



# 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](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

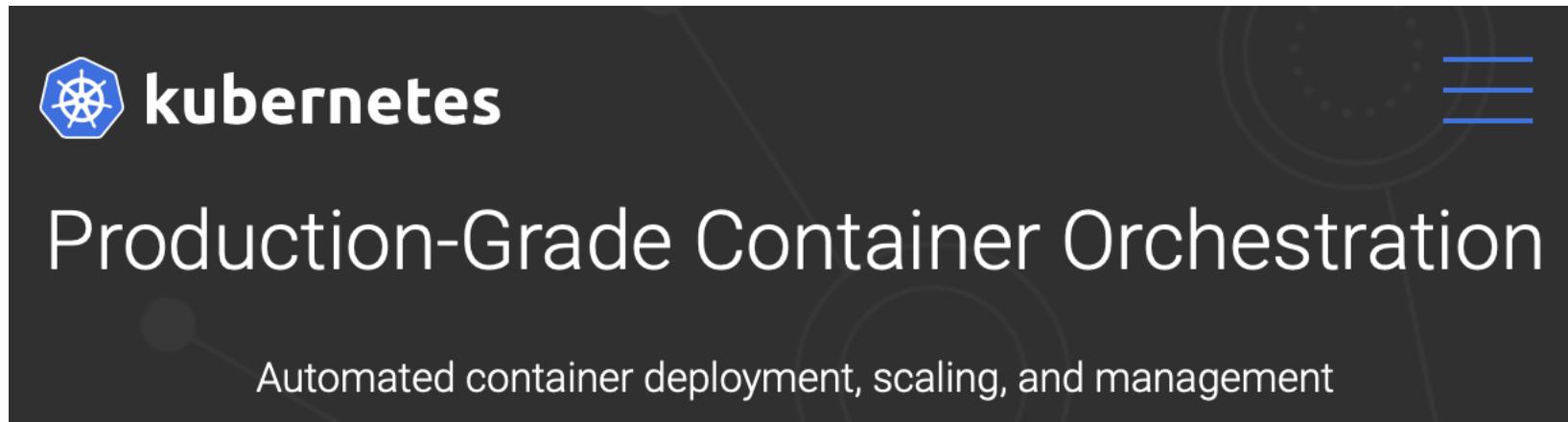
- Also more efficient than VMs

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

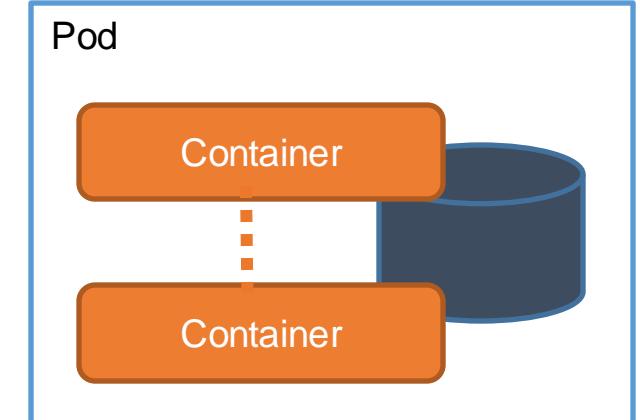
A Pod is a set of containers

- 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)

