Kubernetes for AI Research and Education

Sixth National Research Platform (6NRP) Workshop

January 28, 2025

Presented by Mahidhar Tatineni

School of Computing, Information, and Data Sciences
University of California San Diego

Ref: Tutorial developed based on updates on prior presentations at PEARC and SC conferences by Igor Sfiligoi, Dima Mishin and Mahidhar Tatineni

SLIDES AND HANDS ON MATERIAL

https://github.com/mahidhar/6nrp_k8s_tutorial

Overview of Tutorial

- Kubernetes Basics
  - Background on containers, orchestration of containers
  - Driving Kubernetes with kubectl
  - Basic examples with YAML description
  - Hands On
- Scheduling resources
  - Interactive computing
  - Hands on
- Storage considerations
- Applications
  - Using prebuilt container – LAMMPS
  - Build from source in container environment
  - AI Examples

Overview of Tutorial

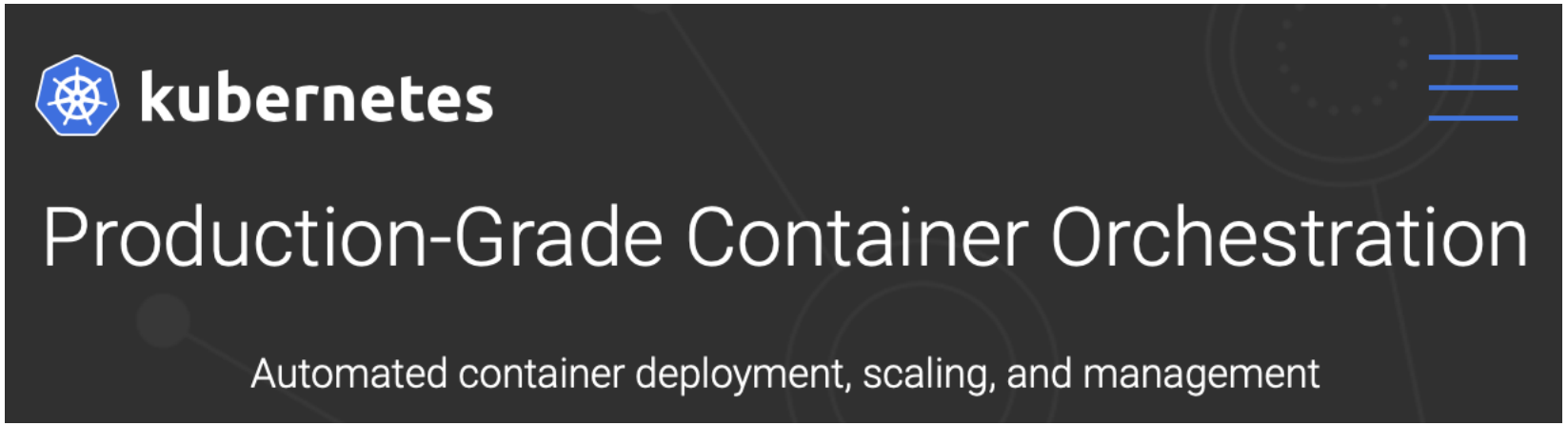
- Kubernetes Basics
  - Background on containers, orchestration of containers
  - Driving Kubernetes with kubectl
  - Basic examples with YAML description
  - Hands On
- Scheduling resources
  - Interactive computing
  - Hands on
- Storage considerations
- Applications
  - Using prebuilt container – LAMMPS
  - Build from source in container environment
  - AI Examples

A containerized world

Containers are becoming the norm	• Although many runtimes exist
Helps with code portability	• Also more efficient than VMs
Just remember containers are stateless	• If state needed, must be held outside

Container Orchestration

- Once you have many containers on many nodes, you need something to manage the whole
  - This is usually referred to as **Orchestration**



Attribution: https://kubernetes.io

Kubernetes or K8S

Originally created by Google

- Now maintained by
  Cloud Native Computing Foundation
  https://kubernetes.io

Open source

- With very large and active development community

Can be deployed anywhere

- Available in HPC centers (e.g. at SDSC)
- Also at all major Clouds (GCP, AWS, Azure)

Packing containers into pods

The smallest concept in K8S is actually the Pod

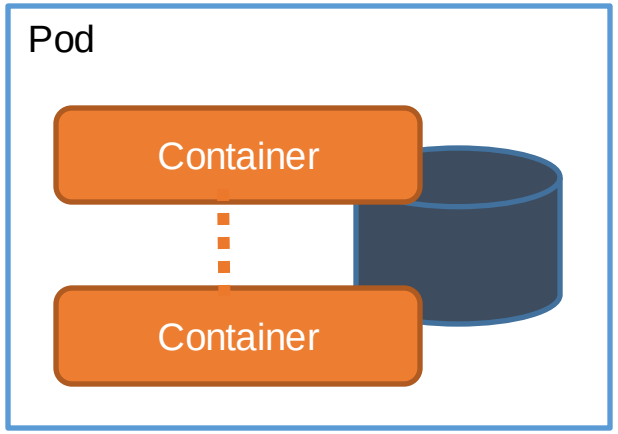
A Pod is a set of containers

- Having a single Container in a Pod OK

Containers within a Pod are guaranteed to run alongside

- And can share a local storage area

https://kubernetes.io/docs/concepts/workloads/pods/pod/



Container image

Each container must pick a container image to use

- Each container can pick its own (typically, no defaults)
- You can mix and match in multi-container pods

Images are externally hosted

- By default, they are loaded from DockerHub
- But you can provide an arbitrary URL, too

Pod scheduling

Kubernetes comes with a reasonable scheduler

Will match Pods to available resources

- Nearly instantaneous, if free compute resources available
- Else, pod will wait in line until some other pod terminates

Packing Pods into batch Jobs

A Job will make sure the pod completes (container exits with 0 exit code)

- Can retry the job up to N times

Handles pod and container failures

- e.g. if node goes offline, the job will restart elsewhere (up to backoff limit)

Facilitates parallel execution

- Including making sure all iterations succeed

Great for scientific compute

https://kubernetes.io/docs/concepts/workloads/controllers/job/

Pod scheduling

Kubernetes comes with a reasonable scheduler

Will match Pods to available resources

- Nearly instantaneous, if free compute resources available
- Else, pod will wait in line until some other pod terminates

Driving Kubernetes

No submit nodes

Cloud-native philosophy

- Any device can be used to control K8S
- Typically, your **laptop** is all you need

No shared storage areas

- You can submit from one device and monitor it through another
- **No local state on your laptop**
- **Requires explicit data movement**

kubectl most used tool

- A simple static binary
  - Available for all major platforms (Linux, MacOS, Windows)
  - Detailed download instructions (**use curl**) at https://kubernetes.io/docs/tasks/tools/install-kubectl/
- Just install it on your laptop
  - Can be used over WiFi/WAN
  - Uses a cluster-specific config file in $KUBECONFIG
    - On Linux and MacOS, if not set, defaults to ~/.kube/config
    - On Windows, it defaults to %USERPROFILE%\.kube\config

https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

Interacting with Kubernetes

kubectl most used options

- `kubectl create -f <filename>` - Create new object (e.g. a job)
- `kubectl get <type> -n <namespace>` - Query existing objects
- `kubectl edit <type> -n <namespace> <id>` - Edit existing object
- `kubectl delete -f <filename>` - Delete existing object
- `kubectl apply -f <filename>` - Create or update an object

https://kubernetes.io/docs/reference/kubectl/

YAML Everywhere

Most interactions with Kubernetes will involve YAML documents

- Both for creating/configuring Pods and Jobs
- And for querying their (detailed) status

YAML is quite easy to use

- Describes itself as "a human friendly markup language"
- Uses Python-like indentation to indicate nesting

https://en.wikipedia.org/wiki/YAML

A simple pod YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-123
spec:
  containers:
  - name: mypod
    image: ubuntu:22.04
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", ”sleep 7200"]
```

A simple pod YAML


```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-123
spec:
  containers:
  - name: mypod
    image: ubuntu:22.04
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", ”sleep 7200"]
```

A simple pod YAML

Container image to use

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-123
spec:
  containers:
  - name: mypod
    image: ubuntu:22.04
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", "sleep 7200"]
```

A simple pod YAML


```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-123
spec:
  containers:
  - name: mypod
    image: ubuntu:22.04
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", "sleep 7200"]
```

Ready to submit your first container

After you have the YAML, it is trivial

- `vim mypod-123.yaml`
- `kubectl create -f mypod-123.yaml` # Create the pod

More than just pod launching

List your pods

- Another kubectl command
  `kubectl get pods`
  - Using the
    `-o wide`
    option provides good balance between detail and readability

Understanding your workload

List your requests

Check progress

- Sometimes things don’t go as expected, and the pod is not starting
- Events provide good overview of what is happening
  `kubectl get events`
  - You will likely want to order them in chronological order with
    `--sort-by=.metadata.creationTimestamp`

Understanding your compute

List your requests

Check progress

Log into running pods

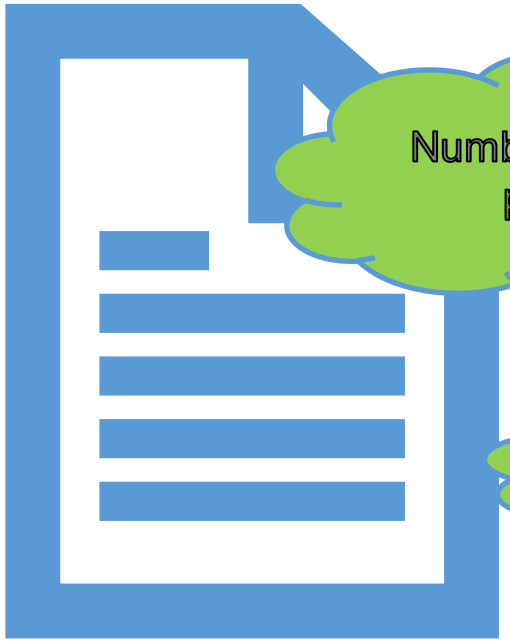
- Useful both for true interactive pods as well as for debugging
  `kubectl exec`
- By default just runs a command, but can be made interactive with
  `-it -- /bin/bash`

Putting the info so far together

The lifetime of the simple interactive pod

- `vim mypod-123.yaml`
- `kubectl create -f mypod-123.yaml` # Create the pod
- `kubectl get pods -o wide` # Check if the pod is running yet
- `kubectl exec -it mypod-123 -- /bin/bash` # Log into the node
- `kubectl delete -f mypod-123.yaml` # Delete the pod

