



**High-Performance Data-Intensive
Computing Systems Laboratory**

With Big Data Comes Big Compute: Scaling Machine Learning onto Public and Commercial Clouds with Kubernetes - Part 2

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data



University of Missouri

December 2023

Part 2 - Learning Objectives

- ▶ Deploying Scikit Learn ML Jobs to Kubernetes
- ▶ Deploying GPU Jobs to Kubernetes for Training Computer Vision Models
- ▶ Scaling Computer Vision Models on Kubernetes with Job Automation



MU-HPDI/Nautilus - Reminder from Part 1

- ▶ Sample Dockerfiles
- ▶ Sample Kubernetes YAML File
- ▶ Wiki with detailed walkthroughs for:
 - ▶ Getting Started
 - ▶ Creating PVC
 - ▶ Creating Pods
 - ▶ Creating Jobs
- ▶ JupyterHub Instance:
<https://bigdata-2023.nrp-nautilus.io/>
- ▶ Tutorial Repository of Jupyter Project Pages, Code Samples, YAML, etc.
<https://github.com/MU-HPDI/bigdata-2023>
- ▶ Git Clone Command:
git clone <https://github.com/MU-HPDI/bigdata-2023>.git



Over a short break, we will ensure everyone has cloned this Repo into their JupyterLab environment



**High-Performance Data-Intensive
Computing Systems Laboratory**

Part 2-A

National Research Platform Nautilus Research Cluster

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023



University of Missouri

Motivation

- ▶ Shallow ML modules often require extensive hyperparameter optimization to find optimal performance
- ▶ Deep Learning models require incredible amounts of compute to effectively train
- ▶ Using single developer machines or local on-prem resources often fail to scale effectively
- ▶ Kubernetes Clusters provide an efficient and scalable solution to training ML and deep learning models at scale
- ▶ This workshop will address the building and deployment of ML and deep learning containers to Nautilus to train deep networks



Part-2 Tutorial Outline

- ▶ Quick Containerization Review and Advanced:
 - ▶ Review of containerization
 - ▶ Building ML containers for Scikit-Learn
 - ▶ Building Optimized Containers for Deep Learning with PyTorch
 - ▶ Using Common Frameworks: Detectron2, MMDetection, and Ultralytics
- ▶ Kubernetes ML / Deep Learning:
 - ▶ Review of Kubernetes Architecture and Key Concepts
 - ▶ Introduction to NRP Nautilus HyperCluster
 - ▶ Migrating Data to NRP with S3
 - ▶ Deploying Scikit-Learn ML Jobs to Nautilus
 - ▶ Deploying GPU Jobs to Nautilus for Training Computer Vision Models



Containerization

Building Optimized Containers for Deep Learning with PyTorch

Containerization

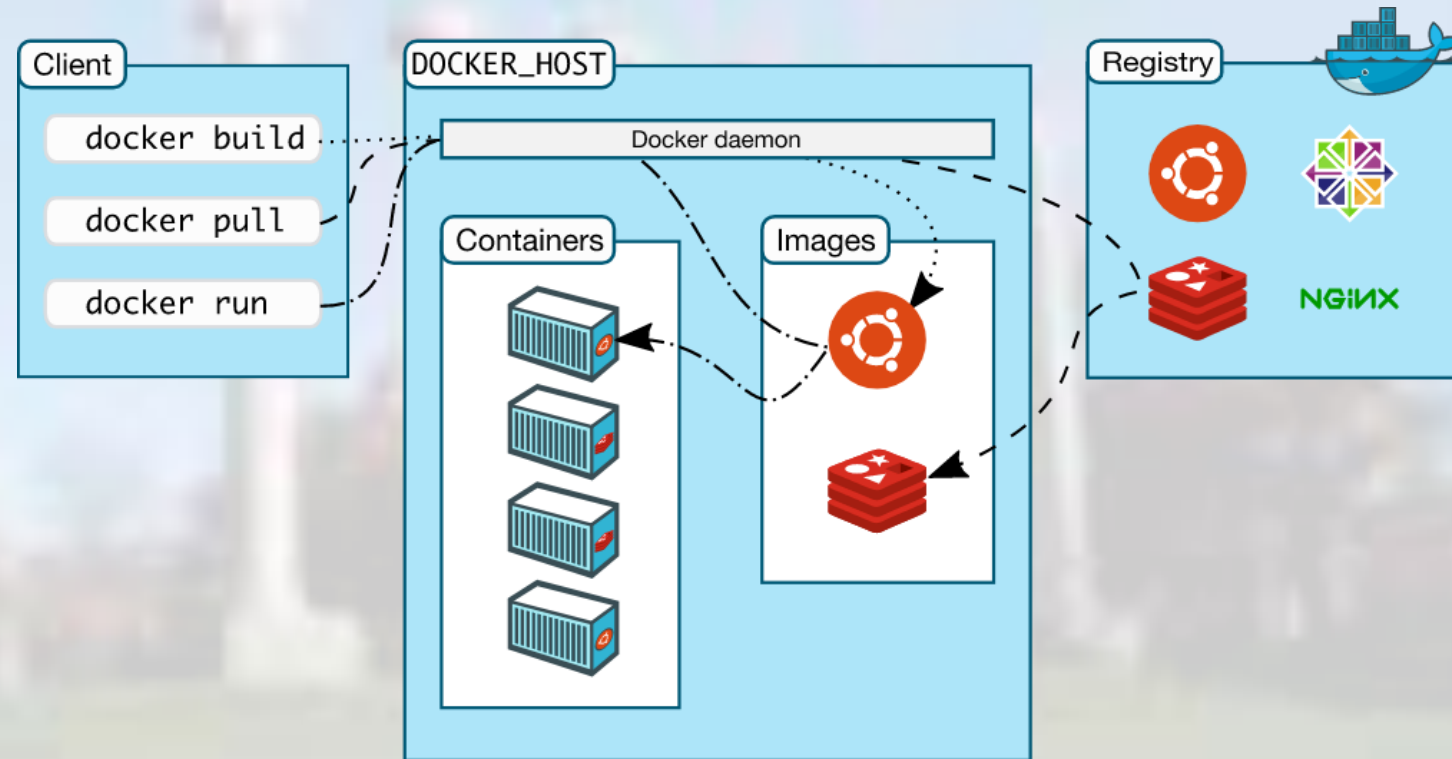
Docker

► Docker

- You can think of Docker containers as mini-VMs that contain all the packages, both at the OS and language-specific level, necessary to run your software.

► Nautilus Gitlab

- Offers the ability to create repo for your code
- Offers the ability to create easily maintained and developed custom images using CI/CD feature



Containerization

Nautilus Gitlab

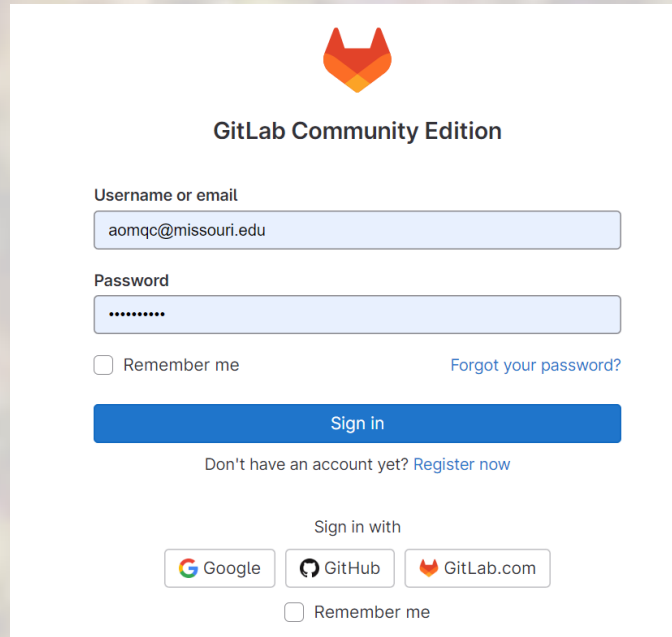
<https://gitlab.nrp-nautilus.io/>

- ▶ Git Version Control System
- ▶ CI/CD services for building and storing container images to use on Nautilus



Containerization

Nautilus Gitlab



GitLab Community Edition

Username or email

Password

☐ Remember me [Forgot your password?](#)

[Sign in](#)

Don't have an account yet? [Register now](#)

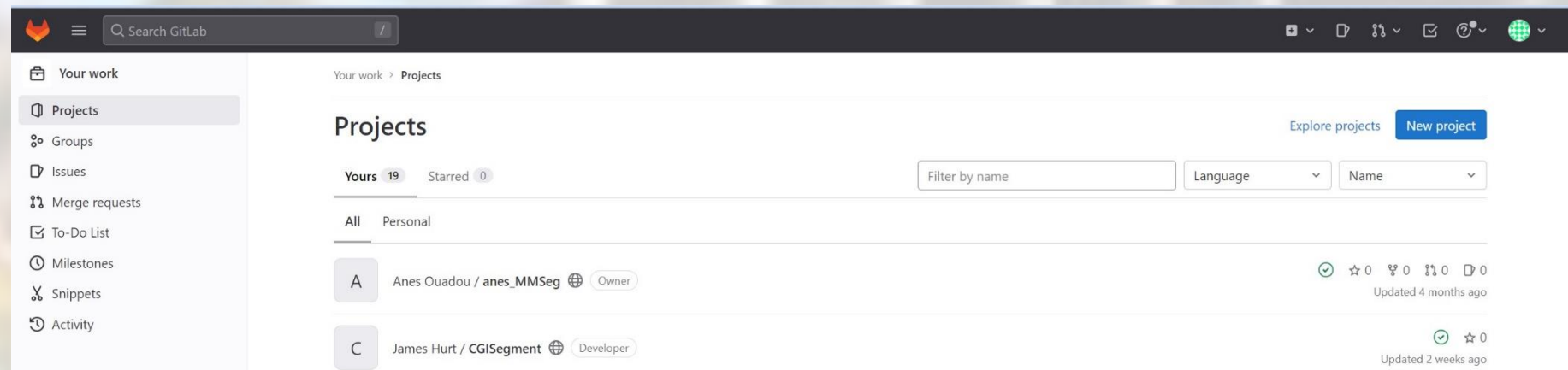
Sign in with

[Google](#) [GitHub](#) [GitLab.com](#)

☐ Remember me

Create an account on Nautilus GitLab

Note: You may need to have your university system added by NRP Admins to use NRP GitLab



GitLab Projects page showing a list of projects. The left sidebar shows navigation options: Your work, Projects, Groups, Issues, Merge requests, To-Do List, Milestones, Snippets, and Activity. The main content area shows a list of projects with columns for name, owner, and update date.

Project	Owner	Updated
A Anes Ouadou / anes_MMSeg	Owner	Updated 4 months ago
C James Hurt / CGISegment	Developer	Updated 2 weeks ago

Nautilus Gitlab

create custom image

Create new repository

- Code & other files can be private, but the repository and the container registry MUST be public
- Open the repository in the Web IDE or clone it locally

Create Gitlab CI YAML File

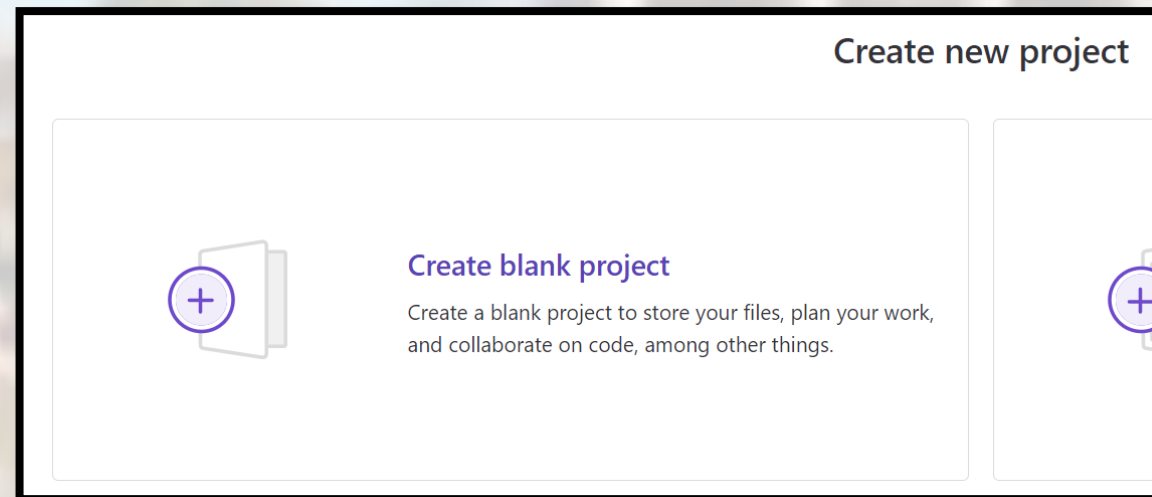
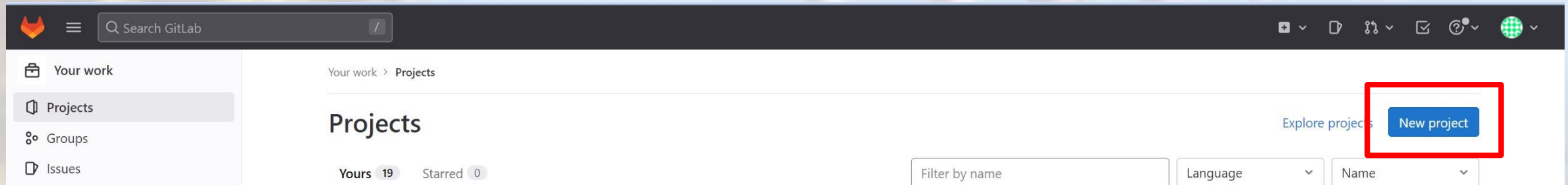
- Create a file called `.gitlab-ci.yml`
- Copy its content from this link: [.gitlab-ci.yml](#)
- Commit and push to your repository

Create Dockerfile

- Write the appropriate commands to build the image you need
- Use a community container as your base
- Commit and push to your repository

Nautilus Gitlab

create custom image



Nautilus Gitlab

create custom image

Projects

Groups

Issues


Merge requests

To-Do List

Milestones

Snippets

Activity



Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Project name

My awesome project

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

Project URL

https://gitlab.nrp-nautilus.io/aomqc/

Project slug

my-awesome-project

Want to organize several dependent projects under the same namespace? [Create a group](#).