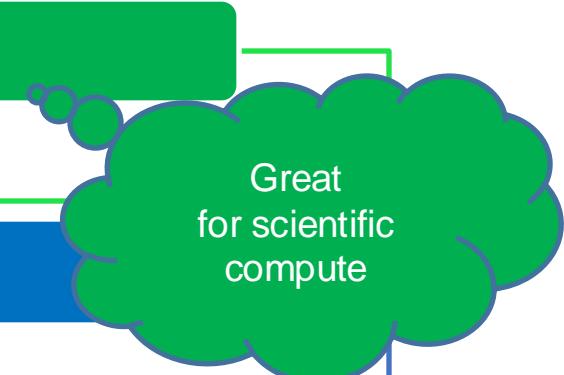
- Can retry the job up to N times

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



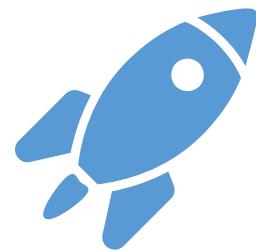
Great  
for scientific  
compute

# 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

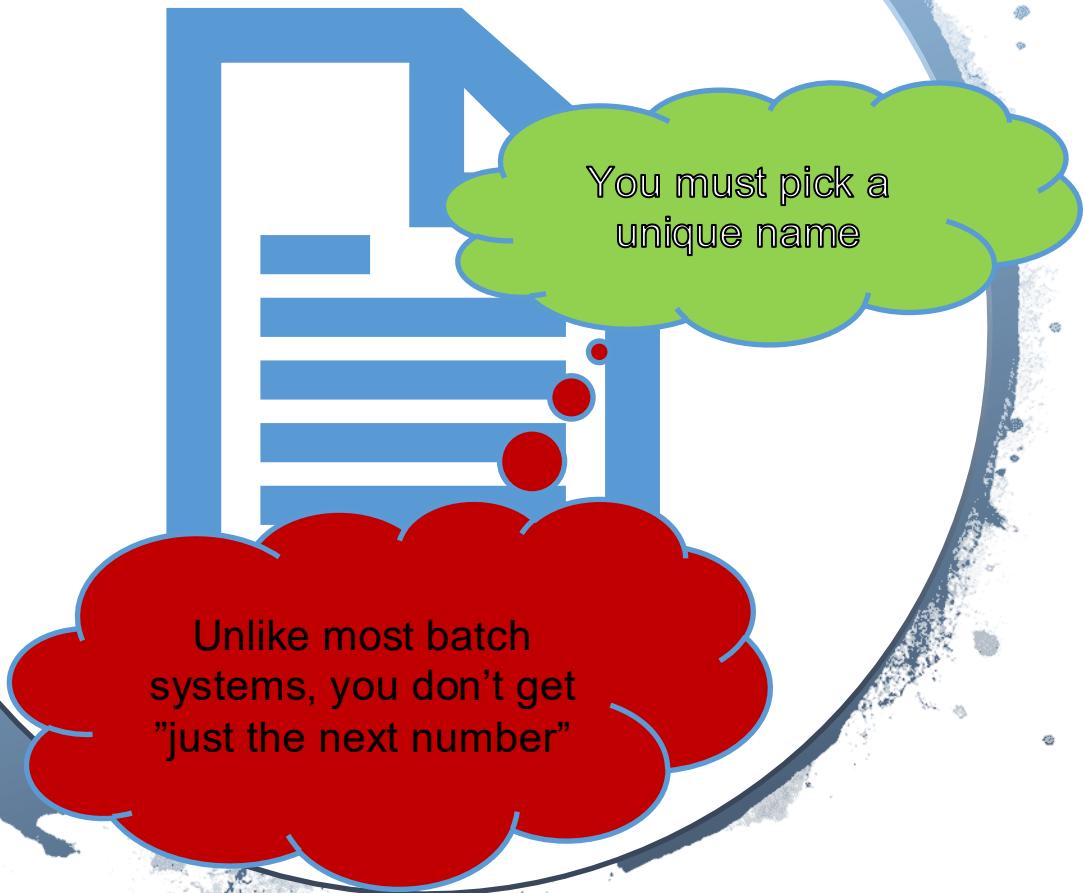
<https://en.wikipedia.org/wiki/YAML>



# 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



```
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

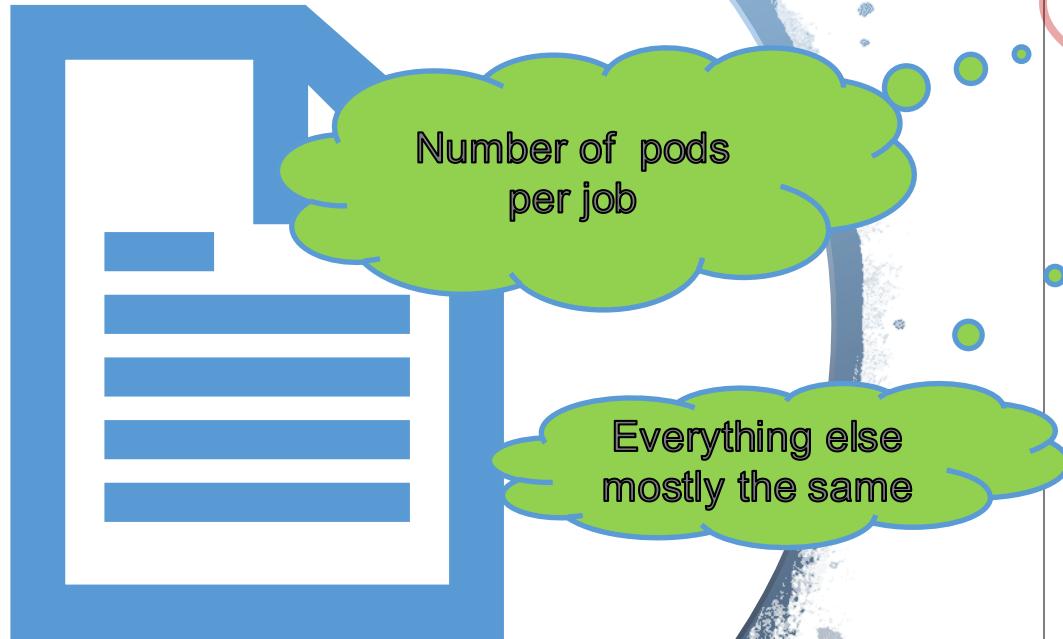
- Useful both for true interactive pods as well as for debugging  
`kubectl exec`
- By default just runs a command, but can be made interactive with  
`-it -- /bin/bash`

# 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



```
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"]
```

A callout bubble in the top right corner contains the text "Tell K8S it is a job, not just a pod". A callout bubble at the bottom right contains the text "Index within the job". Several parts of the YAML code are highlighted with colored ovals: "apiVersion: batch/v1" (black), "spec:" (red), "ttlSecondsAfterFinished: 1800" (red), "restartPolicy: OnFailure" (blue), "resources:" (blue), "limits:" (black), "requests:" (black), "command:" (black), and the final command line (black).

# 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)

Monitoring jobs is optional, but you should still know how to do it.

# Pod Networking

Each container gets 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](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)