Job example



Number of pods per job

Everything else mostly the same

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-444
spec:
  completionMode: Indexed
  completions: 10
  parallelism: 10
  ttlSecondsAfterFinished: 1800
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - name: mypod
        image: rockylinux:8
        resources:
          limits:
            memory: 100Mi
            cpu: 0.1
          requests:
            memory: 100Mi
            cpu: 0.1
        command: ["sh", "-c",
                  "let s=10+2*$JOB_COMPLETION_INDEX
                   date; sleep $s; date;
                   echo Done $JOB_COMPLETION_INDEX"]
```

Tell K8S it is a job, not just a pod

Index within the job

Nothing changes in the submission

After you have the YAML, it is trivial

- `vim myjob-444.yaml`
- `kubectl create -f myjob-444.yaml` # Create the job

Jobs are independent objects

Jobs are not pod objects

- Must use a different argument to list them
  `kubectl get jobs`

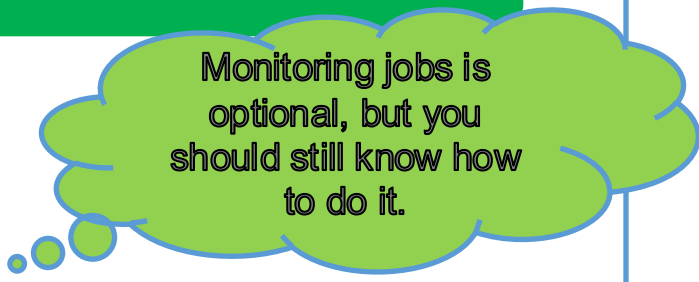
A job will create pods on your behalf

- You still list the pods the same way
  `kubectl get pods`
  - Easy to match pods to jobs,
    job just appends a hash to its name when creating the pod(s)
- You also directly interact with pods, not jobs, e.g., to fetch the stdout
  `kubectl logs <pod name>`

Putting it all together

You now have to deal with two types of objects

- `vim myjob-444.yaml`
- `kubectl create -f myjob-444.yaml` # Create the job
- `kubectl get jobs -o wide` # Get summary info about the job
- `kubectl get pods -o wide` # Check if the pod(s) are running yet
- `kubectl logs job-444-5hs46a` # Fetch the stdout (result)
- `kubectl delete -f myjob-444.yaml` # Delete the job (and associated pods)



Pod Networking

Each container get its own private IP address

- Allows for easy communication between Pods
- But new (non-deterministic) IP given every time a Pod starts

No incoming networking from WAN

- Outgoing TCP networking (typically) allowed
- Tunnel can be created to any Pod from your laptop (just like with ssh)
  `kubectl port-forward`
- We will have a hands-on example illustrating this in the applications section

Hands-on session

https://github.com/mahidhar/6nrp_k8s_tutorial

1_basic_hands_on.md

Overview of Tutorial

- Kubernetes Basics
  - Background on containers, orchestration of containers
  - Driving Kubernetes with kubectl
  - Basic examples with YAML description
  - Hands On
- Scheduling resources
  - Interactive computing
  - Hands on
- Storage considerations
- Applications
  - Using prebuilt container – LAMMPS
  - Build from source in container environment
  - AI Examples

Why scheduling?

- Every Pod has some needs
  - Memory used
  - Disk space used
  - A certain number of CPU cores
  - One or more GPU?

- The system has a finite number of resources to offer
- Some resources can be shared (to some extent)
  - e.g. CPUs
- Others cannot
  - e.g. Memory

Why scheduling?

- Every Pod has some needs
  - Memory used
  - Disk space used
  - A certain number of CPU cores
  - One or more GPU?

- The system has a finite number of resources to offer
- Some resources can be shared (to some extent)
  - e.g. CPUs
- Others cannot
  - e.g. Memory, GPUs

Scheduling finds needed resources for Pods

Pod requirements

All Pods have needs

- But they may not be fixed
- Most Pods will have different needs at different points in their lifetimes

Some requirements are critical

- e.g. Need a GPU and 4GB of RAM

Others are preferences

- e.g. Would rather use a faster GPU
- e.g. Having a 100GB NIC would be nice

Node capabilities

There is a finite amount of nodes

- Definitely true in on-prem deployments
- Cloud deployments often offer (limited) auto-scaling capabilities

Not all nodes are the same

- Each advertises its capabilities (e.g. CPU type and count, RAM size, etc.)
- Arbitrary labels can be added, too (e.g., cost)

Each node keeps track what's in use

- Critical resources are consumed as pods start (e.g. CPU cores and RAM)
- But not all (e.g. networking)

Pod scheduling

Kubernetes comes with a matchmaking scheduler

Will match Pods to available resources (CPU, Memory, GPU, etc.)

https://kubernetes.io/docs/concepts/configuration/manage-compute-resources-container/

Pod scheduling

Kubernetes comes with a matchmaking scheduler

Will match Pods to available resources (CPU, Memory, GPU, etc.)

Optimized for resource-rich environments

- Basically, FIFO scheduling
- But there is a notion of Priorities
- And nodes can be reserved (tainted) for special uses

https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

The simple pod YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod-123
spec:
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", ”sleep 7200"]
```

An example of requirements

(Simplified)

The Pod will start only if K8S can find all the resources

```
apiVersion: v1
kind: Pod
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: nvidia.com/gpu.product
            operator: In
            values:
            - "NVIDIA-A100-SXM4-40GB"
            - "NVIDIA-A10"
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      requests:
        cpu: "1"
        memory: 12Gi
        nvidia.com/gpu: "1"
```

An example of preferences

(Simplified)

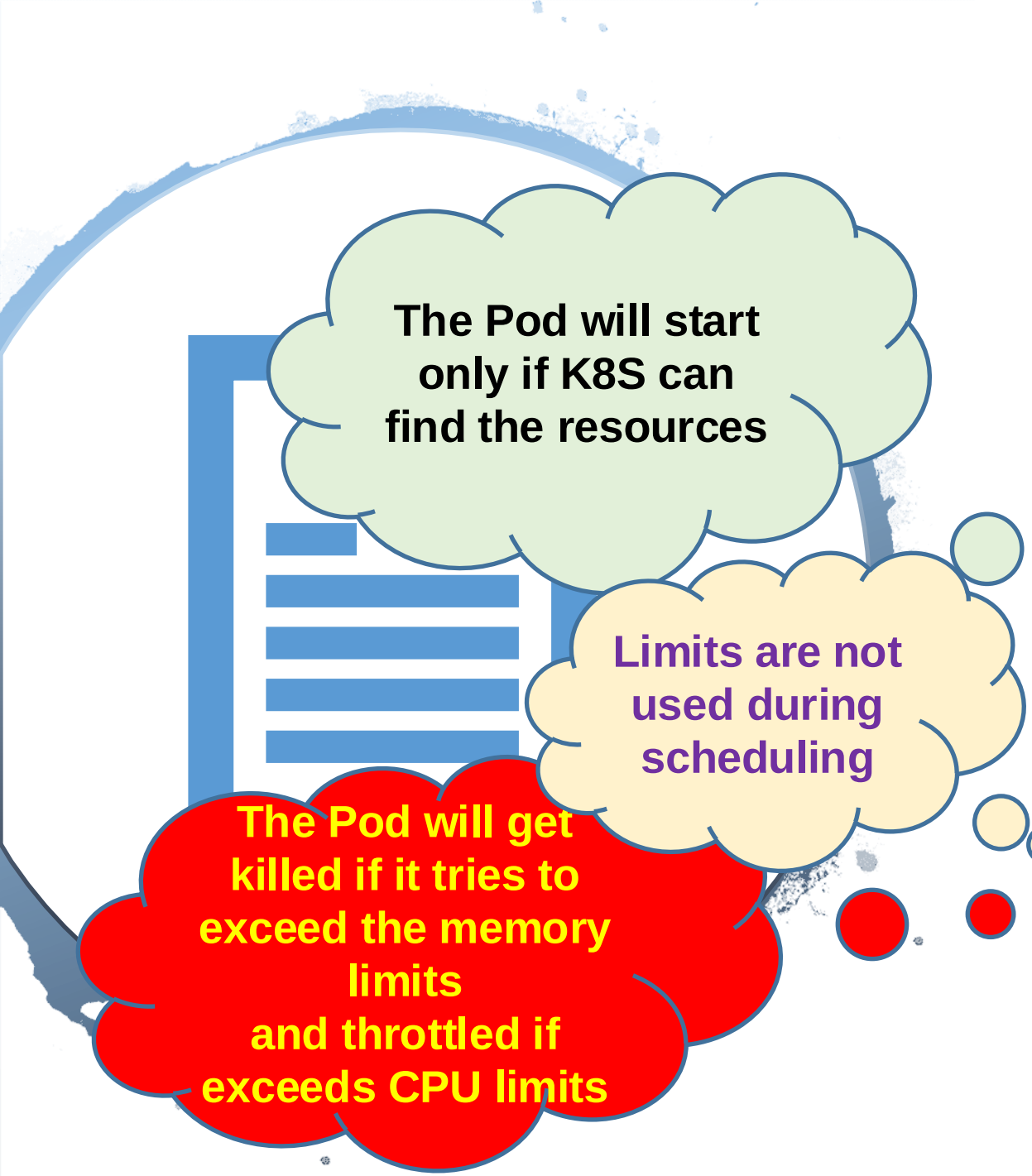
The Pod will start even if only those GPUs are available

The Pod will start only if K8S can find the requested resources

```
apiVersion: v1
kind: Pod
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: nvidia.com/gpu.product
            operator: NotIn
            values:
            - Tesla-T4
            - Quadro-M4000
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      requests:
        cpu: "1"
        memory: 12Gi
        nvidia.com/gpu: "1"
```

An example of limits

(Simplified)




```
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      requests:
        cpu: "1"
        memory: 12Gi
        nvidia.com/gpu: "1”
      limits:
        cpu: ”2"
        memory: 16Gi
        nvidia.com/gpu: "1”
```

INTERACTIVE USE OF KUBERNETES

- In the first part of the tutorial, we saw how to interactively access resources on running pods. That is the simplest interactive approach using Kubernetes.
- Can easily run interactive applications and services on Kubernetes so that can be leveraged. For example, on Nautilus we have:
  - Jupyter Hub running that can spawn Jupyterlab sessions for end users to do interactive development – used for CPU, GPU, and FPGA code development
  - Gitlab integration allows for changing options easily and we provide our implementation details so others can reuse
  - Coder based development environment running using Kubernetes. Allows for interactive container builds, CPU, GPU, and FPGA application development

INTERACTIVE COMPUTING EXAMPLE:
JupyterHubs on Nautilus Cluster

- PNRP Staff maintained and supported JupyterHub
  - Institutional credentials to login using CILogon
  - Provide several container images

- Institutions/Research Groups have cloned and customized our JupyterHub instances
  - Over 75 JupyterHubs
  - Admins share information and recipes via gitlab and Matrix channel

Interactive Jobs: Jupyterhub

- https://jupyterhub-west.nrp-nautilus.io
- Template setup to use CILogon
- Choice of prebuilt images for environment - can choose PyTorch or TensorFlow images for example
- Choose resources: number of cores, gpus, and amount of memory

Interactive Jobs: Jupyterhub

Server Options

/home/jovyan is persistent volume, 5GB by default. Make sure you don't fill it up - jupyter won't start next time. You can ask admins to increase the size.
The storage is created in West ceph pool by default. You can ask admins to move it to a different region.