Visibility Level ?

☐ Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

☐ Internal

The project can be accessed by any logged in user except external users.

☒ Public

The project can be accessed without any authentication.

Project Configuration

☒ Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

☐ Enable Static Application Security Testing (SAST)

Analyze your source code for known security vulnerabilities. [Learn more](#).

Create project

Cancel

Nautilus Gitlab

create custom image

C CGISegment

Project information

Repository

CI/CD

Deployments

Packages and registries

Wiki

C CGISegment

Project ID: 2316 [Leave project](#)

568 Commits
1 Branch
0 Tags
1.1 MB Project Storage

Fix Docker image
Alex Hurt authored 6 months ago

e98e742e

master
cgisegment /
+

Find file

Web IDE


Clone

README

CI/CD configuration

Wiki

Name	Last commit	Last update
cgisegment	Wrong indentation for final test set aggregation	9 months ago
sample_configs	Updating the sample configs	1 year ago
tests	Getting Tests Up-To-Date	1 year ago
.gitignore	*ipynb	1 year ago
.gitlab-ci.yml	Add gitlab CI YML	7 months ago
Dockerfile	Fix Docker image	6 months ago
README.md	Remove ref	7 months ago
main.py	SegFormer and Lawin	10 months ago



University of Missouri

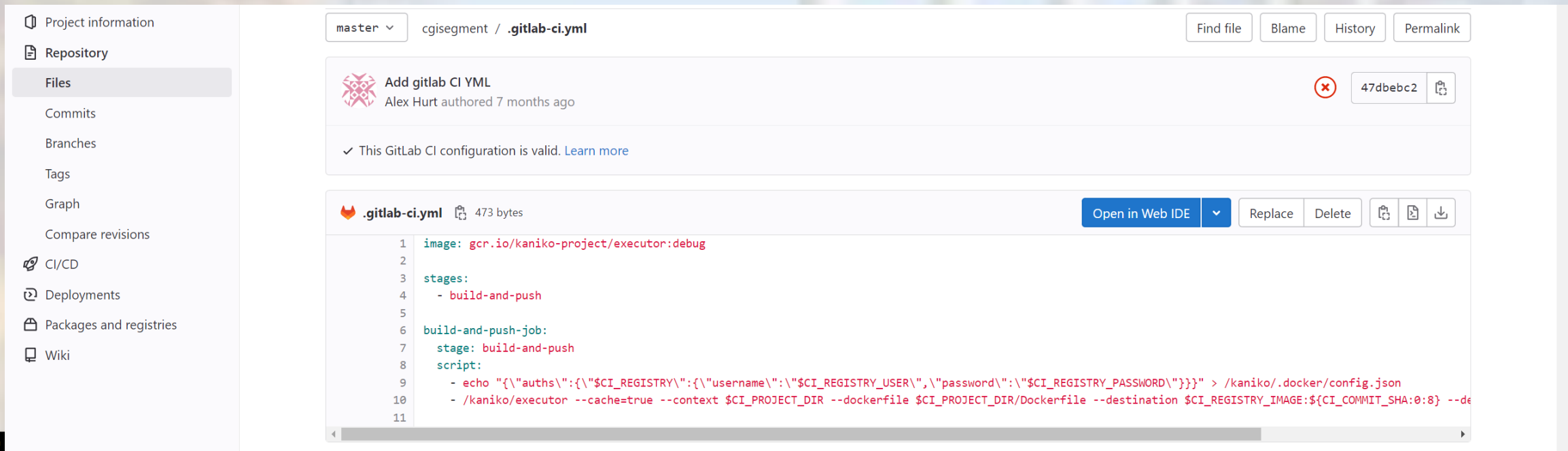
14

Nautilus Gitlab

create custom image

Note: The content of `.gitlab-ci.yml` in the screenshot below is incorrect. The correct contents of this file can be found at this link:

<https://ucsd-prp.gitlab.io/userdocs/development/gitlab/#step-3-continuous-integration-automation>



The screenshot displays the GitLab web interface for a project named 'cgisegment'. The left sidebar shows navigation options: Project information, Repository, Files (selected), Commits, Branches, Tags, Graph, Compare revisions, CI/CD, Deployments, Packages and registries, and Wiki. The main content area shows the 'master' branch with the file '.gitlab-ci.yml' (473 bytes). A message indicates that the configuration is valid. The file content is as follows:

```
1 image: gcr.io/kaniko-project/executor:debug
2
3 stages:
4   - build-and-push
5
6 build-and-push-job:
7   stage: build-and-push
8   script:
9     - echo "{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_USER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
10    - /kaniko/executor --cache=true --context $CI_PROJECT_DIR --dockerfile $CI_PROJECT_DIR/Dockerfile --destination $CI_REGISTRY_IMAGE:${CI_COMMIT_SHA:0:8} --de
11
```

Nautilus Gitlab

create custom image

Two different styles of writing Docker file

```

main v deeplearning_pytorch / Dockerfile
Find file Blame History Permalink

Dockerfile 1.24 KiB
Open in Web IDE Replace Delete

1 FROM nvidia/cuda:11.2.2-cudnn8-runtime-ubuntu20.04
2
3 RUN chmod 1777 /tmp && chmod 1777 /var/tmp
4 RUN apt-get update
5 RUN apt-get upgrade -y
6 RUN apt-get install -y vim curl ca-certificates amqp-tools wget graphviz
7 RUN apt-get -y install git
8
9 RUN apt-get install -y python3
10 RUN apt-get install -y python3-pip
11
12 RUN pip install --upgrade protobuf==3.20.1
13
14 RUN pip install numpy scipy
15 RUN pip install matplotlib
16 RUN pip install pyyaml
17 RUN pip install pytest-shutil
18 RUN pip install rasterio
19 RUN pip install pandas
20 RUN pip install geopandas
21 RUN pip install geojson
22 RUN pip install rtree
23 RUN pip install pillow
24 RUN pip install Jinja2
25 RUN pip install bs4
26 RUN pip install lxml
27 RUN pip install pydot
28 RUN pip install pydotplus
29
30 RUN apt-get install -y binutils libproj-dev gdal-bin
31

```

```

master v cgisegment / Dockerfile
Find file Blame History Permalink

Fix Docker image
Alex Hurt authored 6 months ago
e98e742e

Dockerfile 1.29 KiB
Open in Web IDE Replace Delete

1 FROM pytorch/pytorch:1.12.1-cuda11.3-cudnn8-devel
2
3 #####
4 # env
5 #####
6 RUN mkdir -p /workspace
7 ENV CGISEGMENT /workspace/cgisegment/
8 ENV DEBIAN_FRONTEND noninteractive
9 ENV DISTRO ubuntu2004
10 ENV ARCH x86_64
11 ENV TORCH_HOME /pytorch_data
12
13 #####
14 # Install dependencies
15 #####
16 RUN apt-key adv --fetch-keys https://developer.download.nvidia.com/compute/cuda/repos/$DISTRO/$ARCH/3bf863cc.pub
17 RUN apt update -y && apt install -y \
18     git \
19     build-essential \
20     libsm6 \
21     libxext6 \
22     libxrender-dev \
23     libgl1-mesa-glx \
24     libglu1-mesa-dev \
25     libglu1-mesa-glx \
26
27 #####
28 # Copy files
29 #####
30 RUN mkdir -p ${CGISEGMENT}
31 COPY /main.py ${CGISEGMENT}
32 COPY /cgisegment ${CGISEGMENT}/cgisegment
33 COPY /requirements.txt /tmp/requirements.txt

```

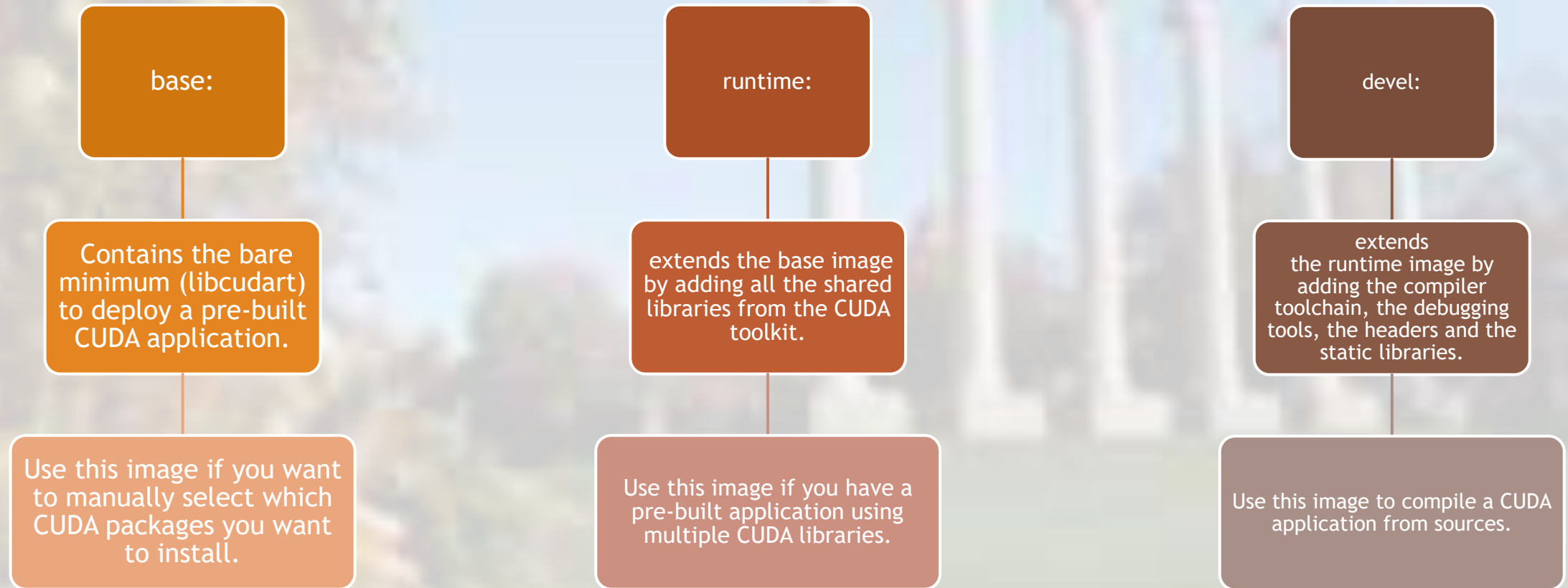
Nautilus Gitlab

create custom image

- ▶ To utilize the GPUs in a cluster, **GPU-enabled image is a MUST**
- ▶ GPU containers can be built from community developed and published base containers, such as nvidia/cuda and the NVIDIA NGC
- ▶ CUDA GPU base images comes in three variations:
 - ▶ base
 - ▶ runtime
 - ▶ devel