```
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"
```

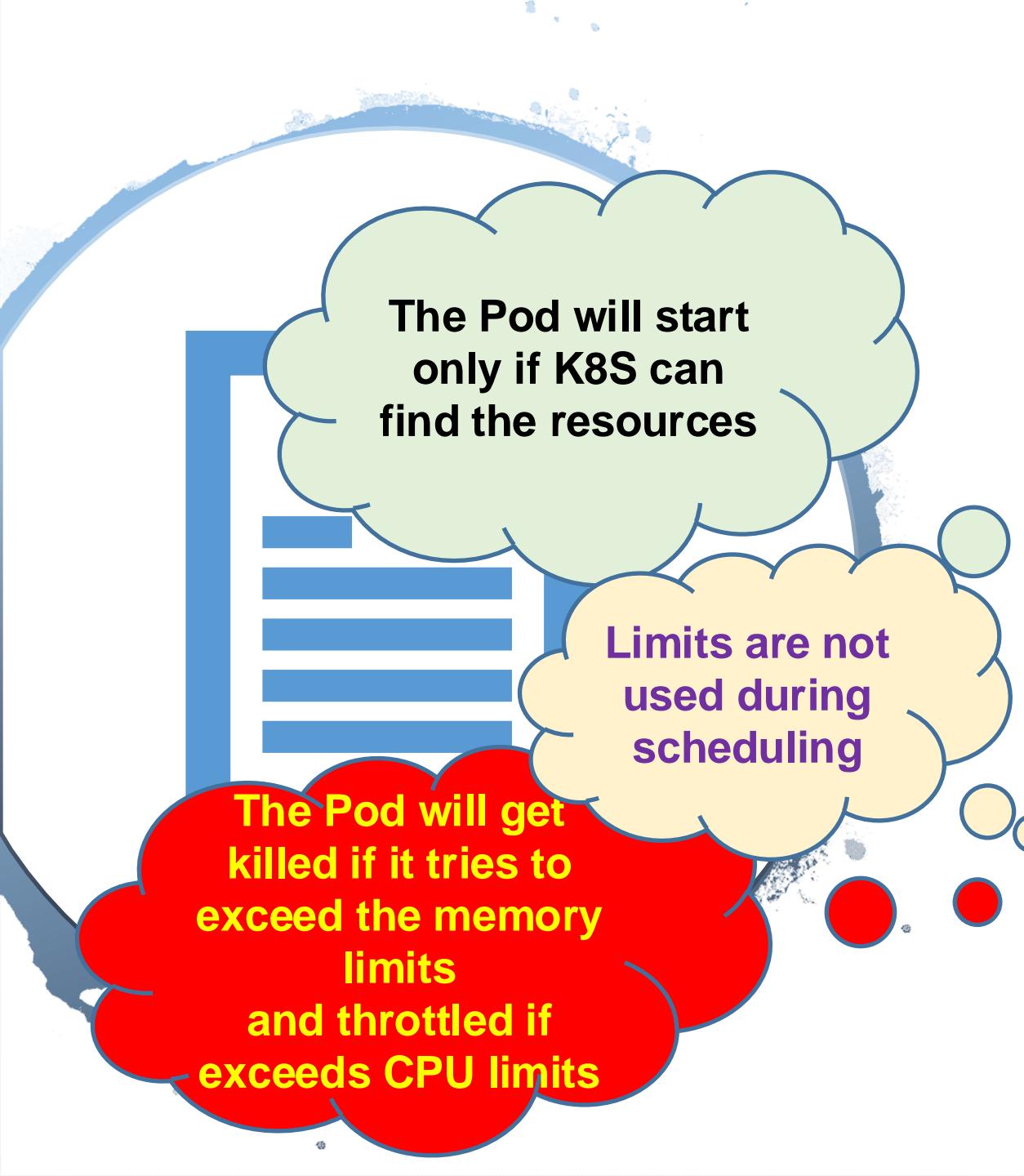
# An example of preferences

(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"
```



The Pod will start  
only if K8S can  
find the resources

Limits are not  
used during  
scheduling

The Pod will get  
killed if it tries to  
exceed the memory  
limits  
and throttled if  
exceeds CPU limits

# 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

### GPUs

### Cores

### RAM, GB

### GPU type

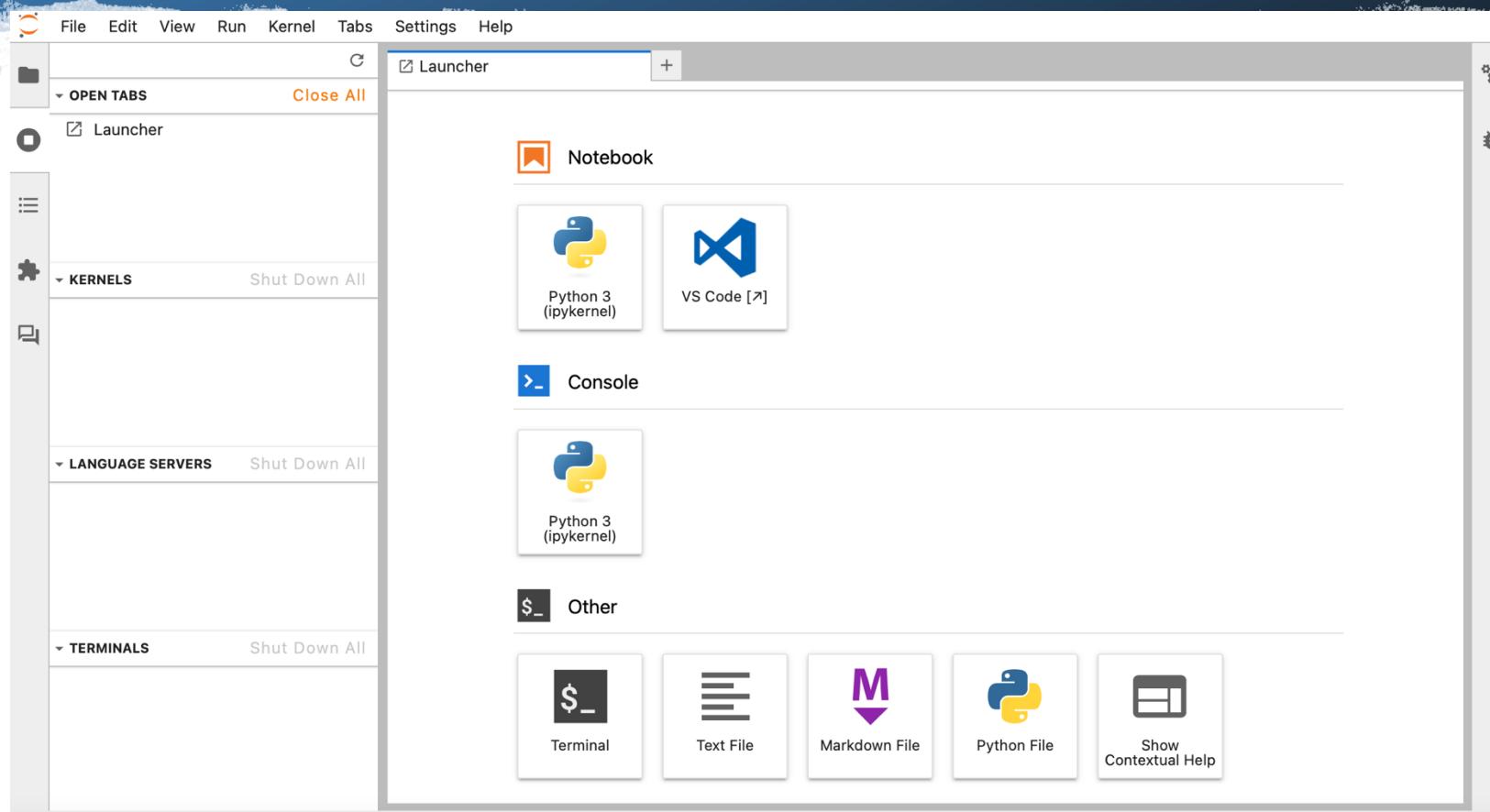
[/dev/shm for pytorch](#)