Available resources page

Contact admins in Matrix.

Region

Any

GPUs

0

Cores

1

RAM, GB

8

GPU type

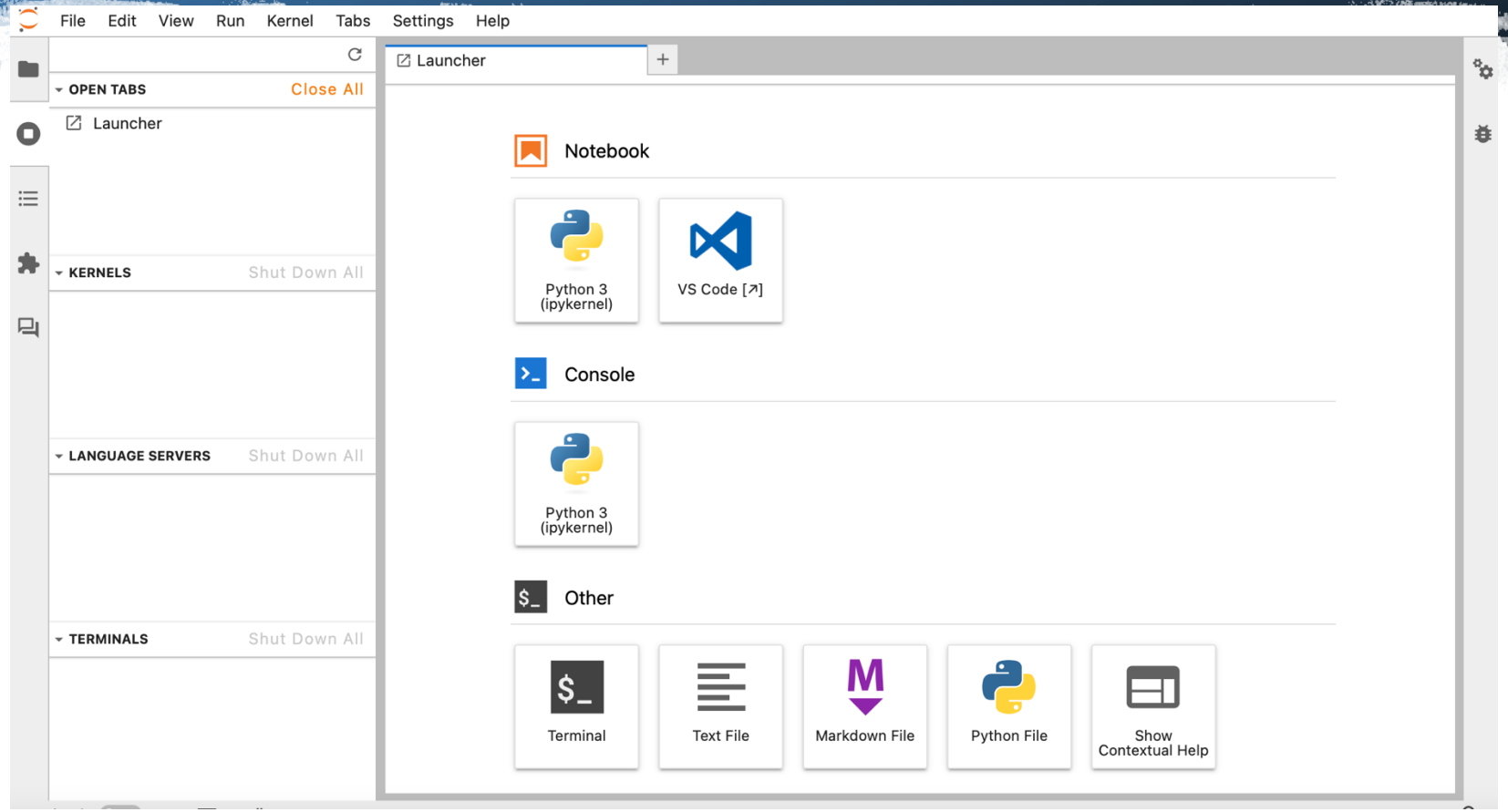
Any

/dev/shm for pytorch

Image

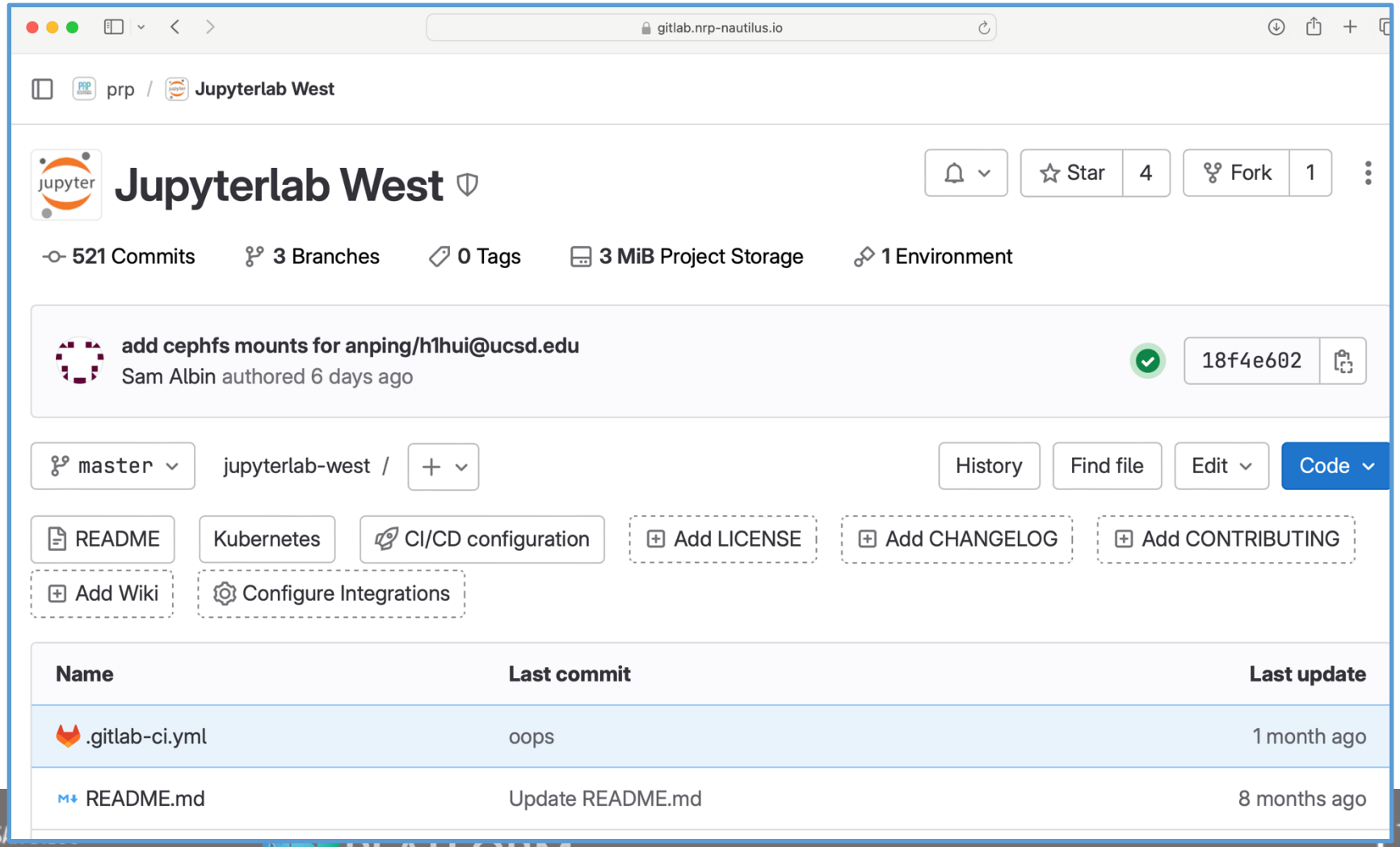
- Stack Minimal
- Stack Scipy
- Stack R
- Stack Julia
- Stack Tensorflow
- Stack Datascience (scipy, Julia, R)
- Stack Pyspark
- Stack All Spark
- NRP Stack Tensorflow + (only default tag, v1.3.1)
- NRP Stack Tensorflow +, other tags
- NRP Desktop GUI
- NRP Desktop GUI + Relion
- NRP Desktop GUI + PRISM
- NRP R Studio Server
- NRP Matlab
- NRP OSGEO

Interactive Jobs: Jupyterhub



Production JupyterHub West Gitlab Repository

- JupyterHub service provided by NRP allows for multiple users to spin up Jupyter instances with chosen containers. Repo has been cloned and customized by several institutions/research groups. Each instance spins up JupyterLab web-based interactive development environment for notebooks, code, and data