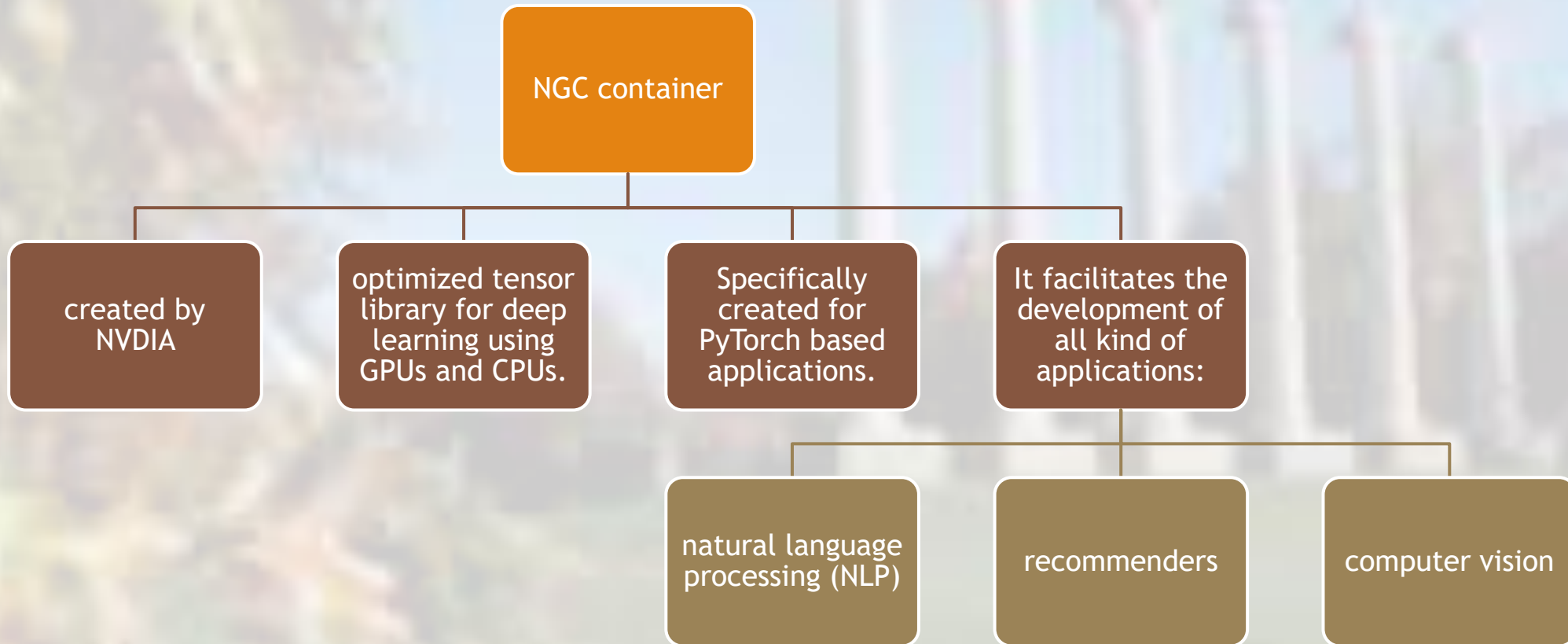
Nautilus Gitlab

create custom image



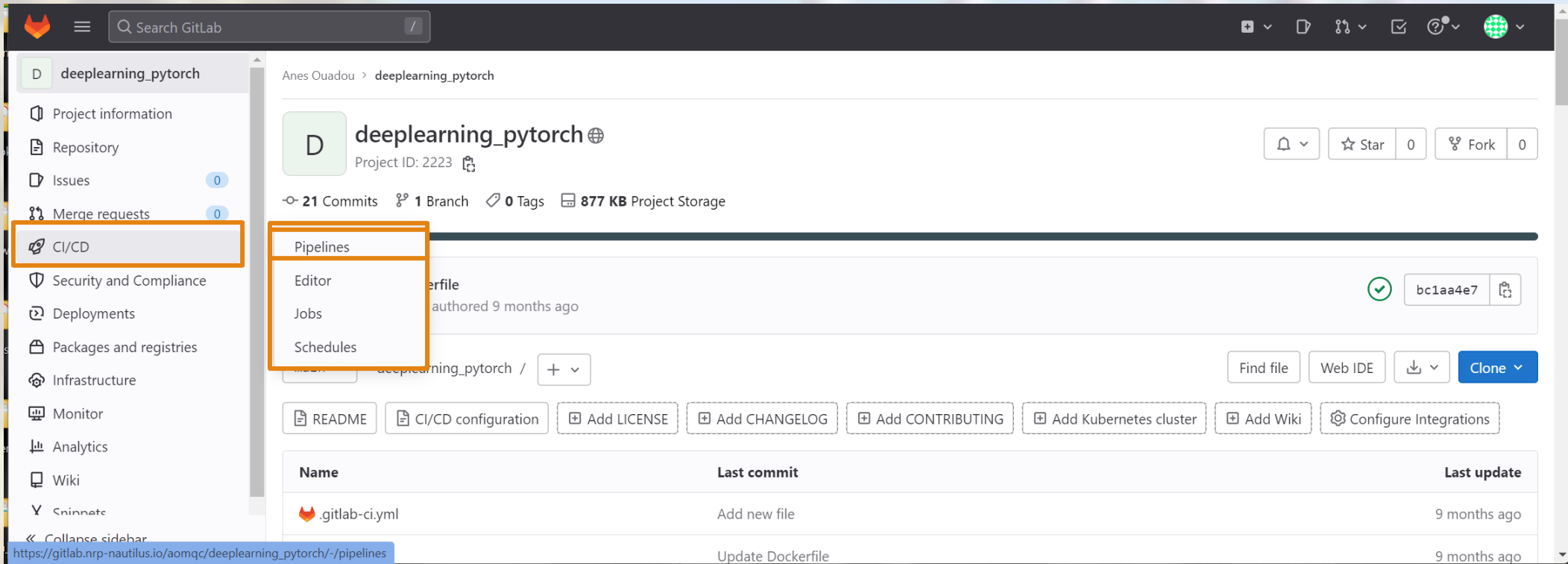
Nautilus Gitlab

create custom image



Nautilus Gitlab

create custom image



Search GitLab

deeplearning_pytorch

Project information
Repository
Issues
Merge requests
CI/CD
Security and Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets

deeplearning_pytorch

Project ID: 2223

21 Commits 1 Branch 0 Tags 877 KB Project Storage

Pipelines
Editor
Jobs
Schedules

bc1aa4e7

Find file Web IDE Clone

README CI/CD configuration Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster Add Wiki Configure Integrations

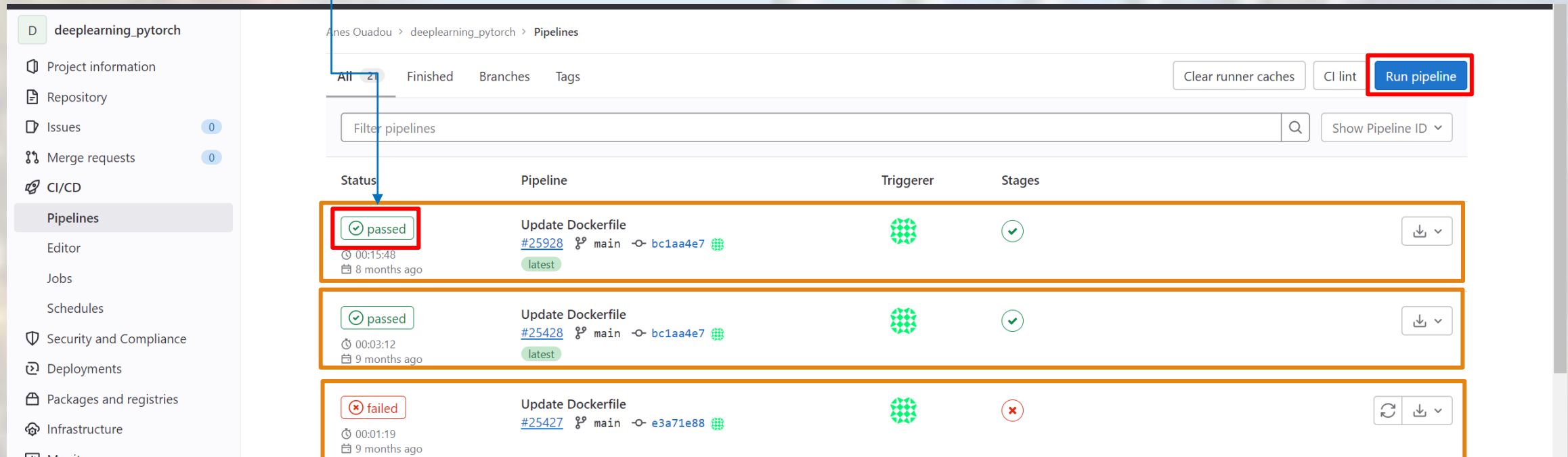
Name	Last commit	Last update
.gitlab-ci.yml	Add new file	9 months ago
	Update Dockerfile	9 months ago

https://gitlab.nrp-nautilus.io/aomqc/deeplearning_pytorch/-/pipelines

Nautilus Gitlab

create custom image

The image is being built the status is:
Running



deeplearning_pytorch

Project information
Repository
Issues
Merge requests
CI/CD
Pipelines
Editor
Jobs
Schedules
Security and Compliance
Deployments
Packages and registries
Infrastructure
Monitor

anes Ouadou > deeplearning_pytorch > Pipelines

All 21 Finished Branches Tags

Clear runner caches CI lint Run pipeline

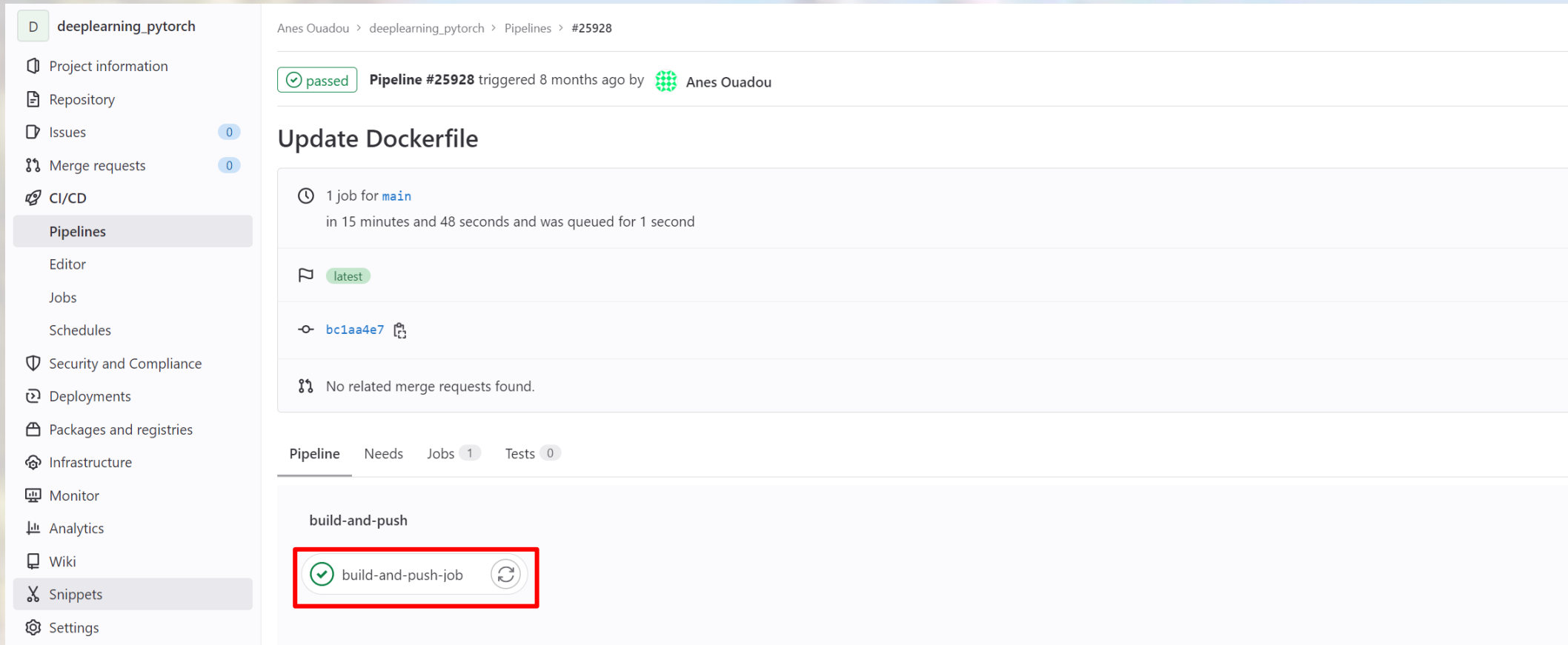
Filter pipelines

Show Pipeline ID

Status	Pipeline	Triggerer	Stages
passed	Update Dockerfile #25928 main bc1aa4e7 latest		
passed	Update Dockerfile #25428 main bc1aa4e7 latest		
failed	Update Dockerfile #25427 main e3a71e88		

Nautilus Gitlab

create custom image



deeplearning_pytorch

Project information

Repository

Issues 0

Merge requests 0

CI/CD

Pipelines

Editor

Jobs

Schedules

Security and Compliance

Deployments

Packages and registries

Infrastructure

Monitor


Analytics

Wiki

Snippets

Settings


Anes Ouadou > deeplearning_pytorch > Pipelines > #25928

passed Pipeline #25928 triggered 8 months ago by  Anes Ouadou

Update Dockerfile

🕒 1 job for `main`
in 15 minutes and 48 seconds and was queued for 1 second


🚩 latest

🔗 `bc1aa4e7` 

🔗 No related merge requests found.