### Image

- Stack Minimal
- Stack Scipy
- Stack R
- Stack Julia
- Stack Tensorflow
- Stack Datasience (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

The screenshot shows a GitLab repository page for 'Jupyterlab West'. The repository has 521 Commits, 3 Branches, 0 Tags, 3 MiB Project Storage, and 1 Environment. A recent commit titled 'add cephfs mounts for anping/h1hui@ucsd.edu' by Sam Albin was made 6 days ago. The repository interface includes tabs for History, Find file, Edit, and Code (which is currently selected). Below the code tab, there are buttons for README, Kubernetes, CI/CD configuration, Add LICENSE, Add CHANGELOG, Add CONTRIBUTING, Add Wiki, and Configure Integrations. A table at the bottom lists files with their names, last commits, and last update times.

Name	Last commit	Last update
.gitlab-ci.yml	oops	1 month ago
README.md	Update README.md	8 months ago

# JupyterHub Deployment

- Step by step guide:
- <https://docs.nationalresearchplatform.org/userdocs/jupyter/jupyterhub/>

# INTERACTIVE DEVELOPMENT

## *CODER FPGA development example from PNRP Staff*

The screenshot shows a GitLab repository page for the project 'coder-fpga-smartnic-docker'. The repository has 120 commits, 6 branches, and 0 tags. A recent merge commit is highlighted: 'Merge branch 'revert-e5faeb4b' into 'main'' by Mohammad Firas Sada, authored 3 months ago. The page includes navigation buttons for 'main', 'fpga-smartnic-p4', and '+', and links for 'History', 'Find file', 'Edit', and 'Code'. It also shows a 'README' file and an 'Auto DevOps enabled' status. A table lists files with their last commit details:

Name	Last commit	Last update
files	Update 49 files	7 months ago
.gitlab-ci.yml	revert	1 year ago
Dockerfile	Revert "Merge branch 'xilinx-ubuntu' into ..."	3 months ago

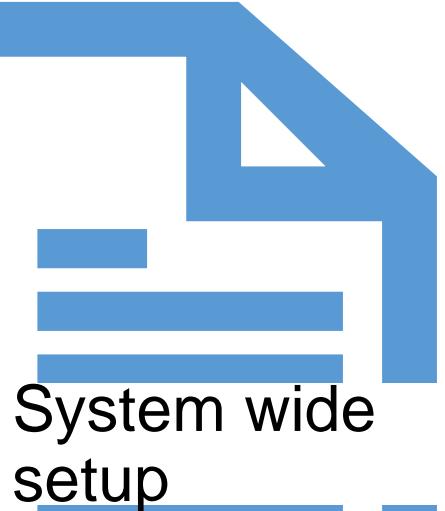
# Coder Templates Examples

The screenshot shows a web browser window with the URL `coder.nrp-nautilus.io` in the address bar. The page title is "Templates". The main content area displays a table of workspace templates with the following columns: Name, Used by, Build time, and Last updated. Each template entry includes a "Create Workspace" button.

Name	Used by	Build time	Last updated	
<b>U55C FPGA Vitis Workflow</b> U55C FPGA Development with Vitis and Vivado	10 developers	89s	8 days ago	<a href="#">→ Create Workspace</a>
<b>ESnet FPGA SmartNICs</b> ESnet FPGA SmartNIC Development	7 developers	74s	4 days ago	<a href="#">→ Create Workspace</a>
<b>containerlab-gNMIC</b>	4 developers	51s	8 months ago	<a href="#">→ Create Workspace</a>
<b>Cuda/PyTorch/TensorFlow</b>	4 developers	145s	2 months ago	<a href="#">→ Create Workspace</a>
<b>Docker development</b>	4 developers	67s	8 months ago	<a href="#">→ Create Workspace</a>