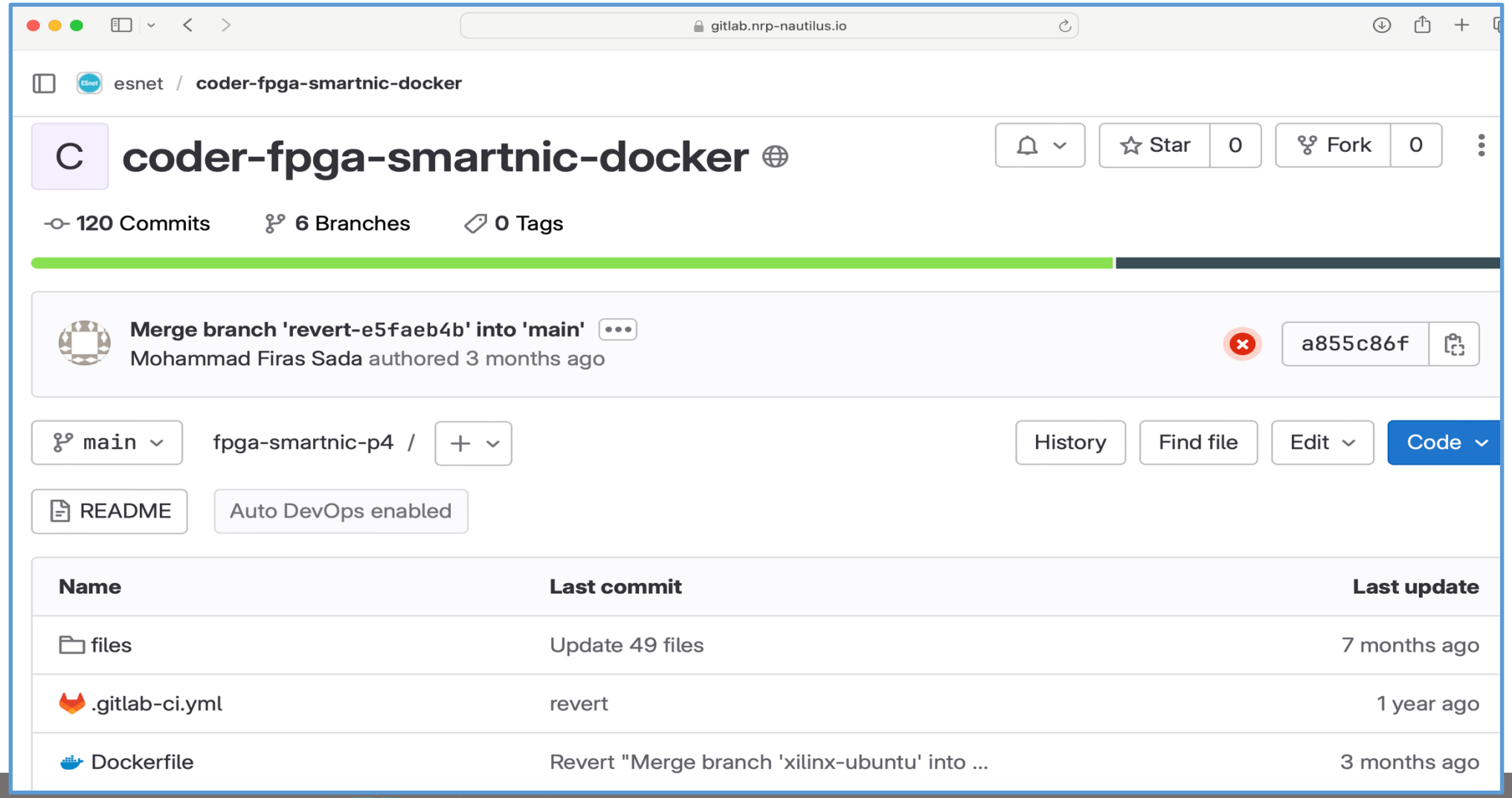
JupyterHub Deployment

- Step by step guide:

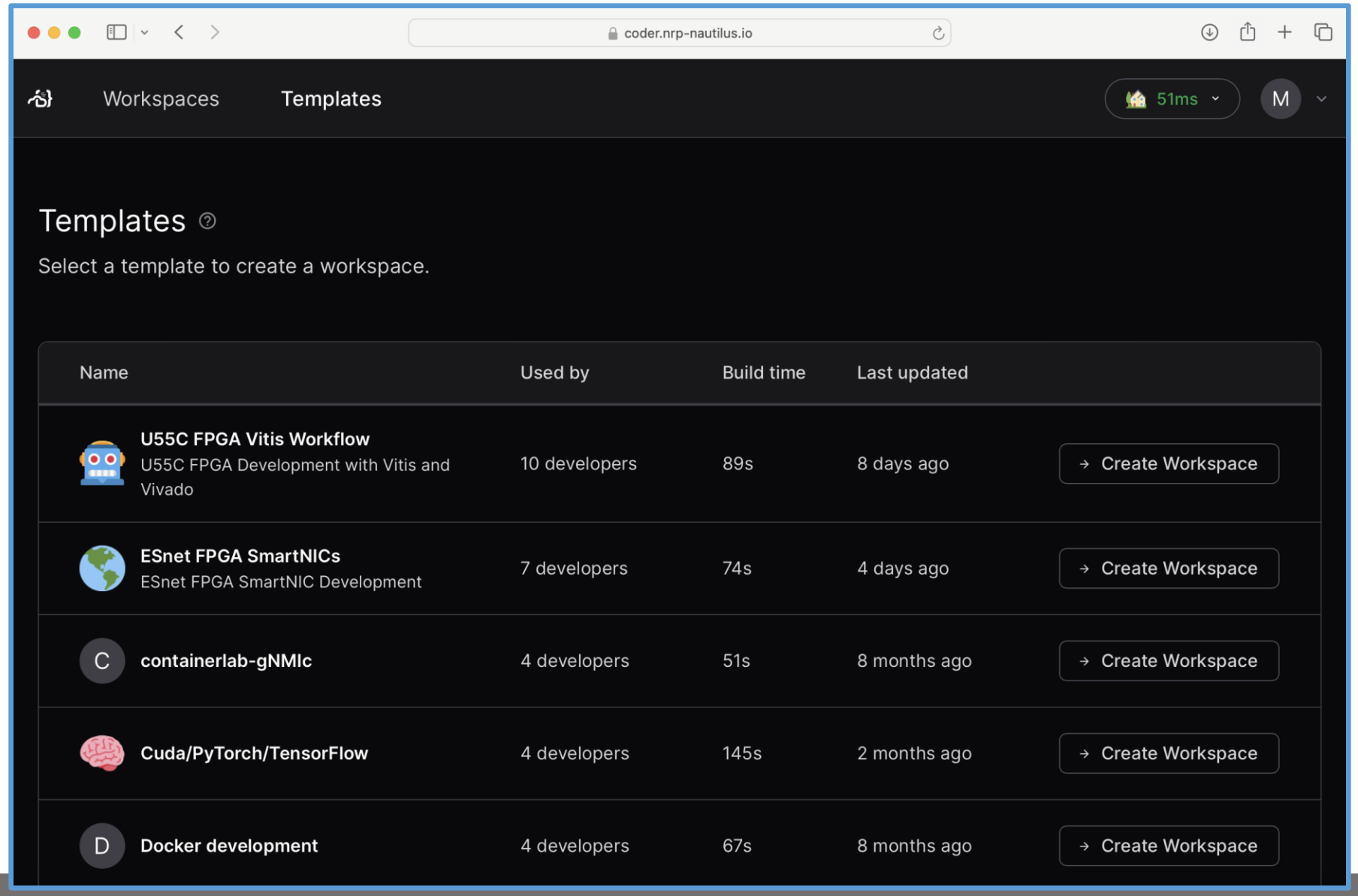
- https://docs.nationalresearchplatform.org/userdocs/jupyter/jupyterhub/

INTERACTIVE DEVELOPMENT

CODER FPGA development example from PNRP Staff



Coder Templates Examples



Additional Scheduling Topics

An example of priority

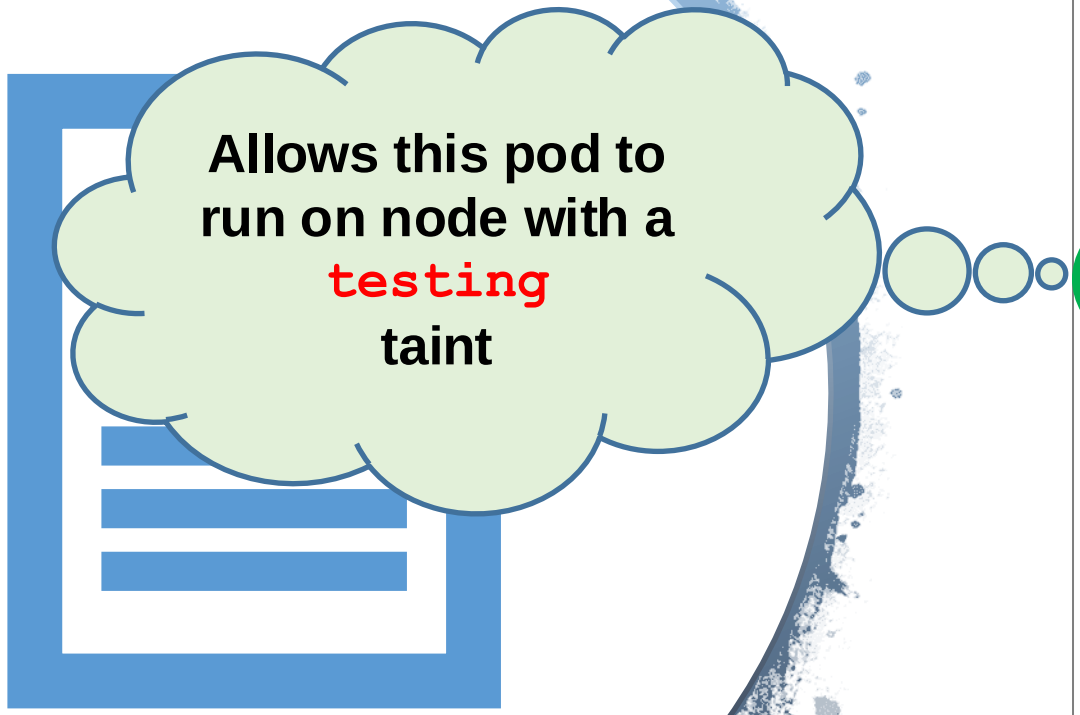
System wide setup

```
apiVersion: scheduling.k8s.io/v1beta1
kind: PriorityClass
metadata:
  name: opportunistic
value: -2000000000
globalDefault: false
```

```
apiVersion: v1
kind: Pod
spec:
  priorityClassName: opportunistic
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      requests:
        cpu: "1"
        memory: 12Gi
        nvidia.com/gpu: "1"
```

https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/

An example of toleration




```
apiVersion: v1
kind: Pod
spec:
  tolerations:
  - effect: NoSchedule
    key: nautilus.io/testing
    operator: Exists
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      requests:
        cpu: 6
        memory: 24Gi
```

https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/

MPI jobs

- Kubernetes does not natively support MPI jobs
  - Or gang-scheduling in general
- Several external projects help getting a cluster to support them, e.g.
  - kubeflow: https://www.kubeflow.org/docs/components/training/mpi/
  - kube-openmpi: https://github.com/everpeace/kube-openmpi