Pipeline Needs Jobs 1 Tests 0

build-and-push

build-and-push-job 

Nautilus Gitlab

create custom image

D
deeplearning_pytorch

Project information
Repository
Issues
Merge requests
CI/CD
Pipelines
Editor
Jobs
Schedules
Security and Compliance
Deployments
Packages and registries
Infrastructure
Monitor
Analytics
Wiki
Snippets
Settings

Search job log

```

2349 Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (1.0.5)
2350 Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (3.0.9)
2351 Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (4.37.3)
2352 Requirement already satisfied: cyclor>=0.10 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (0.11.0)
2353 Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (2.8.2)
2354 Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (9.2.0)
2355 Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from matplotlib->mmsegmentation==0.28.0) (1.4.4)
2356 Collecting wcwidth
2357   Downloading wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
2358 Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7->matplotlib->mmsegmentation==0.28.0) (1.16.0)
2359 Installing collected packages: mmcls, wcwidth, prettytable, mmsegmentation
2360   Running setup.py develop for mmsegmentation
2361 Successfully installed mmcls-0.23.2 mmsegmentation prettytable-3.4.1 wcwidth-0.2.5
2362 INFO[0431] Taking snapshot of full filesystem...
2363 INFO[0433] Pushed gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache@sha256:a4c9c94b28c76135095545240c83671bdeb446bcc18fc87002db017aa4758d6e
2364 INFO[0435] Pushing layer gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache:22beaa3e535c0e58c905455b9fc7c01cc06e31172da3b68b5fe46e4eb982f9b7 to cache now
2365 INFO[0435] Pushing image to gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache:22beaa3e535c0e58c905455b9fc7c01cc06e31172da3b68b5fe46e4eb982f9b7
2366 INFO[0436] Pushed gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache@sha256:0503959bebdafca73555094b1d364581f26ec4ad0caf5cd3f914db1f09bf13ac
2367 INFO[0438] Pushed gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache@sha256:2c1d0df479dd67f880213aecf3ac4324bbd2cf908a801027dc596e105d4a2ba5
2368 INFO[0616] Pushed gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch/cache@sha256:d60f4cc2d0e845a47edbdbbbbc5d60ed5ed7fb3b70403d149c98b3cefa6bdc33e
2369 INFO[0616] Pushing image to gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch:bc1aa4e7
2370 INFO[0931] Pushed gitlab-registry.nrp-nautilus.io/aomqc/deeplearning_pytorch@sha256:916d181c81af70e1e0db2ef9cc777845ec3e89c3d97a1dc82f40bf6cfa436859

```

build-and-push-...

Duration: 15 minutes 48 seconds
Finished: 8 months ago
Queued: 0 seconds
Timeout: 1h (from project)

Commit bc1aa4e7
Update Dockerfile

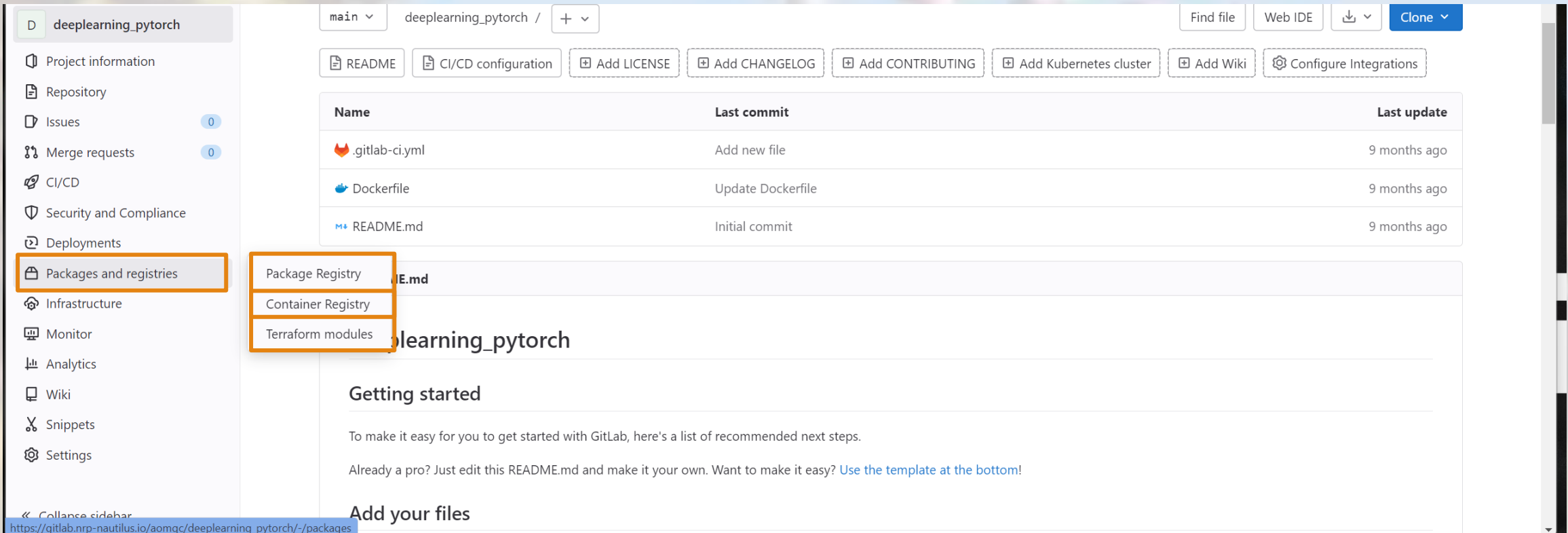
Pipeline #25928 for main

build-and-push




→ build-and-push-job

Nautilus Gitlab

create custom image



The screenshot displays the Nautilus GitLab interface for a project named 'deeplearning_pytorch'. The left sidebar contains a navigation menu with the following items: Project information, Repository, Issues (0), Merge requests (0), CI/CD, Security and Compliance, Deployments, Packages and registries (highlighted with an orange box), Infrastructure, Monitor, Analytics, Wiki, Snippets, and Settings. The 'Packages and registries' section is expanded, showing a sub-menu with 'Package Registry', 'Container Registry', and 'Terraform modules' (all highlighted with orange boxes). The main content area shows the 'main' branch of the 'deeplearning_pytorch' repository. It includes a table of files with columns 'Name', 'Last commit', and 'Last update'. The table lists three files: '.gitlab-ci.yml' (Add new file, 9 months ago), 'Dockerfile' (Update Dockerfile, 9 months ago), and 'README.md' (Initial commit, 9 months ago). Below the table, there is a 'Getting started' section with instructions on how to get started with GitLab and a link to the template at the bottom. The URL at the bottom of the page is https://gitlab.nrp-nautilus.io/aomqc/deeplearning_pytorch/-/packages.

Name	Last commit	Last update
 .gitlab-ci.yml	Add new file	9 months ago
 Dockerfile	Update Dockerfile	9 months ago
 README.md	Initial commit	9 months ago

Getting started

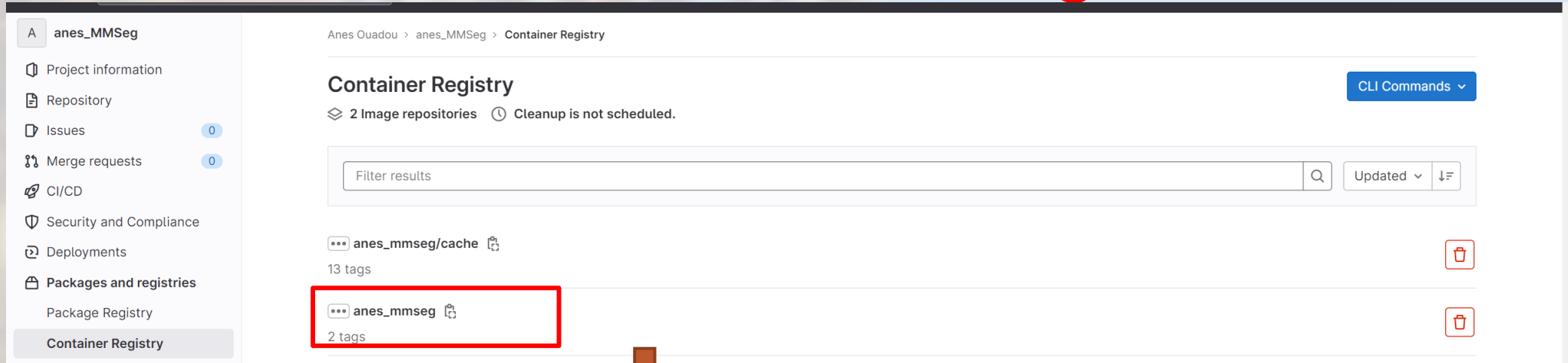
To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? [Use the template at the bottom!](#)

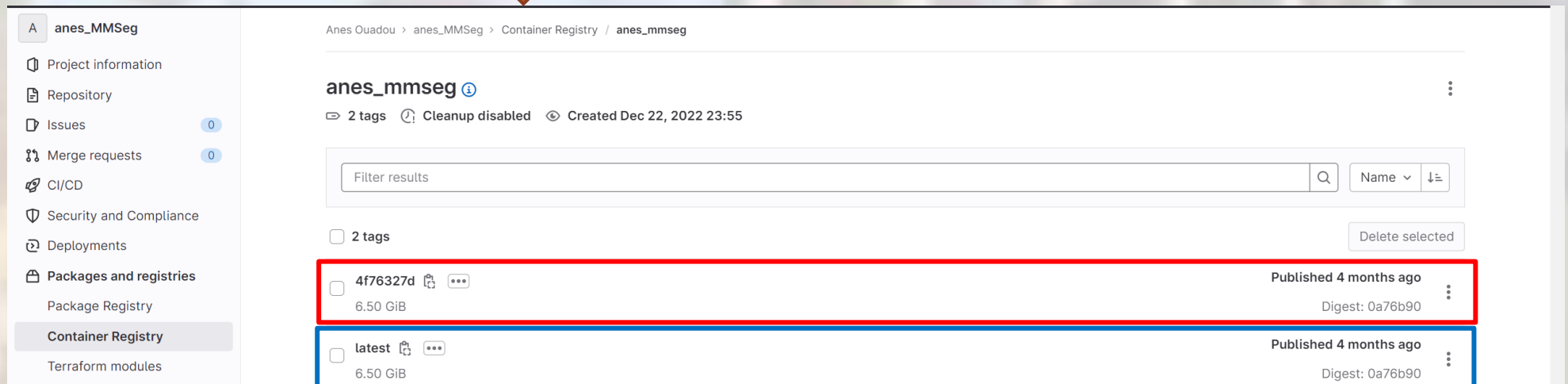
Add your files

Nautilus Gitlab

create custom image



The screenshot shows the GitLab Container Registry interface for the 'anes_MMseg' repository. The left sidebar contains navigation links: Project information, Repository, Issues (0), Merge requests (0), CI/CD, Security and Compliance, Deployments, Packages and registries, Package Registry, and Container Registry (selected). The main content area shows the 'Container Registry' for 'Anes Ouadou > anes_MMseg'. It displays '2 Image repositories' and 'Cleanup is not scheduled.' A search bar with 'Filter results' and a dropdown menu with 'Updated' and '↓↑' are present. Below the search bar, two repositories are listed: 'anes_mmseg/cache' with 13 tags and 'anes_mmseg' with 2 tags. The 'anes_mmseg' repository is highlighted with a red box. A large brown arrow points from this repository to the detailed view below.