# Additional Scheduling Topics

# An example of priority



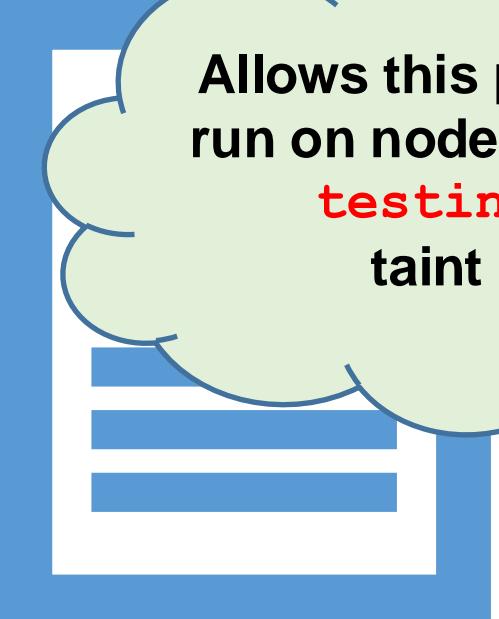
System wide  
setup

```
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



Allows this pod to  
run on node with a  
**testing**  
**taint**

```
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](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

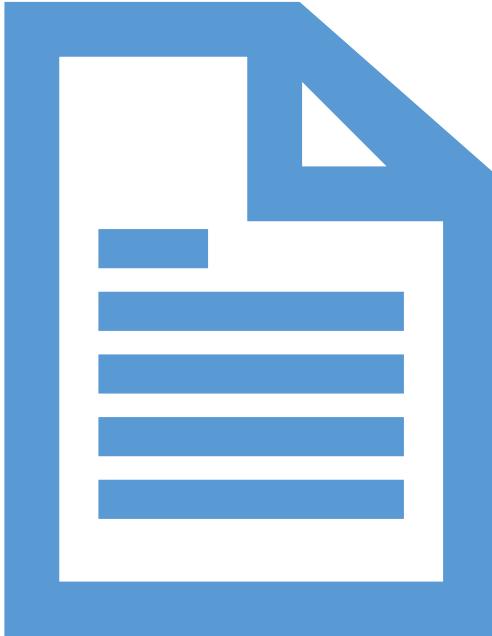
- Any directory path will work
- Whatever works for you

# 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 your favorite (non-K8S) tool (e.g. S3, Globus, scp)

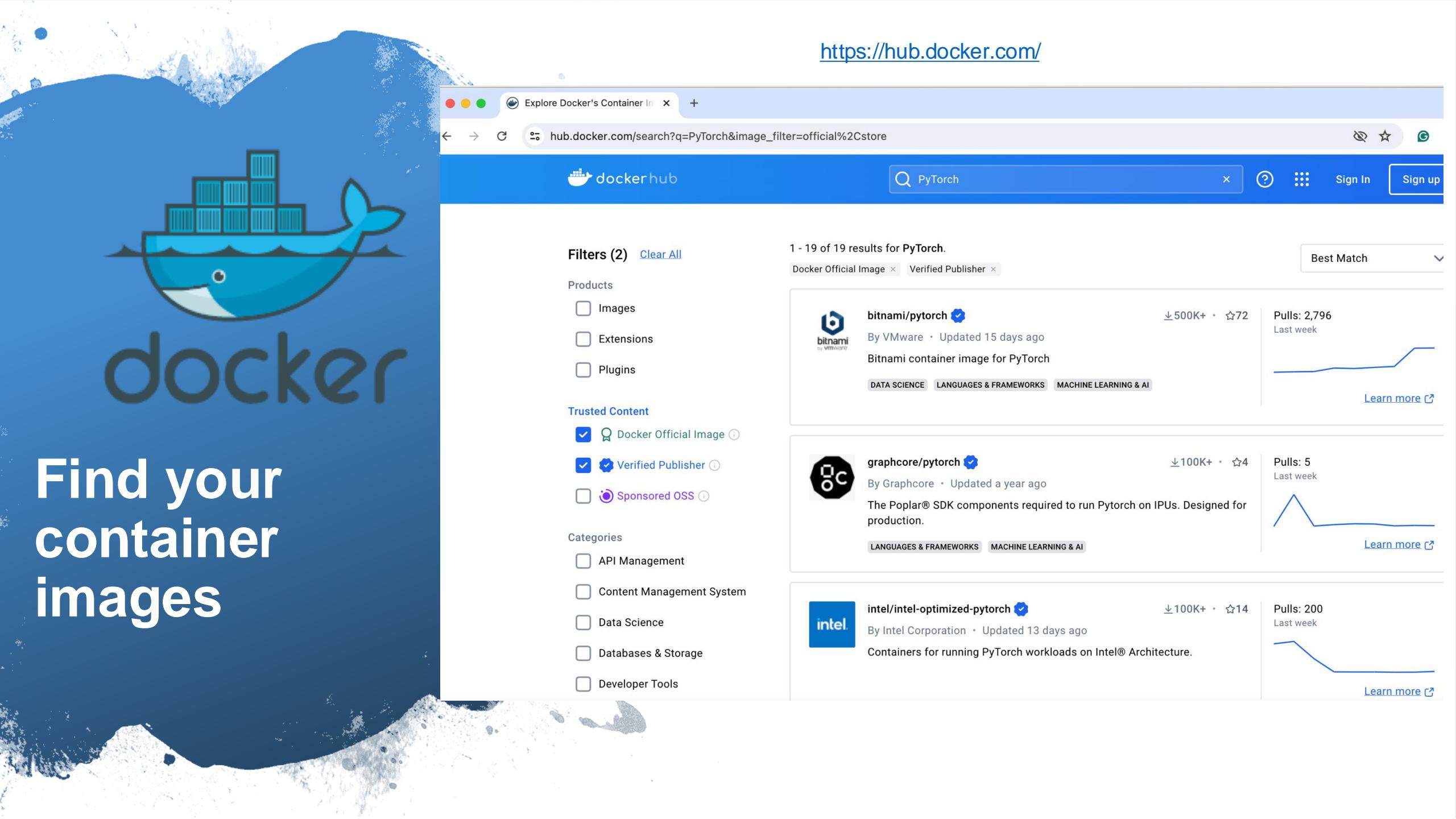
# Hands-on session

[https://github.com/mahidhar/6nrp\\_k8s tutorial](https://github.com/mahidhar/6nrp_k8s_tutorial)