MPI jobs

- Kubeflow MPI allows for specs on both launcher (where mpirun command runs) and worker nodes (where the actual MPI tasks land)
- There are pods on the launcher node and each of the worker nodes
- stdout/stderr from the MPI job is in the launcher logs as that is where the mpirun is executed
- Example job showing the launcher and worker pods below

```
mahidhar@vgr-1-20:~$ kubectl get pods -n default | grep mpiexample
mpiexample-mahi-launcher-nqbth          1/1     Running             0          35s
mpiexample-mahi-worker-0                1/1     Running             0          35s
mpiexample-mahi-worker-1                1/1     Running             0          35s
mahidhar@vgr-1-20:~$
```

Hands-on session

https://github.com/mahidhar/6nrp_k8s_tutorial

2_scheduling_hands_on.md

Overview of Tutorial

- Kubernetes Basics
  - Background on containers, orchestration of containers
  - Driving Kubernetes with kubectl
  - Basic examples with YAML description
  - Hands On
- Scheduling resources
  - Interactive computing
  - Hands on
- Storage considerations
- Applications
  - Using prebuilt container – LAMMPS
  - Build from source in container environment
  - AI Examples

Ephemeral storage

(work area while the pod is running)

Storage inside the container image

- All areas inside the container are writable (typically)
- You can write data straight into the directories provided by the image

Ephemeral partition

- Sometimes you need a larger and faster partition
- Kubernetes allows for an explicit ephemeral mount
- Known as an emptyDir volume

RAM disk

- As with all Linux systems, RAM disk is mounted in all containers
- But (typically) by default limited to 64M
- Must explicitly request a larger one (memory-based emptyDir)

https://kubernetes.io/docs/concepts/storage/volumes/#emptydir

Using external storage

Important to pick the right type of storage to meet your IO needs

- For IO workloads with heavy metadata loads – i.e. lots of small files, extremely high rates of file opens/closes etc
  - Node local ephemeral storage is the best option
  - You can use CVMFS as a solution if you are read only as the filesystem will locally cache more frequently read data
  - Block storage can be a good option if there are frequent updates
  - Do not use CephFS for this type of workload as it can cause high loads on the filesystem and slow it down for everyone
- CephFS is good for IO workloads with large files and can provide high I/O performance in such cases

STRATEGIES for HIGH IOPs JOBS

- Copy dataset into node local NVMe storage if it fits
- Open multiple parallel streams (such as using gcloud storage / gsutil), aggregate many small files to a few large files that can be stored in Ceph
- Can use parallel archiver tools such as 7-Zip with LZMA2, pbzip2, ripunzip/piz, or zpaq to go from/to a Ceph filesystem
- Store your data in a S3 location, download data to local scratch or RAM disk in a rolling window and then have your code sample from the local data

Mounting storage

Pick the volume to mount

- You may be able to create it at Pod creation type
- But most persistent storage pre-created as Persistent Volume Claims (PVC)

Mount it inside the container

- Any directory path will work
- Whatever works for you

Example PVC creation yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vol-mahidhar
spec:
  storageClassName: rook-cephfs
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

Example PVC mount yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: s3-mahidhar
spec:
  completionMode: Indexed
  completions: 10
  parallelism: 10
  ttlSecondsAfterFinished: 1800
  template:
    spec:
      restartPolicy: OnFailure
      containers:
      - name: mypod
        image: rockylinux:8
        resources:
          limits:
            memory: 100Mi
            cpu: 0.1
          requests:
            memory: 100Mi
            cpu: 0.1
        command: ["sh", "-c", "let s=2*$JOB_COMPLETION_INDEX; d=`date +%s`; date; sleep $s; (echo Job $JOB_COMPLETION_INDEX; ls -l /mnt/mylogs/) >
/mnt/mylogs/log.$d.$JOB_COMPLETION_INDEX; sleep 1000"]
        volumeMounts:
        - name: mydata
          mountPath: /mnt/mylogs
      volumes:
      - name: mydata
        persistentVolumeClaim:
          claimName: vol-mahidhar
```

Example CVMFS mount yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: s4-mahidhar
spec:
  containers:
  - name: mypod
    image: rockylinux:8
    resources:
      limits:
        memory: 1Gi
        cpu: 1
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sleep", "1000"]
    volumeMounts:
    - name: cvmfs
      mountPath: /cvmfs
      readOnly: true
      mountPropagation: HostToContainer
  volumes:
  - name: cvmfs
    persistentVolumeClaim:
      claimName: cvmfs
```

Fetching the output

Stdout and stderr can be accessed at any time

- `kubectl logs <pod name>`

If you used persistent storage, output can be stored and remain there

Or you can explicitly copy it out

- Pick you favorite (non-K8S) tool (e.g. S3, Globus, scp)

Hands-on session

https://github.com/mahidhar/6nrp_k8s_tutorial

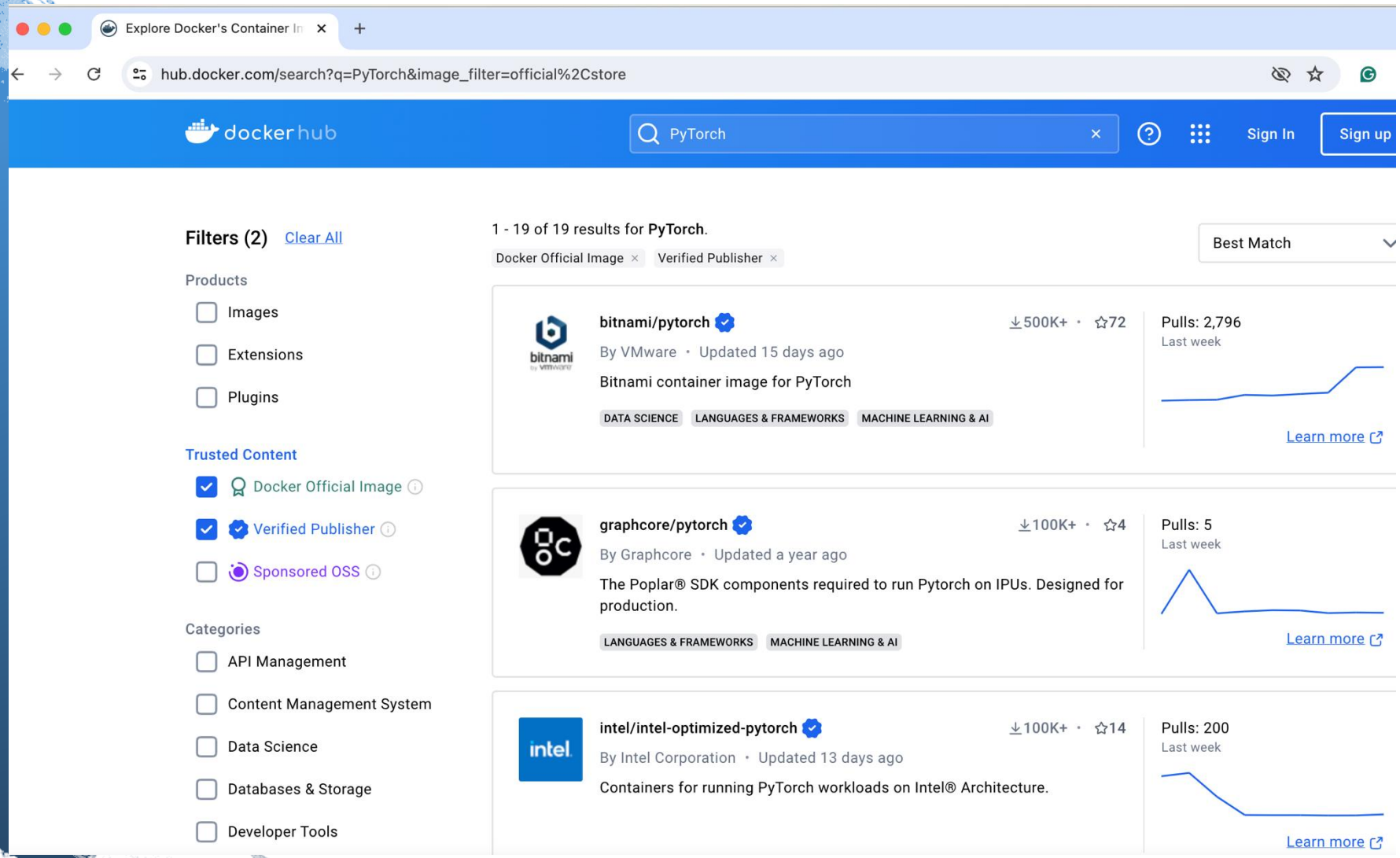
3_storage.md

Overview of Tutorial

- Kubernetes Basics
  - Background on containers, orchestration of containers
  - Driving Kubernetes with kubectl
  - Basic examples with YAML description
  - Hands On
- Scheduling resources
  - Interactive computing
  - Hands on
- Storage considerations
- Applications
  - Using prebuilt container – LAMMPS
  - Build from source in container environment
  - AI Examples