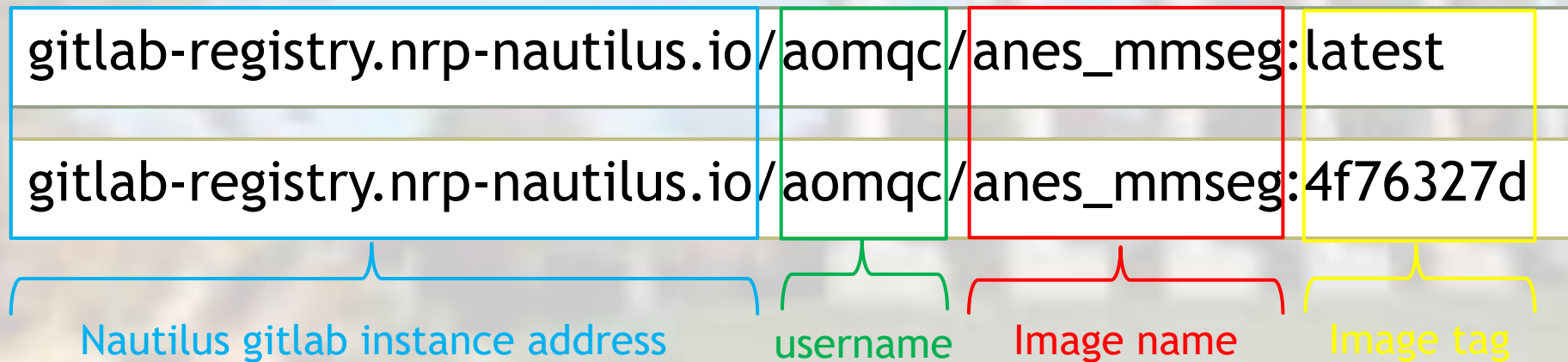


The screenshot shows the detailed view of the 'anes_mmseg' repository. The left sidebar is the same as the previous screenshot. The main content area shows the 'Container Registry / anes_mmseg' page. It displays '2 tags', 'Cleanup disabled', and 'Created Dec 22, 2022 23:55'. A search bar with 'Filter results' and a dropdown menu with 'Name' and '↓↑' are present. Below the search bar, two tags are listed: '4f76327d' and 'latest'. Both tags are 6.50 GiB and were published 4 months ago. The '4f76327d' tag is highlighted with a red box, and the 'latest' tag is highlighted with a blue box. A 'Delete selected' button is visible in the top right corner.

Nautilus Gitlab

create custom image

- The list of images will always have one image labeled latest and at least one image with a randomly generated tag



Note: Never use the latest tag in your YAML files, as it causes collision issues on nodes in the cluster

Using Common Deep Learning Frameworks

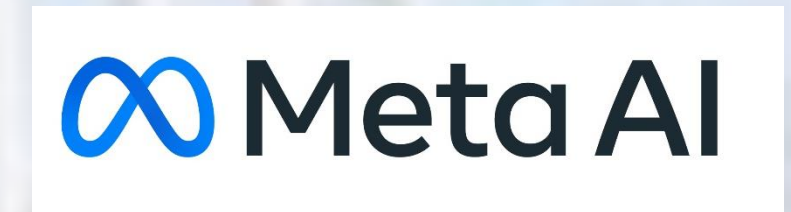
Open MMLab, FAIR, and Ultralytics



University of Missouri

Deep Learning Frameworks

- ▶ Many companies and research labs are developing open source Deep Neural Network frameworks to perform Computer Vision tasks
 - ▶ Open MMLab
 - ▶ Facebook AI Research
 - ▶ Ultralytics
 - ▶ Google
- ▶ We can build optimized Docker containers with these frameworks and use them as a baseline for research
- ▶ To use a given framework on Nautilus, we only need a Dockerfile → From the Dockerfile, we can build a container image that can be deployed on the cluster



MMLab Frameworks: MMDetection, MMClassification, MMSegmentation

- ▶ Open MMLab has developed a set of extensible libraries for performing key Computer Vision tasks:
 - ▶ Classification: MMClassification
 - ▶ Object Detection: MMDetection
 - ▶ Semantic Segmentation: MMSegmentation
- ▶ These libraries can serve as an excellent starting point for many CV applications
- ▶ Very large model zoo with community trained models and benchmarks
- ▶ Highly extensible and configurable frameworks



MMDetection Dockerfile

- ▶ Dockerfile contains all steps to create a fully functional MMDetection Python environment
- ▶ Add your code to the container or use this Dockerfile as a base in a multi-stage build

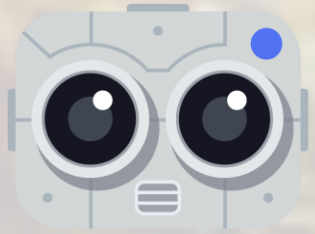
```
1 FROM nvcr.io/nvidia/pytorch:22.06-py3
2
3 # arg
4 ARG MMDET_VERSION=2.25.0
5 ARG MMCV_VERSION=1.5.2
6 ARG MMCLS_VERSION=0.23.1
7 ARG BUILD_DIR="/build"
8
9 # env variables
10 ENV MPLCONFIGDIR /tmp
11 ENV TORCH_HOME /tmp
12 ENV MMCV_WITH_OPS 1
13 ENV FORCE_CUDA 1
14 ENV CUDA_HOME /usr/local/cuda
15
16 # create the build dir
17 RUN mkdir -p ${BUILD_DIR}
18
19 RUN apt update -y && apt install -y libpng-dev libjpeg-dev libgl-dev wget && rm -rf /var/lib/apt/lists/*
20 # install mm cv
21 # https://mmdcv.readthedocs.io/en/latest/get_started/build.html#build-on-linux-or-macos
22 RUN wget https://github.com/open-mmlab/mmcv/archive/refs/tags/v${MMCV_VERSION}.tar.gz -O /tmp/mmcv.tar.gz
23 RUN tar -xzf /tmp/mmcv.tar.gz
24 RUN mv ./mmcv-${MMCV_VERSION} ${BUILD_DIR}/mmcv
25 # install it
26 RUN pip install -r ${BUILD_DIR}/mmcv/requirements/optional.txt
27 RUN pip install -v ${BUILD_DIR}/mmcv
28
29
30 # install mm detection
31 # https://github.com/open-mmlab/mmdetection/blob/v2.25.0/docs/en/get_started.md#customize-installation
32 RUN wget https://github.com/open-mmlab/mmdetection/archive/refs/tags/v${MMDET_VERSION}.tar.gz -O /tmp/mmdet.tar.gz
33 RUN tar -xzf /tmp/mmdet.tar.gz
34 RUN mv ./mmdetection-${MMDET_VERSION} ${BUILD_DIR}/mmdet
35 # install it
36 RUN pip install ${BUILD_DIR}/mmdet
```



MU-HPDI/Nautilus



University of Missouri



Detectron2

- ▶ Detectron2 is an open source framework for Object Detection and Instance Segmentation developed by FAIR
- ▶ The original Mask R-CNN model was developed and published via this framework
- ▶ Highly extensible with custom components and highly configurable
- ▶ High quality pretrained models and dataset benchmarks



Detectron2 Dockerfile

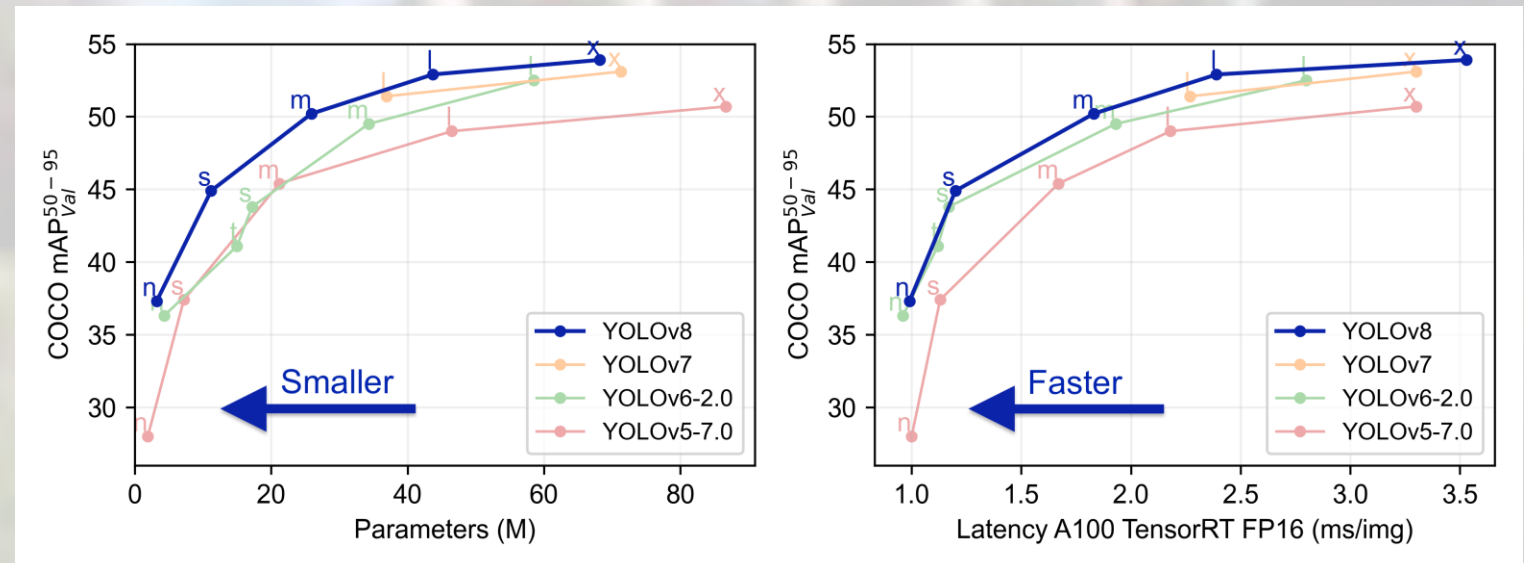
- ▶ Dockerfile contains all steps to create a fully functional Detectron2 Python environment
- ▶ Add your code to the container or use this Dockerfile as a base in a multi-stage build

```
1 FROM nvcr.io/nvidia/pytorch:21.06-py3
2
3 # install system reqs
4 RUN apt update && apt install -y vim libgl-dev
5 RUN apt-get install --reinstall ca-certificates # for git
6
7 # env variables
8 ENV MPLCONFIGDIR /tmp
9 ENV TORCH_HOME /tmp
10 ENV FVCORE_CACHE /tmp
11
12
13 #####
14 # Detectron 2
15 #####
16 RUN git clone -b 'v0.4' --single-branch --depth 1 --recursive https://github.com/facebookresearch/detectron2.git /workspace/detectron2
17 RUN pip install -v /workspace/detectron2/
```




Ultralytics

- ▶ Developer of most used Open-Source implementation of YOLOv3, YOLOv5, and more recently, YOLOv8
- ▶ You Only Look Once (YOLO) is a common real-time object detection architecture with many variations over the years
 - ▶ YOLO
 - ▶ YOLO9000 (a.k.a. YOLOv2)
 - ▶ YOLOv3
 - ▶ YOLOv5
 - ▶ YOLOv8



Ultralytics Dockerfile

- ▶ Dockerfile contains all steps to create a fully functional YOLOv3, YOLOv5, or YOLOv8 Python environment
- ▶ Add your code to the container or use this Dockerfile as a base in a multi-stage build



ultralytics/ultralytics



University of Missouri

```
1 # Ultralytics YOLO 🚀, AGPL-3.0 license
2 # Builds ultralytics/ultralytics:latest image on DockerHub https://hub.docker.com/r/ultralytics/ultralytics
3 # Image is CUDA-optimized for YOLOv8 single/multi-GPU training and inference
4
5 # Start FROM PyTorch image https://hub.docker.com/r/pytorch/pytorch or nvcr.io/nvidia/pytorch:23.03-py3
6 FROM pytorch/pytorch:2.0.0-cuda11.7-cudnn8-runtime
7
8 # Downloads to user config dir
9 ADD https://ultralytics.com/assets/Arial.ttf https://ultralytics.com/assets/Arial.Unicode.ttf /root/.config/Ultralytics/
10
11 # Install linux packages
12 # g++ required to build 'tflite_support' package
13 RUN apt update \
14     && apt install --no-install-recommends -y gcc git zip curl http libgl1-mesa-glx libglib2.0-0 libpython3-dev gnupg g++
15 # RUN alias python=python3
16
17 # Security updates
18 # https://security.snyk.io/vuln/SNYK-UBUNTU1804-OPENSSL-3314796
19 RUN apt upgrade --no-install-recommends -y openssl tar
20
21 # Create working directory
22 RUN mkdir -p /usr/src/ultralytics
23 WORKDIR /usr/src/ultralytics
24
25 # Copy contents
26 # COPY . /usr/src/app (issues as not a .git directory)
27 RUN git clone https://github.com/ultralytics/ultralytics /usr/src/ultralytics
28 ADD https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt /usr/src/ultralytics/
29
30 # Install pip packages
31 RUN python3 -m pip install --upgrade pip wheel
32 RUN pip install --no-cache -e . albumentations comet tensorboard thop pycocotools
33
34 # Set environment variables
35 ENV OMP_NUM_THREADS=1
```

NRP Nautilus (Review)