[3 storage.md](#)

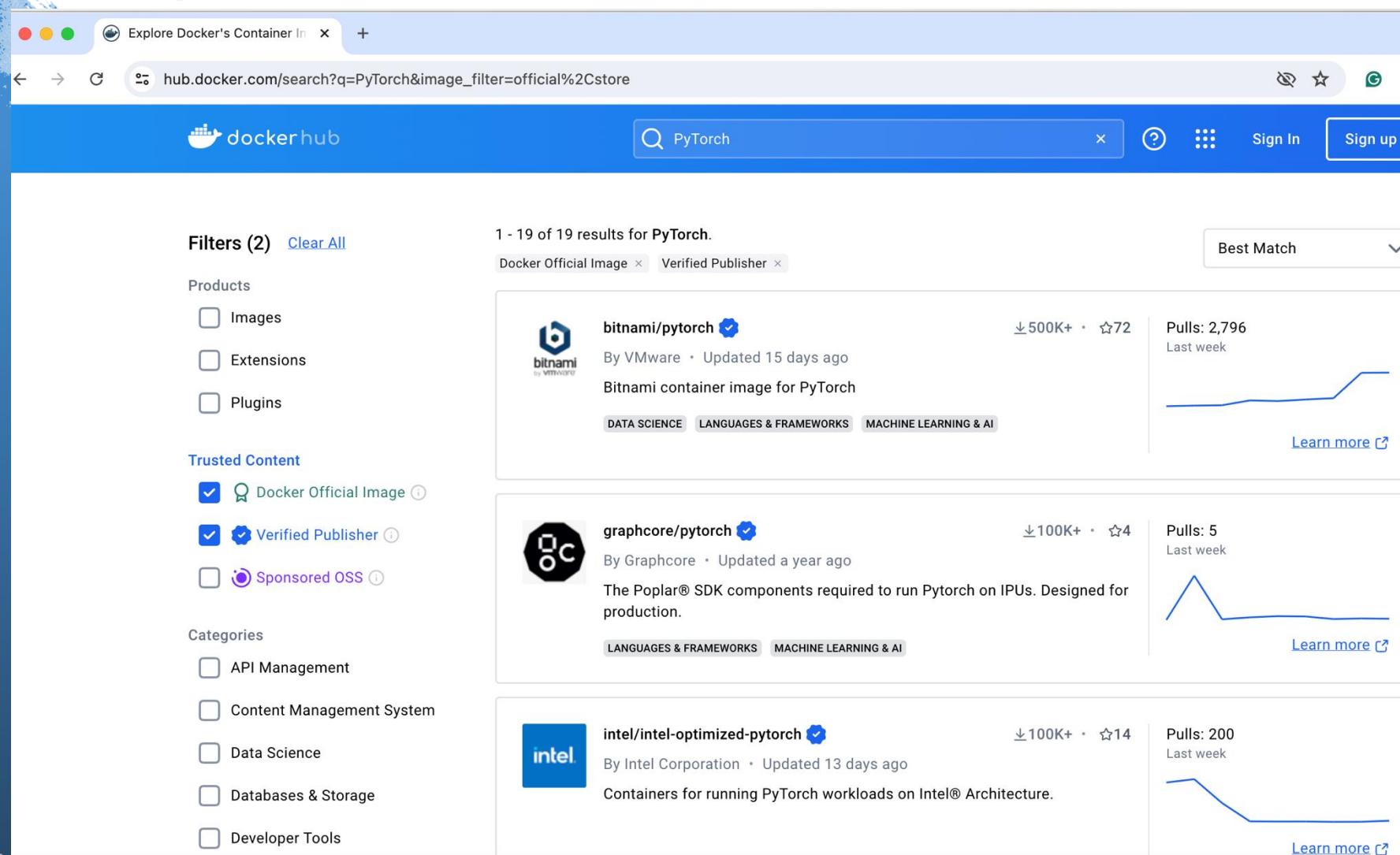
# Overview of Tutorial

- 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



# Find your container images

<https://hub.docker.com/>



The screenshot shows the Docker Hub search interface with the query "PyTorch". The results page displays three official PyTorch images from different publishers:

- bitnami/pytorch** (Verified Publisher) - Last updated 15 days ago. Description: Bitnami container image for PyTorch. Categories: DATA SCIENCE, LANGUAGES & FRAMEWORKS, MACHINE LEARNING & AI. Pulls: 2,796 last week. Graph shows a steady increase.
- graphcore/pytorch** (Verified Publisher) - Last updated a year ago. Description: The Poplar® SDK components required to run Pytorch on IPUs. Designed for production. Categories: LANGUAGES & FRAMEWORKS, MACHINE LEARNING & AI. Pulls: 5 last week. Graph shows a peak followed by a decline.
- intel/intel-optimized-pytorch** (Verified Publisher) - Last updated 13 days ago. Description: Containers for running PyTorch workloads on Intel® Architecture. Categories: LANGUAGES & FRAMEWORKS, MACHINE LEARNING & AI. Pulls: 200 last week. Graph shows a peak followed by a decline.

Filters applied: Docker Official Image, Verified Publisher.