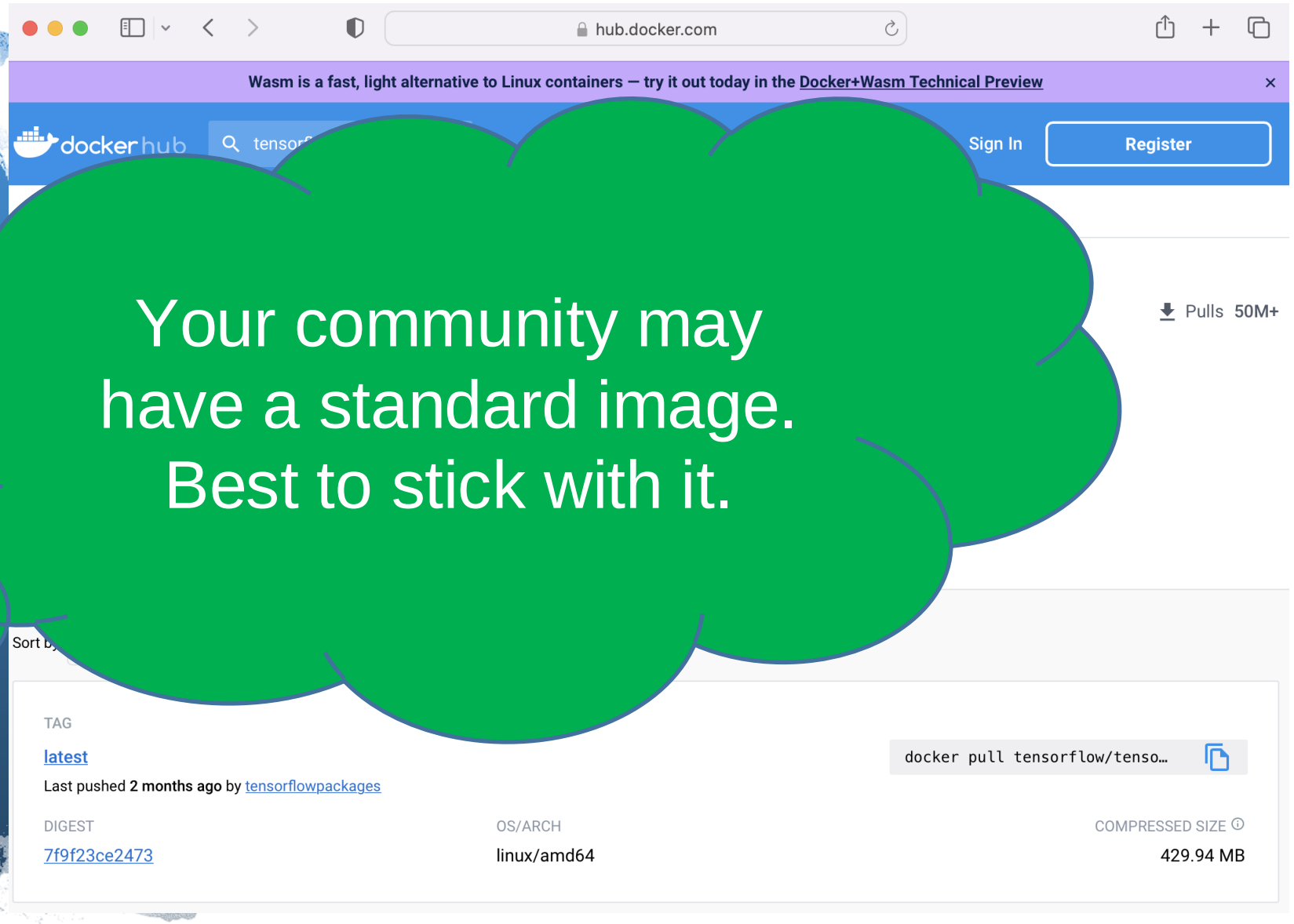
https://hub.docker.com/

Find your container images



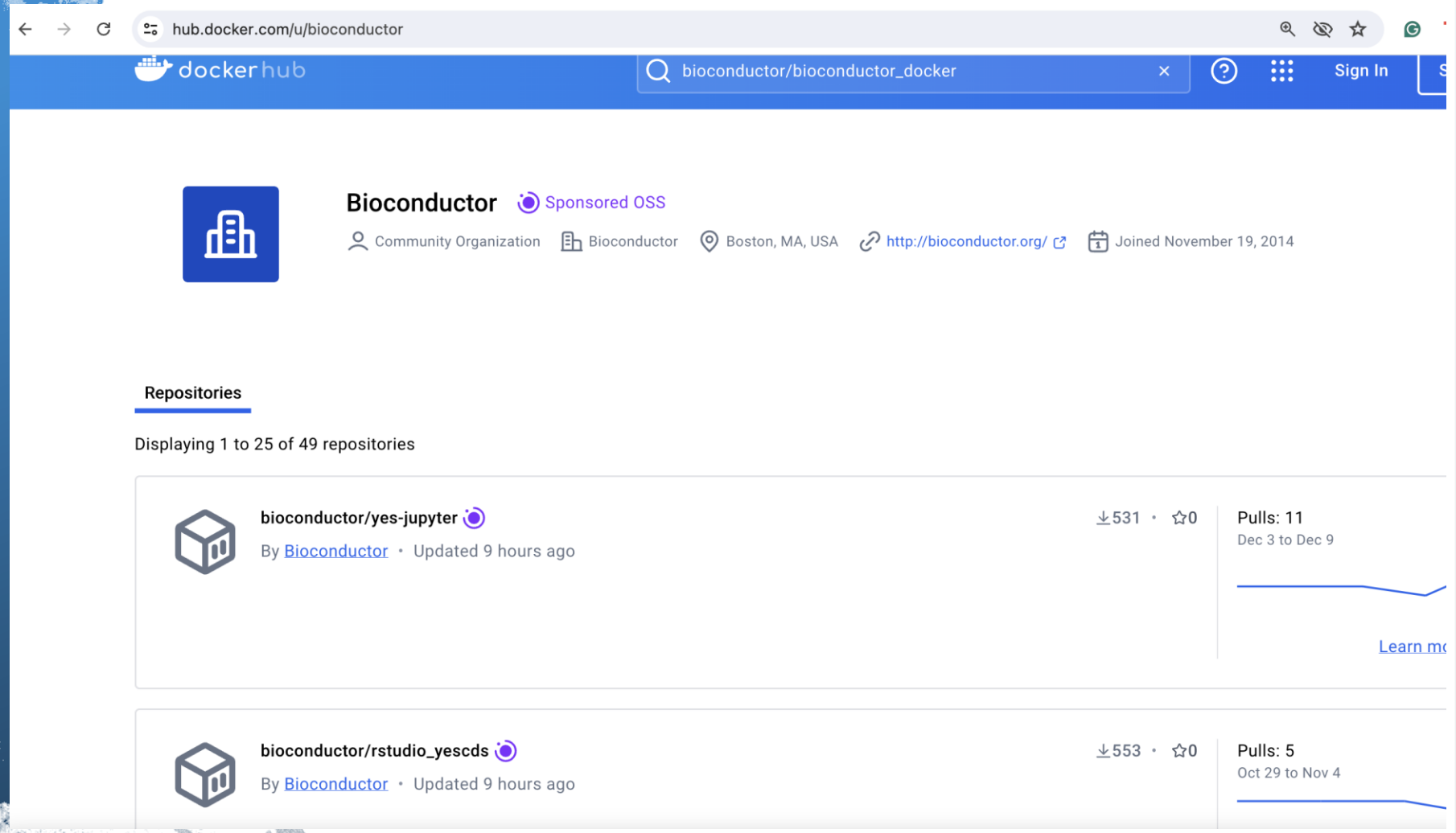
https://hub.docker.com/

Find your container images



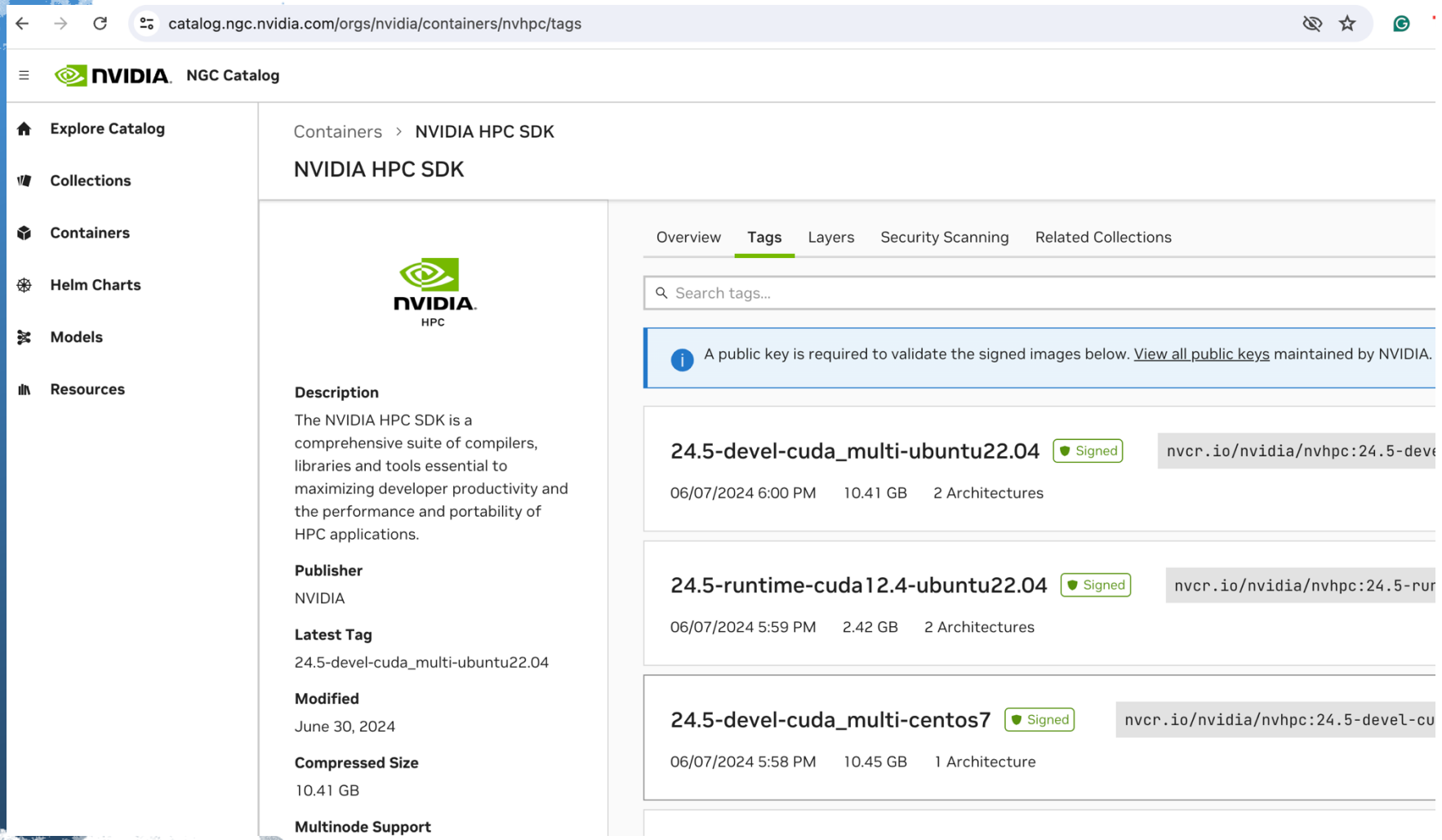
https://hub.docker.com/

Community maintained images



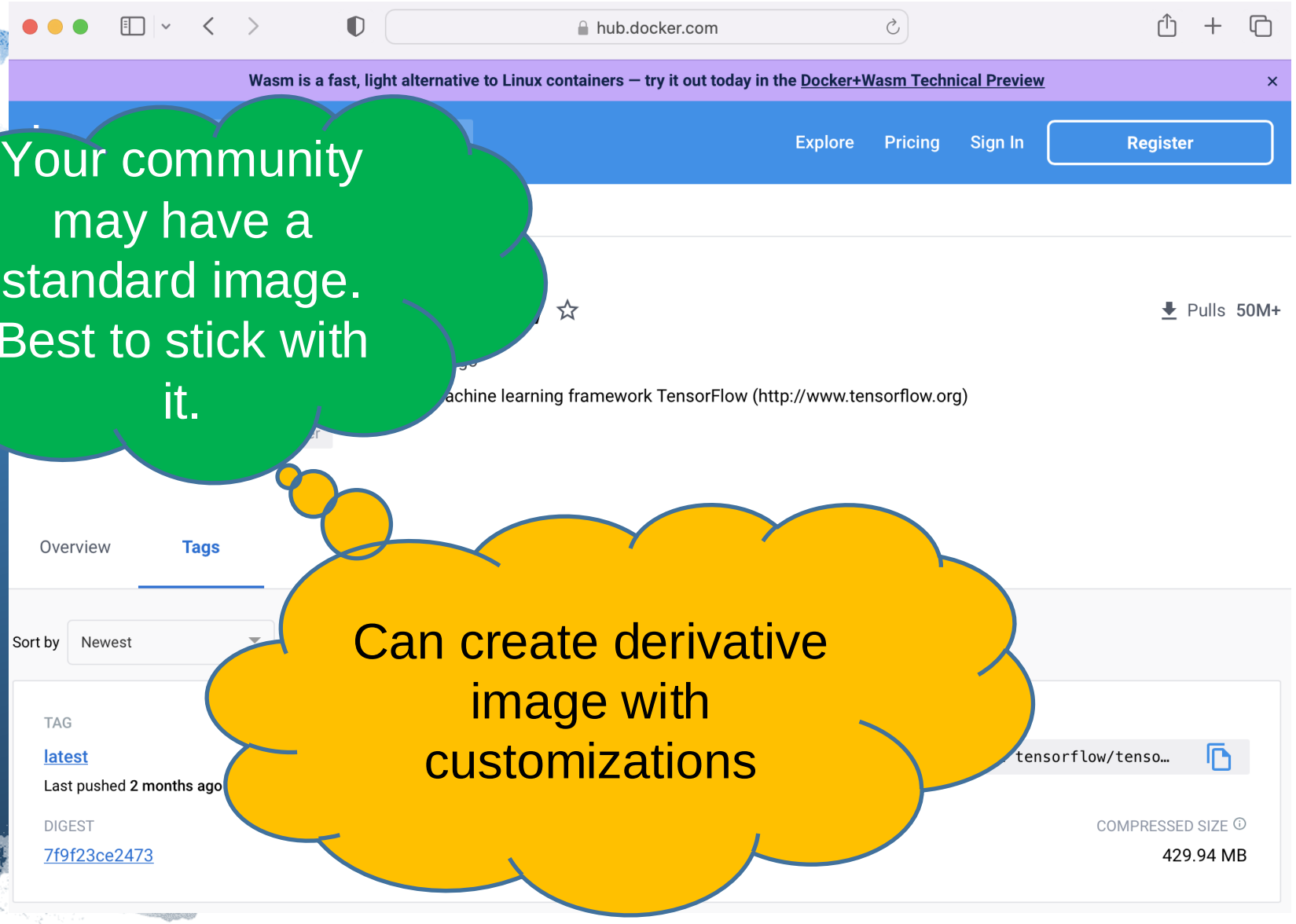
Vendor optimized and signed images

https://catalog.ngc.nvidia.com/
(Look for public images)



https://hub.docker.com/

Find your container images