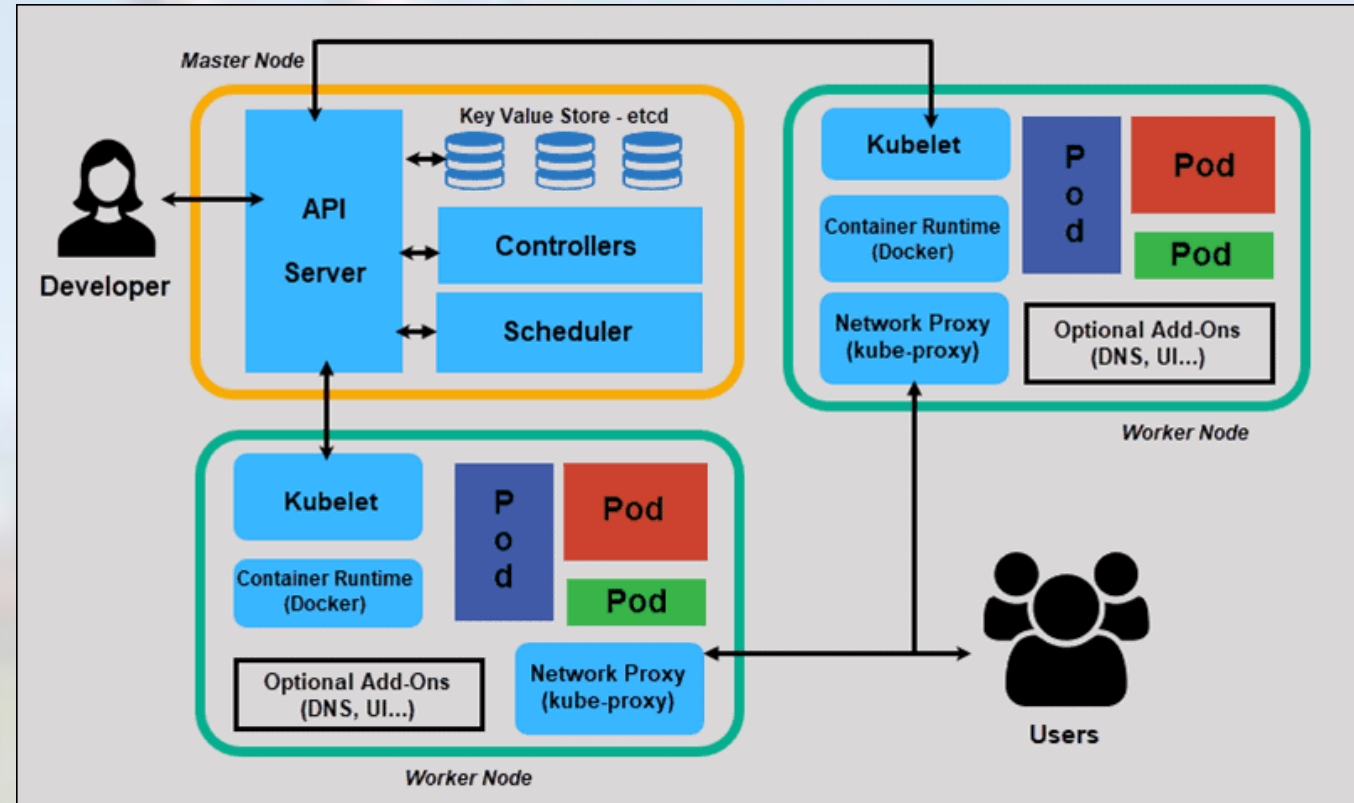
Kubernetes Architecture and Key Concepts



Kubernetes



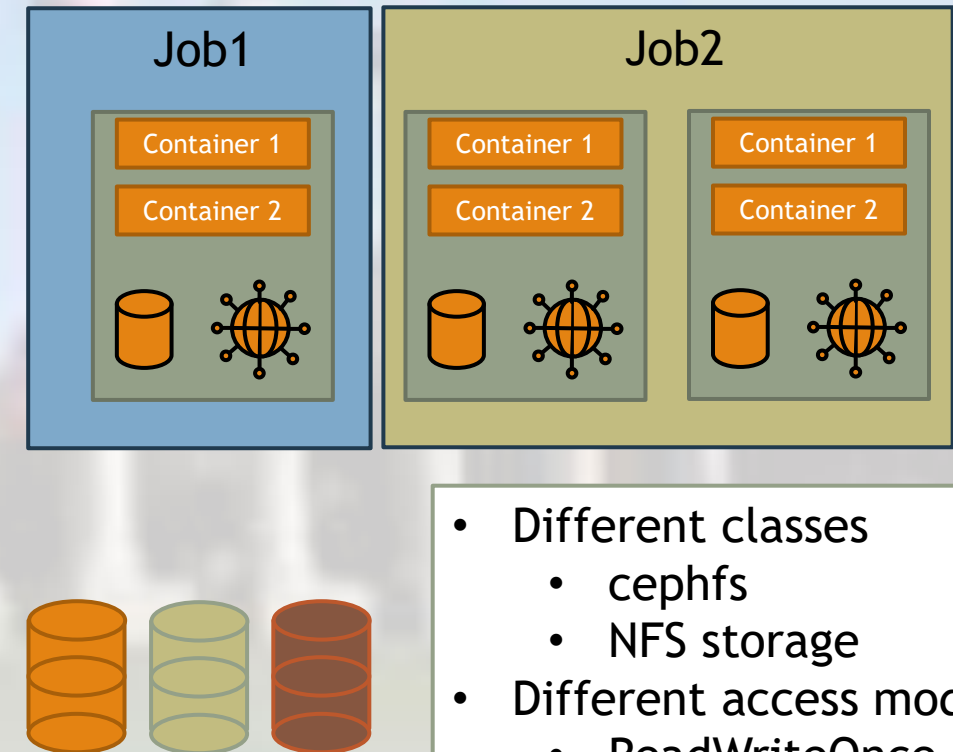
- ▶ Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.¹
- ▶ Kubernetes enables both simple and complex container orchestration
- ▶ Kubernetes cluster has two main components
 - ▶ Master node
 - ▶ Worker node



Key Kubernetes Concepts

ReplicaSet and Deployment

- ▶ **Pods** - are the basic scheduling unit of K8s.
- ▶ **ReplicaSet** - its purpose is to maintain a stable set of replica Pods running at any given time.¹
- ▶ **Deployment** - is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.¹
- ▶ **Jobs** - creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate.¹
- ▶ **Persistent volume** - is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using storage classes

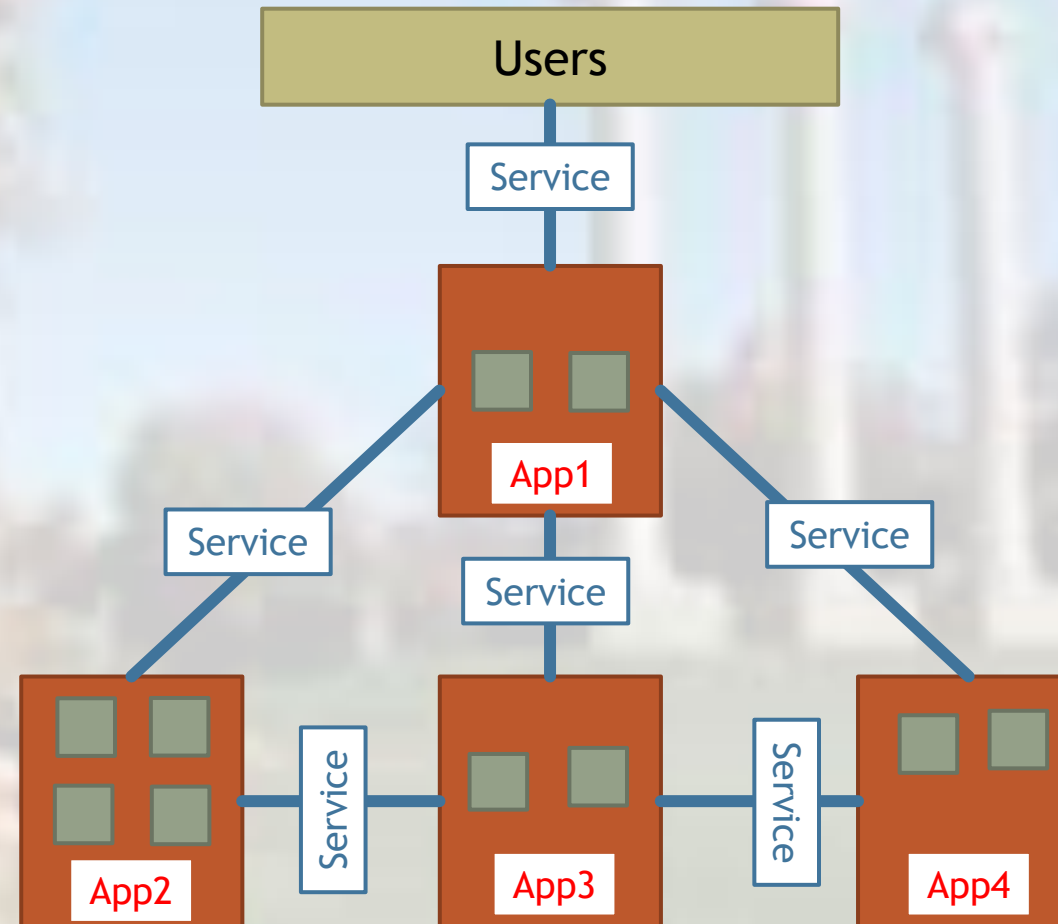


- Different classes
 - cephfs
 - NFS storage
- Different access mode
 - ReadWriteOnce
 - ReadOnlyMany
 - ReadWriteMany

Key Kubernetes Concepts

Services

Each Pod has a unique IP address which changes every time a Pod is dead and restarted, this render communication hard



Services enable communication between application running in pods within the cluster and with outside users if necessary

Yet Another Markup Language (YAML)

XML	JSON	YAML
<pre><Servers> <Server> <name>Server1</name> <owner>John</owner> <created>123456</created> <status>active</status> </Server> </Servers></pre>	<pre>{ Servers: [{ name: Server1, owner: John, created: 123456, status: active }] }</pre>	<pre>Servers: - name: Server1 owner: John created: 123456 status: active</pre>

- ▶ YAML is a key-value pair file format, similar to JSON and XML
- ▶ Kubernetes operations are performed using **YAML** files, known as a Spec file
 - ▶ Creating Persistent Storage
 - ▶ Creating Pods
 - ▶ Creating Jobs
 - ▶ Deploying services

Config file (YAML)

Pod with mounted volume

We begin by setting the API Version and the type of object we are creating (Pod), as well as the name of the pod

From here we are defining the container to run in this pod

Set the name of the container, the image the container should run, and the command that should run when the container begins

Here, we define the requested and maximum amount of resources our container needs to run, in this case that is 2 CPU cores and 10 GB of RAM

Information of the mounted volume and how it is defined within the pod

```
apiVersion: v1
kind: Pod

metadata:
  name: pod-name-sso

spec:
  containers:
    - name: pod-name-sso
      image: python:3.8
      command: ["sh", "-c", "echo 'Im a new pod' && sleep infinity"]
      resources:
        limits:
          memory: 12Gi
          cpu: 2
        requests:
          memory: 10Gi
          cpu: 2
      volumeMounts:
        - mountPath: /data
          name: anes-pv
  volumes:
    - name: anes-pv
      persistentVolumeClaim:
        claimName: anes-pv
```



NRP Nautilus

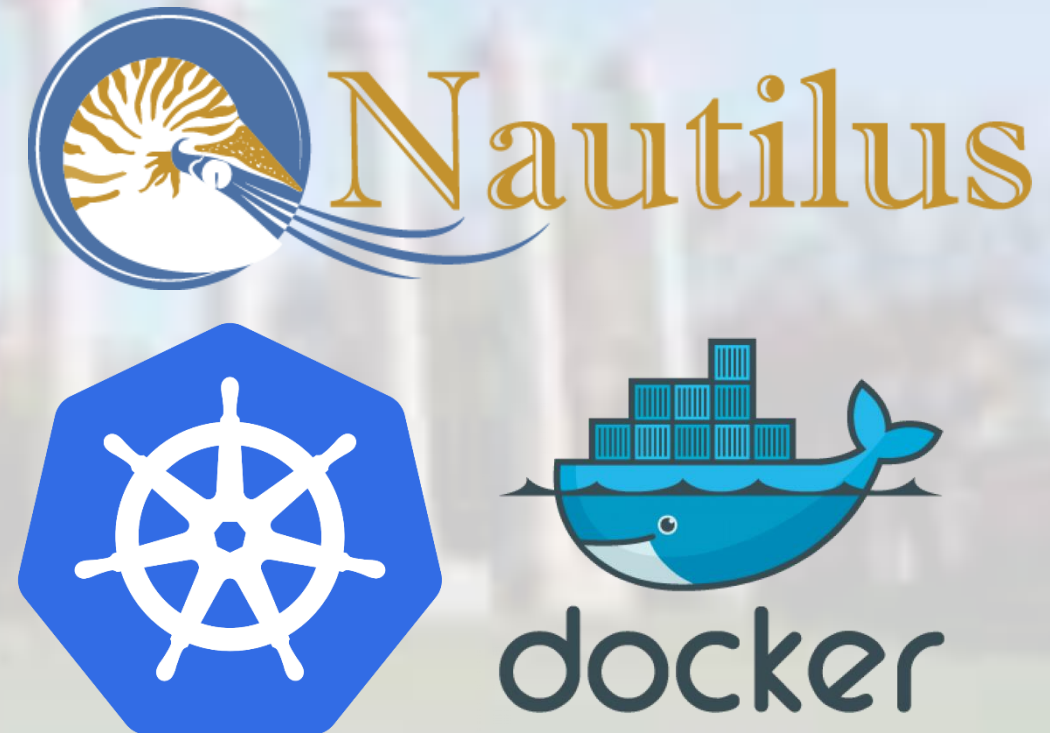
Introduction to NRP Nautilus HyperCluster