# Find your container images

<https://hub.docker.com/>

The screenshot shows the Docker Hub homepage with a purple banner at the top stating "Wasm is a fast, light alternative to Linux containers – try it out today in the Docker+Wasm Technical Preview". Below the banner, the Docker Hub logo is visible, along with a search bar containing "tensorflow" and buttons for "Sign In" and "Register". A large green cloud-shaped callout box overlaps the center of the page, containing the text "Your community may have a standard image. Best to stick with it.". To the right of the callout, a statistic "Pulls 50M+" is displayed. At the bottom of the page, the TensorFlow repository details are shown: TAG latest, DIGEST 7f9f23ce2473, OS/ARCH linux/amd64, and COMPRESSED SIZE 429.94 MB. There is also a "docker pull tensorflow/tenso..." command and a download icon.

Wasm is a fast, light alternative to Linux containers – try it out today in the [Docker+Wasm Technical Preview](#)

Sign In Register

Pulls 50M+

Sort by

TAG

[latest](#)

Last pushed 2 months ago by [tensorflowpackages](#)

DIGEST

[7f9f23ce2473](#)

OS/ARCH

linux/amd64

COMPRESSED SIZE

429.94 MB

docker pull tensorflow/tenso...

# Community maintained images



<https://hub.docker.com/>

A screenshot of a web browser displaying the Docker Hub interface. The URL in the address bar is [hub.docker.com/u/bioconductor](https://hub.docker.com/u/bioconductor). The search bar at the top right contains the text `bioconductor/bioconductor_docker`. The main content area shows the profile for the "Bioconductor" organization, which is a Community Organization from Boston, MA, USA, joined on November 19, 2014. It is a Sponsored OSS project. Below this, the "Repositories" section is shown, with a message indicating 1 to 25 of 49 repositories. Two specific repositories are listed: `bioconductor/yes-jupyter` and `bioconductor/rstudio_yescds`, both updated 9 hours ago by Bioconductor. Each repository card includes a star count (531 and 553 respectively), pull requests (11 and 5), and a date range (Dec 3 to Dec 9 and Oct 29 to Nov 4). There are also "Learn more" links for each repository.

**Bioconductor** Sponsored OSS

Community Organization Bioconductor Boston, MA, USA <http://bioconductor.org/> Joined November 19, 2014

**Repositories**

Displaying 1 to 25 of 49 repositories

**bioconductor/yes-jupyter** By Bioconductor • Updated 9 hours ago ↘ 531 • ⭐ 0 Pulls: 11 Dec 3 to Dec 9 Learn more

**bioconductor/rstudio\_yescds** By Bioconductor • Updated 9 hours ago ↘ 553 • ⭐ 0 Pulls: 5 Oct 29 to Nov 4 Learn more

# Vendor optimized and signed images

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

The screenshot shows the NVIDIA NGC Catalog interface. On the left is a sidebar with links: Explore Catalog, Collections, Containers, Helm Charts, Models, and Resources. The main content area is titled "Containers > NVIDIA HPC SDK" and "NVIDIA HPC SDK". It features the NVIDIA HPC logo. Below it, there's a "Description" section stating: "The NVIDIA HPC SDK is a comprehensive suite of compilers, libraries and tools essential to maximizing developer productivity and the performance and portability of HPC applications." There are also sections for "Publisher" (NVIDIA), "Latest Tag" (24.5-devel-cuda\_multi-ubuntu22.04), "Modified" (June 30, 2024), "Compressed Size" (10.41 GB), and "Multinode Support". The top navigation bar shows the URL "catalog.ngc.nvidia.com/orgs/nvidia/containers/nvhpc/tags". The right side of the screen displays a table of container tags:

Tag	Status	Image URI
24.5-devel-cuda_multi-ubuntu22.04	Signed	<a href="#">nvcr.io/nvidia/nvhpc:24.5-devel-cuda_multi-ubuntu22.04</a>
24.5-runtime-cuda12.4-ubuntu22.04	Signed	<a href="#">nvcr.io/nvidia/nvhpc:24.5-runtime-cuda12.4-ubuntu22.04</a>
24.5-devel-cuda_multi-centos7	Signed	<a href="#">nvcr.io/nvidia/nvhpc:24.5-devel-cuda_multi-centos7</a>

Each row includes a timestamp (06/07/2024), size (e.g., 10.41 GB, 2.42 GB), and number of architectures (2 or 1).



# Find your container images

<https://hub.docker.com/>

The screenshot shows the Docker Hub website at [hub.docker.com](https://hub.docker.com/). At the top, there's a purple banner with the text "Wasm is a fast, light alternative to Linux containers – try it out today in the [Docker+Wasm Technical Preview](#)". The main navigation bar includes "Explore", "Pricing", "Sign In", and a "Register" button. On the right, there's a "Pulls 50M+" badge. The central part of the page displays the "Tags" tab for the "tensorflow/tensorflow" image. It shows a "latest" tag pushed 2 months ago with a digest hash of [7f9f23ce2473](#). A yellow cloud-shaped callout bubble points to this section with the text "Can create derivative image with customizations". Above this, a green cloud-shaped callout bubble points to the main content area with the text "Your community may have a standard image. Best to stick with it."

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

TensorFlow (http://www.tensorflow.org)

Overview Tags

Sort by Newest

TAG latest

Last pushed 2 months ago

DIGEST [7f9f23ce2473](#)

tensorflow/tenso...