Your community may have a standard image. Best to stick with it.

Can create derivative image with customizations

Explore the rich ecosystem

Kubernetes is quite powerful by itself

- But it is the surrounding ecosystem that makes it so valuable!

Kubernetes has become an industry standard

- Many projects build on top of it to deliver additional capabilities

Helm package manager

Helm has become the main package manager

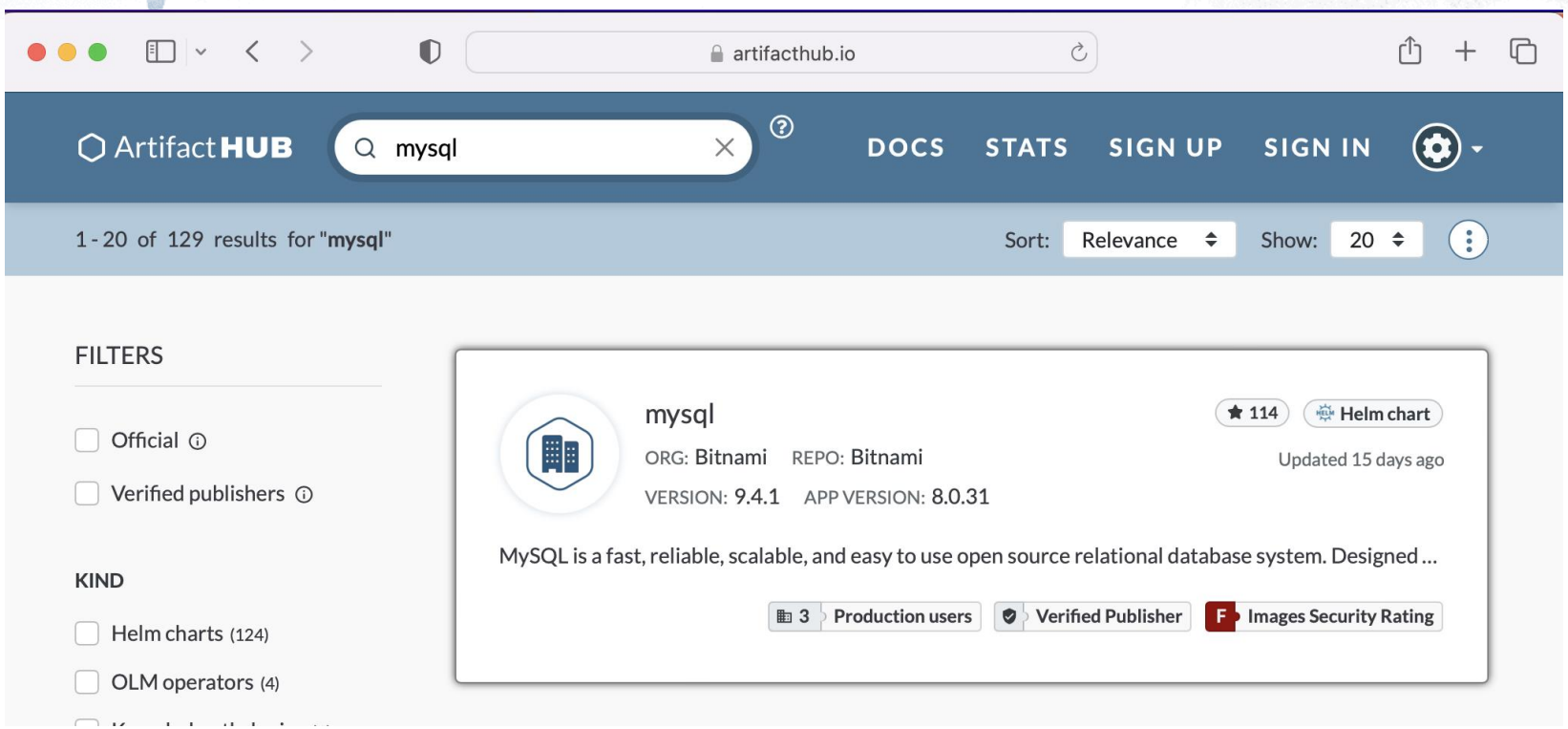
- Many pods can be bundled in a coherent package
- Known as “helm charts”