University of Missouri

NSF NRP Nautilus HyperCluster

- ▶ The NSF Nautilus HyperCluster is a Kubernetes cluster with vast resources that can be utilized for various research purposes:
 - ▶ Prototyping research code
 - ▶ S3 cloud storage for data and models
 - ▶ Accelerated small-scale research compute
 - ▶ Scaling research compute for large scale experimentation
- ▶ Resources Available:
 - ▶ CPU Cores: 14,462
 - ▶ RAM: 69 TB
 - ▶ NVIDIA GPUs: 1150



Migrating Data to Nautilus



Deep Learning & Data

- ▶ Deep Learning algorithms require a vast amount of representative training data to effectively train
 - ▶ The more complex the network architecture, the more quality data is required
- ▶ Previous portions of this workshop have covered creating the containers to train the algorithms, but we have not yet covered how to stage data to Nautilus
 - ▶ All data for use on Nautilus will need to be moved to *persistent volumes* on the cluster
- ▶ Three ways to move data to a persistent volume on Nautilus:
 - ▶ Using KubeCTL
 - ▶ Using Commercial Cloud Storage
 - ▶ Using Nautilus S3 Storage ← **Recommended**



Migrating Data to Nautilus: KubeCTL

- ▶ The command line Kubernetes tool, KubeCTL, has functionality to copy data to and from running pods
 - ▶ We can use this copy utility to move data to the cluster
- ▶ Advantages:
 - ▶ No additional installation or setup
 - ▶ No need to stage data in cloud storage
 - ▶ Requires only the KubeCTL command line tool
- ▶ Disadvantages:
 - ▶ Only small amounts (< 100 MB) of data can be moved per kubectl copy command
 - ▶ KubeCTL copy is *slow* at < 100 Mbps upload speeds
- ▶ How to:

```
kubectl cp localpath podname:/path/on/pod
```



Migrating Data to Nautilus: Using Commercial Cloud Storage

- ▶ We can use commercial cloud storage, such GCP Buckets or AWS S3 to move data to Nautilus
- ▶ Advantages:
 - ▶ Flexibility of cloud platform
 - ▶ Very fast transfer speeds
 - ▶ Capable of virtually any amount of data
- ▶ Disadvantages
 - ▶ Cost
 - ▶ Setup of Cloud Storage
 - ▶ Installation of Cloud Interface
 - ▶ Staging of data in cloud
- ▶ How to:
 - ▶ Create cloud bucket with Commercial Cloud Vendor (i.e., GCP)
 - ▶ Copy data to Bucket: `gsutil cp localpath gs://bucketName`
 - ▶ Create Google Cloud Pod on Nautilus
 - ▶ Copy data from Bucket: `gsutil cp gs://bucketName /path/on/pod`



Google Cloud



MU-HPDI/Nautilus



University of Missouri

Migrating Data to Nautilus: Using Nautilus S3 Storage



- ▶ NRP provides S3 bucket storage for free to Nautilus users upon request
- ▶ Advantages:
 - ▶ Free
 - ▶ High-throughput link to Nautilus cluster
 - ▶ Integration with S3-compatible software
 - ▶ Capability to handle large amounts of data
- ▶ Disadvantages:
 - ▶ Must request access
 - ▶ Setup of cloud integration
 - ▶ Staging of data into Nautilus S3
- ▶ How To:
 - ▶ Request access to cloud storage to receive Access ID and Keys
 - ▶ Install rclone or similar tool at data source and copy data from source to Nautilus S3
 - ▶ Create rclone or similar tool pod on Nautilus cluster and copy from S3 to Persistent Volume





**High-Performance Data-Intensive
Computing Systems Laboratory**

Part 2-B

Deploying Scikit-Learn Jobs to Nautilus for Machine Learning

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023



University of Missouri

Prerequisites

- ▶ To run jobs on Nautilus, the following prerequisites must be met:
 - ▶ You have access to Nautilus and have been assigned a namespace
 - ▶ You have a container published to a public registry with the necessary code to perform the ML task
 - ▶ You have the data for the ML task on a persistent volume in the cluster



Access



ML
Container



Data

Access the MachineLearning_K8s Notebook

Follow along activity



University of Missouri

Job Specification Format

- ▶ We will step through each of these sub-keys in the following slides
- ▶ Keep in mind key distinctions in the format between pods and jobs: the *template* key



MU-HPDI/Nautilus



University of Missouri

spec:

backoffLimit

template:

spec:

restartPolicy

containers:

- name

image

workingDir

command

imagePullPolicy

ports

env

requests

volumeMounts

volumes:

- name

persistentVolumeClaim

- name

emptyDir

affinity

Building the Job Specification: MetaData and Setup

- ▶ We initialize our job specification file by setting the API Version and setting our resource type to a Job
- ▶ We then set a Job Name
- ▶ Next, we begin the job specification and set a backoff limit
- ▶ We then create our template key followed by another specification within the template
- ▶ Finally, we set the restart policy

```
apiVersion: batch/v1
kind: Job

metadata:
  name: my-job-name

spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
```




**High-Performance Data-Intensive
Computing Systems Laboratory**

Part 2-C Deploying GPU Jobs to Nautilus for Computer Vision Applications

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023



University of Missouri

Prerequisites - Review + GPU differences

- ▶ To run GPU jobs on Nautilus, the following prerequisites must be met:
 - ▶ You have access to Nautilus and have been assigned a namespace
 - ▶ You have a **GPU enabled** container published to a public registry with the necessary code to perform the CV task
 - ▶ You have the data for the CV task on a persistent volume in the cluster



Access



GPU
Container



Data

Job Specification Format (Review)

- ▶ We will step through each of these sub-keys in the following slides
- ▶ Keep in mind key distinctions in the format between pods and jobs: the *template* key
- ▶ Full example of running a GPU job and GPU pod for deep learning available on GitHub



MU-HPDI/Nautilus



University of Missouri

spec:

backoffLimit

template:

spec:

restartPolicy

containers:

- name

image

workingDir

command

imagePullPolicy

ports

env

requests

volumeMounts

volumes:

- name

persistentVolumeClaim

- name

emptyDir

affinity

Building the Job Specification (Review): MetaData and Setup

- ▶ We initialize our job specification file by setting the API Version and setting our resource type to a Job
- ▶ We then set a Job Name
- ▶ Next, we begin the job specification and set a backoff limit
- ▶ We then create our template key followed by another specification within the template
- ▶ Finally, we set the restart policy

```
apiVersion: batch/v1
kind: Job

metadata:
  name: my-job-name

spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
```


containers:

- name: my-container-name

image: my-image

workingDir: /path/to/mydir

imagePullPolicy: IfNotPresent

command: ["python", "train.py", "/data/train_cfg.py"]

ports:

- containerPort: 8880

env:

- name: CUDA_VISIBLE_DEVICES

value: "0,1,2,3"

resources:

limits:

memory: 64Gi

cpu: 32

nvidia.com/gpu: 4

requests:

memory: 8Gi

cpu: 8

nvidia.com/gpu: 4

Building the Job Spec: Creating the Container

- ▶ For our job, we will use a single container
- ▶ We set the name (name) and the container image to use for this container (image)
 - ▶ Container image must be publicly accessible
- ▶ We set the working directory when the container starts (workingDir)
- ▶ We set the command to run when the container starts (command)
- ▶ We indicate the ports that should be open on the container (ports) as well as any environment variables we want to be present at runtime (env)
- ▶ Finally, we set the minimum (requests) and maximum (limits) amount of resources our container will need (resources)

Building the Job Spec: Mounting Data

- ▶ We need to set the Volume Mounts to the container (volumeMounts)
- ▶ We then need to specify the volumes that we will use under the spec key
 - ▶ The name of the volume mount in the volumeMounts must match the name of one of the specified volumes
 - ▶ The claim name is the PVC name given at PVC creation time
- ▶ We add an additional volume mount to deep learning containers utilizing PyTorch's Distributed Data Parallel: a shared volume mount to allow for IPC

```
spec:
  template:
    spec:
      containers:
        - name: my-container
        ...
      volumeMounts:
        - mountPath: /data
          name: my-pvc
        - mountPath: /dev/shm
          name: dshm
      volumes:
        - name: my-pvc
          persistentVolumeClaim:
            claimName: my-pvc
        - name: dshm
          emptyDir:
            medium: Memory
```



Building the Job Spec: Setting Node Affinity

- ▶ Node affinity allows us to specify what characteristics we need for the node that is assigned for our job
- ▶ Most commonly, node affinity is used to set the type(s) of GPUs for jobs
- ▶ Best practice: take the *least* powerful GPU for what you need

```
spec:
  template:
    spec:
      containers:
        - name: my-container
          ...
          volumeMounts:
            - mountPath: /data
              name: my-pvc
            - mountPath: /dev/shm
              name: dshm
      volumes:
        - name: my-pvc
          persistentVolumeClaim:
            claimName: my-pvc
        - name: dshm
          emptyDir:
            medium: Memory
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: nvidia.com/gpu.product
                    operator: In
                    values:
                      - NVIDIA-GeForce-RTX-3090
```



Starting the Job

- ▶ Once you have built a full Kubernetes Job Specification YAML file, you can begin your job with KubeCTL:

```
kubectl apply -f myJob.yaml
```

- ▶ Common Pitfalls:
 - ▶ Not ensuring that volume mount names and volume names match
 - ▶ Private container repositories
 - ▶ Using the latest tag on your deep learning container
 - ▶ Building a CPU only container for GPU code
- ▶ Your job may sit at PENDING state if:
 - ▶ You requested too many resources
 - ▶ You requested too powerful a GPU
- ▶ Your job may fail to start if:
 - ▶ There are any permission issues related to pulling the image
 - ▶ There is any exception or error thrown when the container attempts to start



Access the DeepLearning_K8s Notebook

Follow along activity



University of Missouri



**High-Performance Data-Intensive
Computing Systems Laboratory**

Part 2-D Automating Jobs on Nautilus using Bash and Python

Introduction to Containers and Kubernetes

IEEE International Conference on Big Data

December 2023



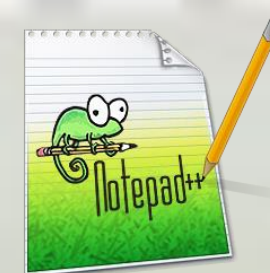
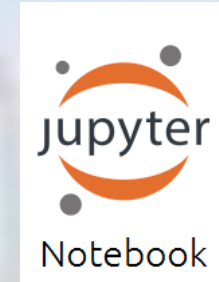
University of Missouri

Automating GPU Jobs on Nautilus using Bash and Python

- ▶ Nautilus is set up for parallel computing allowing for the running of multiple jobs at the same time
- ▶ Automation of jobs handling (submission, deletion) is key for the smooth operation
- ▶ There are multiple ways to automate the job handle processes
- ▶ We present here two ways:
 - ▶ jinja + bash
 - ▶ Nautilus Job Launcher library

jinja & bash

- ▶ We need a Python and/or Jupyter environment with these libraries:
 - ▶ yaml: to read/write yaml files
 - ▶ jinja2: to create and update templates that can be used to generate yaml files
 - ▶ os: to generate directories
- ▶ Text editor such as Notepad++ to write bash files



jinja & bash

► from jinja2 import Template

► Define template

- It needs to be a multi line string
- The variables to be updated are denoted by double braces `{{.}}`
- The name of variable between the braces is used as reference

