydf\_codereview.h'])

\_\_globla\_\_: funtion to be called from a host

\_\_global\_\_ void kernel (int \*d\_out, int \*d\_in)

Bazel installation

* curl -L https://github.com/bazelbuild/bazelisk/releases/latest/download/bazelisk-linux-amd64 -o /usr/local/bin/bazel
* chmod +x /usr/local/bin/bazel
* bazel version

Checksum issue: copy this file into

* ~/cudadev/yggdrasil-oblique-forests/third\_party/eigen3/workspace.bzl

"""Eigen project."""

[load("@bazel\_tools//tools/build\_defs/repo:http.bzl](mailto:load(%22@bazel_tools//tools/build_defs/repo:http.bzl)", "http\_archive")

def deps():

http\_archive(

name = "eigen\_archive",

urls = ["<https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.zip>"],

strip\_prefix = "eigen-3.4.0",

sha256 = "eba3f3d414d2f8cba2919c78ec6daab08fc71ba2ba4ae502b7e5d4d99fc02cda",

build\_file\_content =

"""

cc\_library(

name = 'eigen3\_internal',

srcs = [],

includes = ['.'],

hdrs = glob(['Eigen/\*\*']),

visibility = ['//visibility:public'],

)

alias(

name = "eigen3",

actual = "@eigen\_archive//:eigen3\_internal",

visibility = ["//visibility:public"],

)

""",

)

ICX GPG Key / Installation

* wget -qO /etc/apt/trusted.gpg.d/intel-oneapi.asc <https://apt.repos.intel.com/intel-gpg-keys/GPG-PUB-KEY-INTEL-SW-PRODUCTS.PUB>
* echo "deb https://apt.repos.intel.com/oneapi all main" | tee /etc/apt/sources.list.d/intel-oneapi.list
* apt update
* apt install -y intel-oneapi-compiler-dpcpp-cpp
* echo 'source /opt/intel/oneapi/setvars.sh > /dev/null' | tee -a /etc/profile.d/intel-oneapi.sh
* source /etc/profile
* icx –version
* icpx --version

Include path modifications

Optional:”

# build --cxxopt=-isystem

# build --cxxopt=/opt/intel/oneapi/compiler/2025.2/lib/clang/21/include

# build --action\_env=CPATH=/opt/intel/oneapi/compiler/2025.2/lib/clang/21/include:/opt/intel/oneapi/compiler/2025.2/opt/compiler/include

Copy this file into:

[load("@bazel\_tools//tools/cpp:cc\_toolchain\_config\_lib.bzl](mailto:load(%22@bazel_tools//tools/cpp:cc_toolchain_config_lib.bzl)",

"tool\_path", "feature", "flag\_group", "flag\_set")

[load("@bazel\_tools//tools/build\_defs/cc:action\_names.bzl](mailto:load(%22@bazel_tools//tools/build_defs/cc:action_names.bzl)", "ACTION\_NAMES")

all\_link\_actions = [

ACTION\_NAMES.cpp\_link\_executable,

ACTION\_NAMES.cpp\_link\_dynamic\_library,

ACTION\_NAMES.cpp\_link\_nodeps\_dynamic\_library,

]

def \_impl(ctx):

tool\_paths = [

tool\_path(name = "gcc", path = "/opt/intel/oneapi/compiler/2025.2/bin/icx"),

tool\_path(name = "ld", path = "/opt/intel/oneapi/compiler/2025.2/bin/icx"),

tool\_path(name = "ar", path = "/usr/bin/ar"),

tool\_path(name = "cpp", path = "/opt/intel/oneapi/compiler/2025.2/bin/icx"),

tool\_path(name = "gcov", path = "/bin/false"),

tool\_path(name = "nm", path = "/usr/bin/nm"),

tool\_path(name = "objdump", path = "/usr/bin/objdump"),

tool\_path(name = "strip", path = "/usr/bin/strip"),

]

features = [

feature(

name = "default\_linker\_flags",

enabled = True,

flag\_sets = [

flag\_set(

actions = all\_link\_actions,

flag\_groups = [

flag\_group(flags = ["-lstdc++"])

]

),

],

),

]

return cc\_common.create\_cc\_toolchain\_config\_info(

ctx = ctx,

features = features,

cxx\_builtin\_include\_directories = [

"/opt/intel/oneapi/compiler/2025.2/lib/clang/21/include",

"/opt/intel/oneapi/compiler/2025.2/opt/compiler/include",

"/opt/intel/oneapi/compiler/2025.2/linux/include",

"/opt/intel/oneapi/compiler/2025.2/linux/include/sycl",

"/usr/include",

],

toolchain\_identifier = "intel-linux-toolchain",

host\_system\_name = "local",

target\_system\_name = "local",

target\_cpu = "x86\_64",

target\_libc = "glibc",

compiler = "icx",

abi\_version = "unknown",

abi\_libc\_version = "unknown",

tool\_paths = tool\_paths,

)

cc\_toolchain\_config = rule(

implementation = \_impl,

attrs = {},

provides = [CcToolchainConfigInfo],

)

/root/cudadev/yggdrasil-oblique-forests/toolchains/cc\_toolchain\_config.bzl

grep -rnw ~/cudadev -e '2025.1'

Source /etc/profile

bazel build //examples:train\_oblique\_forest