https://helm.sh

Typically used to deploy new capabilities

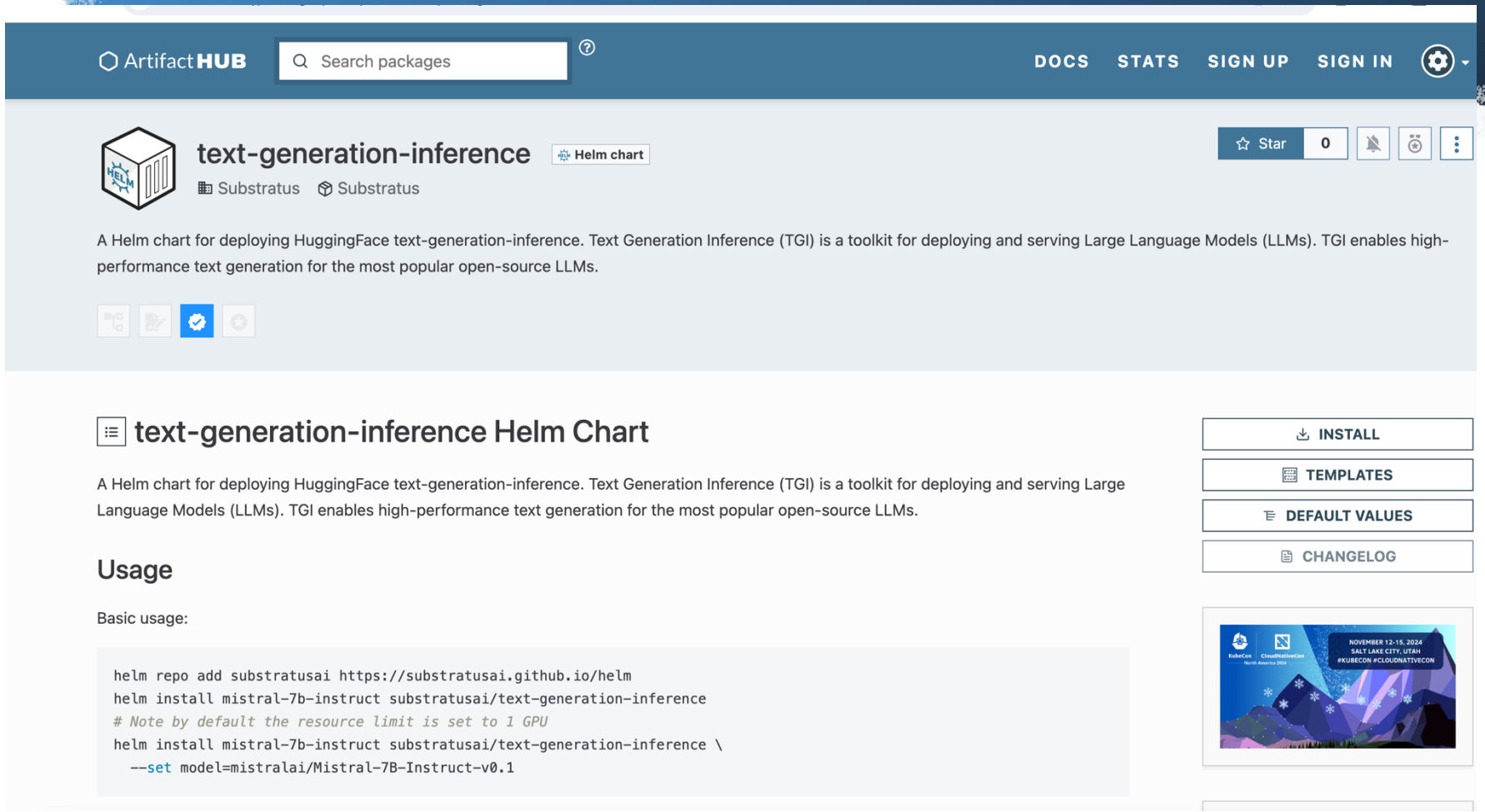
- e.g. multi-pod services
- But can be used for complex science workflows, too
- We will have a LLM as a service deployment example in hands-on session

Large catalogue of services



https://artifacthub.io/

Large catalogue of services



https://artifacthub.io/

SOFTWARE STACK, CONTAINER DEVELOPMENT OPTIONS

- Several development options if you want to go beyond existing containers!
- Nautilus: Gitlab – we have instructions for:
  - Building in Gitlab

  https://docs.nrp.ai/userdocs/development/gitlab/

  How to use your private repository

  https://docs.nrp.ai/userdocs/development/private-repos/
  - Integrate Gitlab and Kubernetes vi CI/CD jobs

  https://docs.nrp.ai/userdocs/development/k8s-integration/
- Nautilus: Coder open-source platform for creating and managing developer workspaces - used for FPGA code development; advanced code development on k8s cluster instead of local resources; develop with profiling level access; templates available for common dev setups
  - https://coder.com
- You can always spin up VMs in cloud resources (commercial or NSF funded options like Jetstream2) to do development work and push to hubs.

Hands-on session

https://github.com/mahidhar/6nrp_k8s_tutorial

4_applications.md

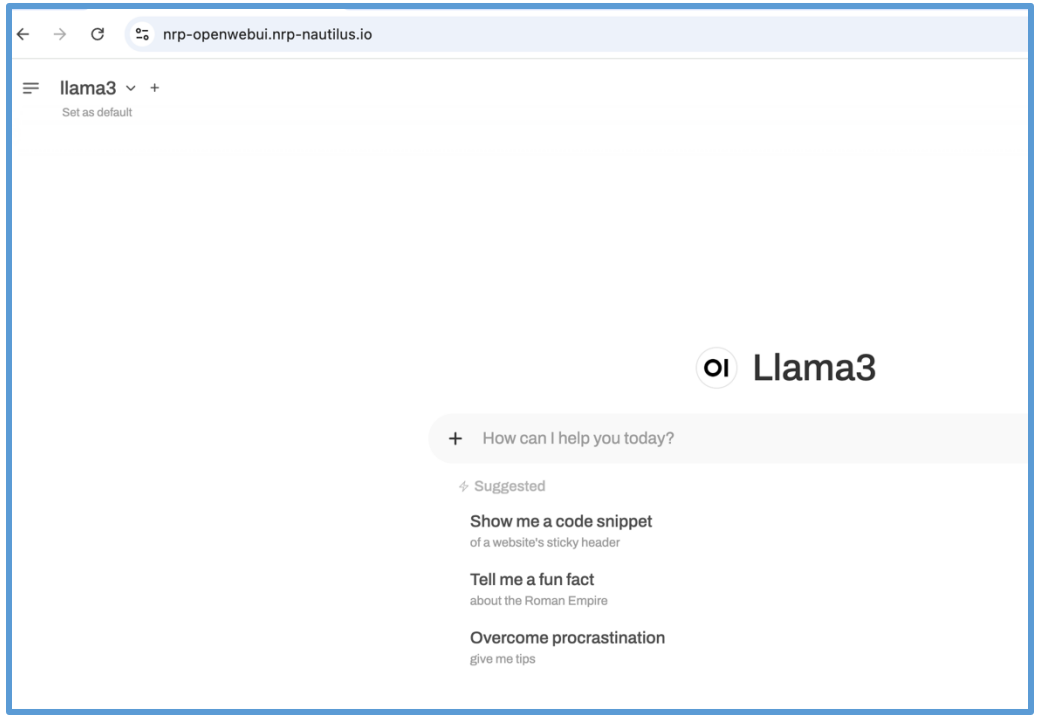
AI resources/services using Nautilus

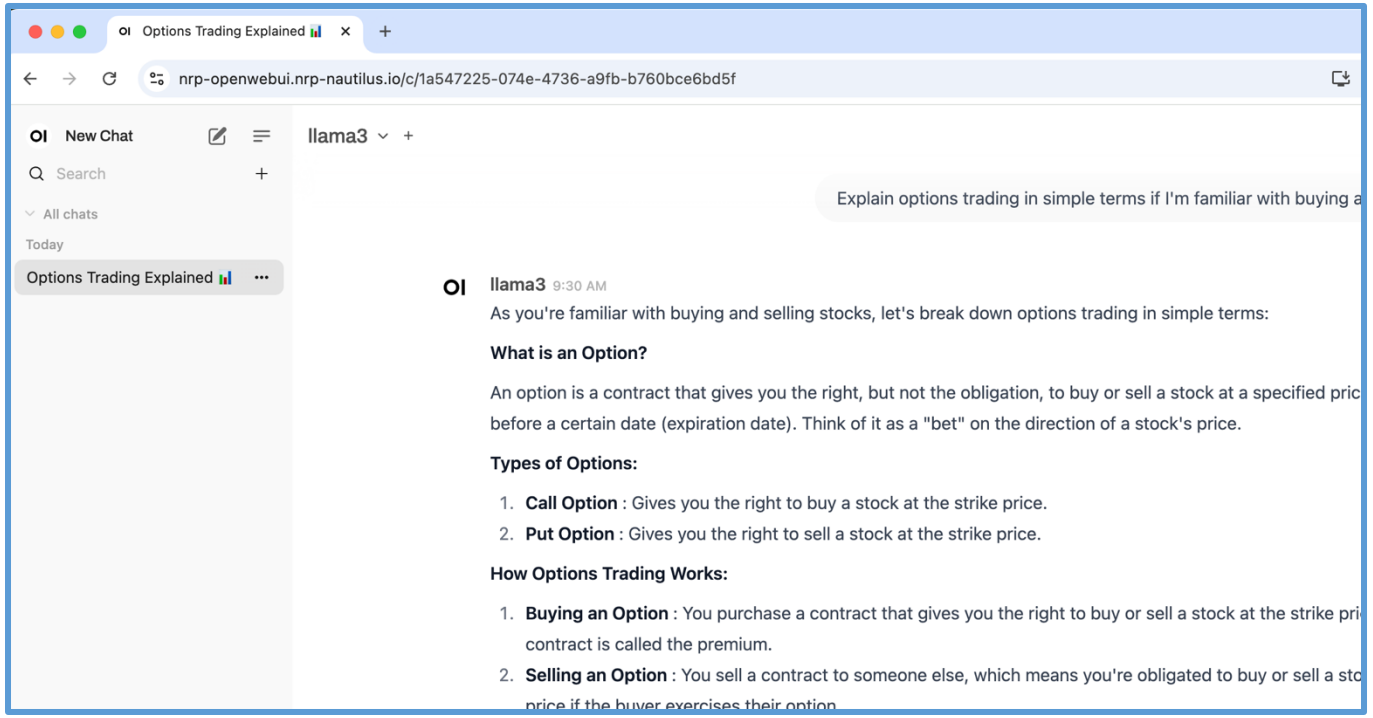
NRP Managed LLM Services

- Public managed LLM API
  - Uses the LiteLLM LLM proxy to provide access to the LLMs NRP staff is running on nautilus
  - Login to https://llm.nrp-nautilus.io/ui/ and request in Matrix channel to be added to a team
  - Once in a team, users can create tokens and access the models

- NRP staff also run several Chat UI instances
  - https://nextcloud.nrp-nautilus.io
  - https://nrp-openwebui.nrp-nautilus.io
  - https://librechat.nrp-nautilus.io

NRP Managed LLM Services





PNRP NAIRR Classroom: Resources

- Jupyter resources for classes that need AI/ML – can be teaching about AI/ML or using AI/ML in domain science
- Can use institutional authentication methods leveraging InCommon
- Professor and TA will have admin status to create namespaces for classes
- Support/Community interaction through Matrix channels – one for the class that includes students so Professor/TA can help them and second for Jupyter admins that includes admins from other classes
- Direct ticketing system available for any issue needing longer follow ups
- GPU nodes each with 8 A10 GPUs, 512GB of RAM, 2 AMD EPYC 7502 CPUs, and 8TB of NVMe.
- Depending needs, other GPU resources can be provisioned via the NAIRR pilot and made available through this platform
- https://nairrpilot.org/opportunities/education-call

The end

Acknowledgements

This work was partially funded by
US National Science Foundation (NSF) awards
OAC-2112167, OAC-1826967, OAC-1541349,
OAC-2030508 and CNS-1730158.