► `j2_template1 = Template(template1)`

```
template1 = '''apiVersion: batch/v1
kind: Job
metadata:
  name: anes-job-train-exp-{{ exp_num }}-{{ network }}-{{ data_type }}-pretrain
spec:
  template:
    spec:
      containers:
      - name: anes-pod-train-exp-{{ exp_num }}-{{ network }}-{{ data_type }}-pretrain
        image: gitlab-registry.nrp-nautilus.io/jhurt/cgisegment:e98e742e
        command: ["/bin/sh", "-c"]
        args:
        - python3 main.py --task train --output_dir /canada2019-3/{{ sourcedir }}/e
        volumeMounts:
        - name: canada2019-3
          mountPath: /canada2019-3
      resources:
        limits:
          memory: 12Gi
          cpu: "4"
          nvidia.com/gpu: 2
        requests:
          memory: 12Gi
          cpu: "4"
          nvidia.com/gpu: 2
      volumes:
      - name: canada2019-3
        persistentVolumeClaim:
          claimName: canada2019-3
        restartPolicy: OnFailure
      backoffLimit:
...
'''
```



Automating GPU Jobs on Nautilus

jinja & bash

- ▶ We use a loop to auto generate the files
- ▶ We need to define variables in a dictionary where:
 - ▶ **Keys**: variable names as defined in the template
 - ▶ **Values**: values of the variables for this iteration
- ▶ Apply values to the template using:
`output_file = j2_template1.render(data)`
- ▶ Save the yaml file to the appropriate location

```
for exp in list(range(8)):
    exp_num = exp + 1

    if os.path.exists('{} /exp{}'.format(source_dir, exp_num)):
        shutil.rmtree('{} /exp{}'.format(source_dir, exp_num))
    os.mkdir('{} /exp{}'.format(source_dir, exp_num))

    for folder in folders_list:

        parts      = folder.split('_')
        network     = parts[0]
        data_type  = parts[1]

        data = {'sourcedir':source_dir,
                'exp_num':exp_num,
                'network':network,
                'data_type':data_type,
                'outputdir':dict1[folder][0],
                'configfile':dict1[folder][1]}

        output_file = j2_template1.render(data)

        fileout = open('{} /exp{} /job_exp{}_{}_{}.yaml'.format(source_dir, exp_num,
                                                                network,
                                                                data_type,
                                                                folder))
        fileout.write(output_file)
        fileout.close()
```

Automating GPU Jobs on Nautilus

jinja & bash

- ▶ Now that all yaml files have been generated we need bash files to
 - ▶ Submit jobs
 - ▶ Delete jobs after they finish
- ▶ We will write a bash file for each operation
 - ▶ Bash for job submission
 - ▶ Bash for deletion of completed jobs
- ▶ Execute bash file in the terminal

```
1 @ECHO OFF
2
3 Rem This batch file executes kubectl commands to create training jobs
4
5 ::echo %kubectl%
6 SET exp_list=2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
7
8 (for %%a in (%exp_list%) do (
9     echo %%a
10
11     kubectl create -f experiments\exp%%a/job_exp%%a_deeplab_img.yaml
12     kubectl create -f experiments\exp%%a/job_exp%%a_deeplab_tci.yaml
13     kubectl create -f experiments\exp%%a/job_exp%%a_deeplab_img_pretrained.yaml
14     kubectl create -f experiments\exp%%a/job_exp%%a_deeplab_tci_pretrained.yaml
15
16     kubectl create -f experiments\exp%%a/job_exp%%a_unet_img.yaml
17     kubectl create -f experiments\exp%%a/job_exp%%a_unet_tci.yaml
18     kubectl create -f experiments\exp%%a/job_exp%%a_unet_img_pretrained.yaml
19     kubectl create -f experiments\exp%%a/job_exp%%a_unet_tci_pretrained.yaml
20 ))
21
22 echo "batch complete"
```

```
1 @ECHO OFF
2
3 Rem This batch file executes kubectl commands to delete training jobs
4
5 ::echo %kubectl%
6 SET exp_list=1 2 3 4 5 6 7 8
7
8 (for %%a in (%exp_list%) do (
9     echo %%a
10
11     kubectl delete -f experiments_2\exp%%a/job_exp%%a_deeplabv3_img.yaml
12     kubectl delete -f experiments_2\exp%%a/job_exp%%a_deeplabv3_tci.yaml
13     kubectl delete -f experiments_2\exp%%a/job_exp%%a_deeplabv3plus_img.yaml
14     kubectl delete -f experiments_2\exp%%a/job_exp%%a_deeplabv3plus_tci.yaml
15
16     kubectl delete -f experiments_2\exp%%a/job_exp%%a_unet_img.yaml
17     kubectl delete -f experiments_2\exp%%a/job_exp%%a_unet_tci.yaml
18     kubectl delete -f experiments_2\exp%%a/job_exp%%a_unetplus_img.yaml
19     kubectl delete -f experiments_2\exp%%a/job_exp%%a_unetplus_tci.yaml
20 ))
21
22 echo "batch complete"
```

Automating GPU Jobs on Nautilus

jinja & bash

- ▶ We can use bash in addition to Powershell and Jinja2 to automate K8s job launch
- ▶ Creation of template Kube Spec YAML with environment variables (preceded by \$)
- ▶ Bash scripting combined with environment variables to set the Dataset and/or Model to train and automatically launch the job

```
spec:
  template:
    spec:
      containers:
        - name: myContainer
          image: $CONTAINER_IMAGE
          workingDir: $WORKDIR
```

Template YAML

```
Dirs="mydir1 mydir2 mydir3 mydir4"
Container="ubuntu:20.04"

for Dirpath in $Dirs; do
  CONTAINER_IMAGE=$Container WORKDIR=$Dirpath envsubst < template.yml | kubectl apply -f -
done
```

Bash Script

Nautilus Job Launcher

- ▶ This Nautilus Job Launcher is an open-source Python library that enables automation of launching jobs on the NRP Nautilus HyperCluster.

- ▶ <https://github.com/MU-HPDI/Nautilus-Job-Launcher>

- ▶ Installation:

- ▶ Use the latest .whl pushed to GitLab's PyPI repository:

```
pip3 install --extra-index-url https://gitlab.nrp-nautilus.io/api/v4/projects/2953/packages/pypi/simple nautiluslauncher
```

- ▶ You can clone this repository and use pip to install it:

```
pip3 install nautilus-job-launcher
```



Automating GPU Jobs on Nautilus

Nautilus Job Launcher

- ▶ The Nautilus Launcher can be used as
 - ▶ an application at the command line that will kick off jobs from a YAML config file
 - ▶ it can be utilized as a library integrated into other Python applications.
- ▶ You must have your Kubernetes **config** file in **~/.kube/config** to use this library!

Automating GPU Jobs on Nautilus

Nautilus Job Launcher

- ▶ Command line: The job launcher is invoked as a library and uses a configuration file (YAML):

```
python3 -m nautiluslauncher -c cfg.yaml
```

- ▶ You can choose to perform a dryrun by passing a `--dryrun` flag:

```
python3 -m nautiluslauncher -c cfg.py --dryrun
```

- ▶ `cfg.yaml`: this file contains the required configuration for the Job launcher library to work

Automating GPU Jobs on Nautilus

Nautilus Job Launcher

- Configuration requires three keys:
- Namespace (**required**):
 - the namespace on the Nautilus cluster you'd like to use
- Jobs (**required**):
 - list of dictionaries that define all of the parameters for each job
- Defaults (**optional**):
 - It is a starting place for all jobs in your config.
 - All jobs will use the defaults as the beginning configuration and then whatever is placed in each job will be added to **or override** what is present in the defaults key

Key	Description	Default	Type
job_name	The name of the job	required	str
image	The container image to use	required	str
command	The command to run when the job starts	required	str/list[str]
workingDir	Working directory when the job starts	None	str
env	The environment variables	None	dict[str, str]
volumes	The volumes to mount	None	dict[str, str]
ports	The container ports to expose	None	list[int]
gpu_types	The types of GPUs required	None	list[str]
min_cpu	Minimum # of CPU Cores	2	int
max_cpu	Max # of CPU cores	4	int
min_ram	Min GB of RAM	4	int
max_ram	Max GB of RAM	8	int
gpu	# of GPUs	0	int
shm	When true, add shared memory mount	false	bool

```
defaults:  
  container: python:3.8  
  workingDir: /mydir  
  
jobs:  
-  
  container: python:3.7  
-  
  workingDir: /mydir2  
-  
  container: python:3.7  
  workingDir: /mydir2
```


Automating GPU Jobs on Nautilus

Nautilus Job Launcher

- ▶ Library usage:
- ▶ The Job launcher can be integrated with user's application/library
- ▶ This can be done in different ways:
 - ▶ import Job launcher into the user's scripts.
 - ▶ utilize a dictionary to configure your jobs and integrate that into your application
 - ▶ from a YAML file

Automating GPU Jobs on Nautilus

Nautilus Job Launcher

import Job Launcher into the user's scripts.

```
from nautiluslauncher import Job, NautilusAutomationClient

client = NautilusAutomationClient("mynamespace")
images = ["python:3.6", "python:3.7", "python:3.8"]
for i, img in enumerate(images):
    j = Job(job_name=f"test_python_{i}", image=img, command=["python", "-c", "print('hello world')"])
    client.create_job(j)
```

Automating GPU Jobs on Nautilus

Nautilus Job Launcher

Utilize a dictionary to configure your jobs

```
from nautiluslauncher import NautilusJobLauncher

my_jobs = {
    "namespace": "mynamespace",
    "jobs": [
        {"image": "python:3.6", "command": ["python", "-c", "print('hello world')"], "job_name": "myjob1"},
        {"image": "python:3.7", "command": ["python", "-c", "print('hello world')"], "job_name": "myjob2"},
        {"image": "python:3.8", "command": ["python", "-c", "print('hello world')"], "job_name": "myjob3"}
    ]
}

launcher = NautilusJobLauncher(my_jobs)
launcher.run()
```

Automating GPU Jobs on Nautilus

Nautilus Job Launcher

from a YAML file

```
from nautiluslauncher import NautilusJobLauncher

my_file = "myCfg.yaml"

launcher = NautilusJobLauncher.from_config(my_file)
launcher.run()
```



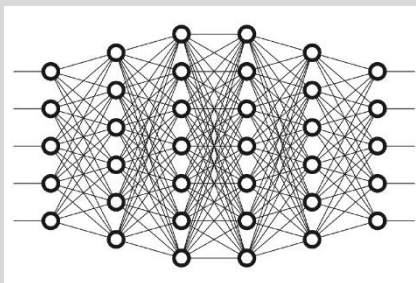

Deep Learning on Nautilus:

Semantic segmentation

- ▶ Iterations of Training Completed: **515,550**
- ▶ Number of images Processed: **7,070,400**
- ▶ Trainable Parameters Optimized: **23 millions per model**
- ▶ The time it took to prepare the experimental set up and to run all the training sessions in parallel is **12 hours**
- ▶ The actual time it would have take to train is **21 days 12 hours 45 minutes**



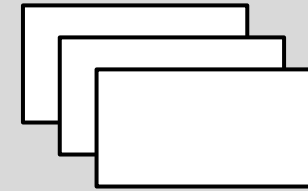
Deep Learning on Nautilus: Transformer Research



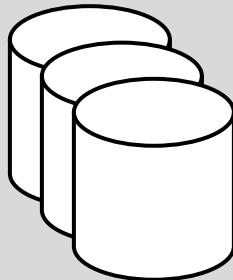
9 Deep Neural
Architectures



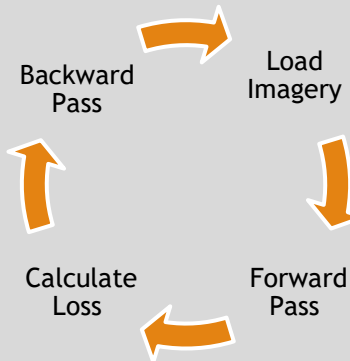
27 Trained Models



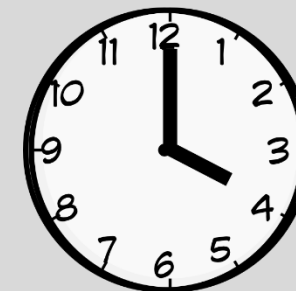
124.74 TB
Imagery
Processed



3 Open Source
HR-RSI Datasets



- 8,100 Epochs
- 30M iterations
- 1.7B parameters



Wall Clock Time:
76 days, 10 hours



MU-HPDI/Nautilus - A resource for exploration

- ▶ Sample Dockerfiles
- ▶ Sample Kubernetes YAML File
- ▶ Wiki with detailed walkthroughs for:
 - ▶ Getting Started
 - ▶ Creating PVC
 - ▶ Creating Pods
 - ▶ Creating Jobs

Check back
occasionally, we are
adding more
content and recipes
for scaling

Machine Learning
and Deep Learning