PULLS 50M+

COMRESSED SIZE 429.94 MB

# 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”

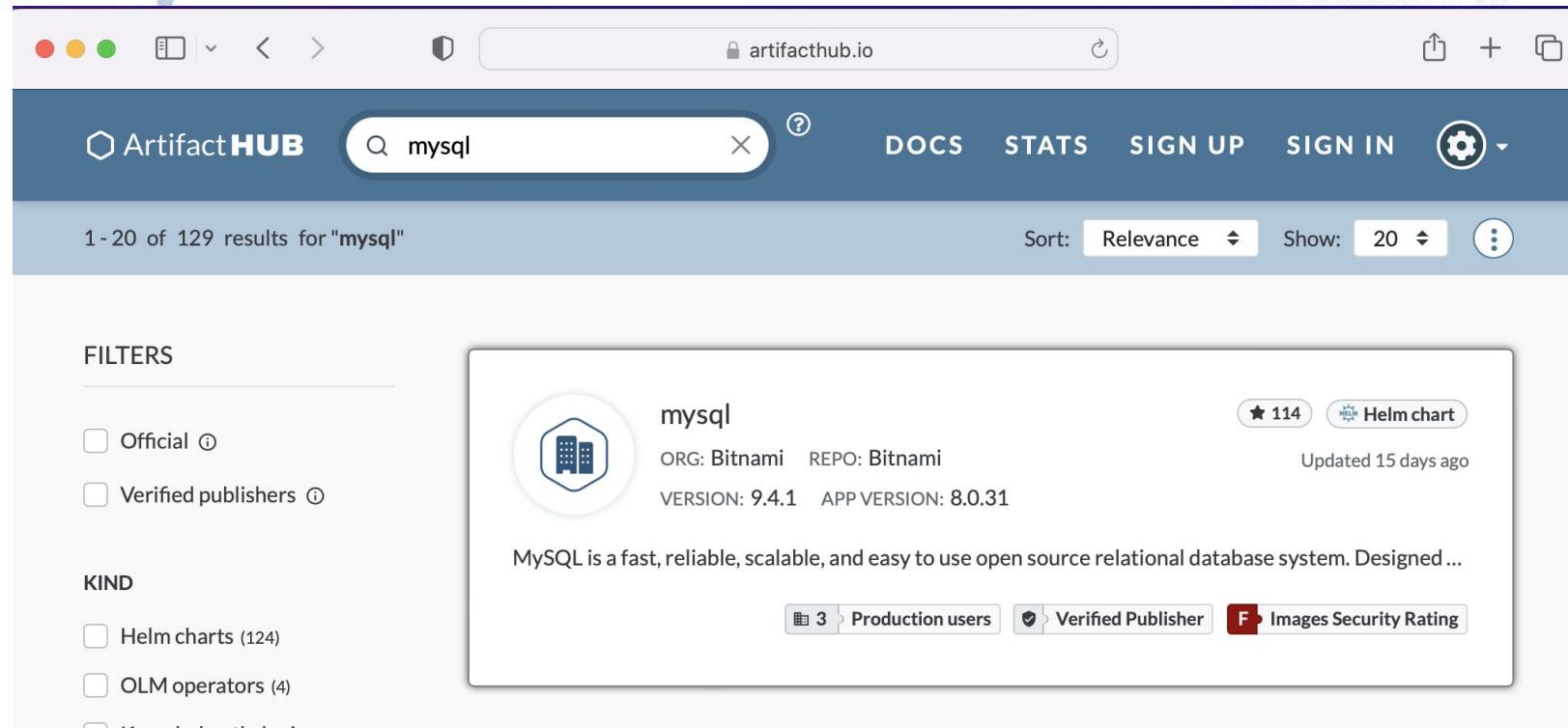
<https://helm.sh>



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



A screenshot of a web browser displaying the Artifact Hub website (<https://artifacthub.io/>). The search bar at the top contains the query "mysql". Below the search bar, the page shows "1-20 of 129 results for 'mysql'" and includes sorting and pagination controls. On the left, there are "FILTERS" and "KIND" sections with checkboxes for "Official", "Verified publishers", "Helm charts", and "OLM operators". The main content area features a card for the "mysql" service by Bitnami, version 9.4.1, with an app version of 8.0.31. The card includes a star rating of 114, a "Helm chart" link, and a note that it was updated 15 days ago. A detailed description of MySQL is provided, along with production users, verified publisher, and security rating information.

https://artifacthub.io/

Artifact HUB

mysql

DOCS STATS SIGN UP SIGN IN

1-20 of 129 results for "mysql"

Sort: Relevance Show: 20

FILTERS

Official ⓘ

Verified publishers ⓘ

KIND

Helm charts (124)

OLM operators (4)

mysql

★ 114 HELM Helm chart

ORG: Bitnami REPO: Bitnami

Updated 15 days ago

VERSION: 9.4.1 APP VERSION: 8.0.31

MySQL is a fast, reliable, scalable, and easy to use open source relational database system. Designed ...

3 Production users Verified Publisher F Images Security Rating

# Large catalogue of services

The screenshot shows the ArtifactHub website interface. At the top, there is a navigation bar with the logo "ArtifactHUB", a search bar containing "Search packages", and links for "DOCS", "STATS", "SIGN UP", "SIGN IN", and a gear icon. Below the navigation bar, the main content area displays a package card for "text-generation-inference". The card includes a container icon, the package name "text-generation-inference", a "Helm chart" badge, and two "Substratus" badges. To the right of the package name are icons for "Star" (0), notifications, settings, and more options. Below the card, a brief description states: "A Helm chart for deploying HuggingFace text-generation-inference. Text Generation Inference (TGI) is a toolkit for deploying and serving Large Language Models (LLMs). TGI enables high-performance text generation for the most popular open-source LLMs." At the bottom of the card, there are four small icons representing different deployment or management tools.

<https://artifacthub.io/>

## text-generation-inference Helm Chart

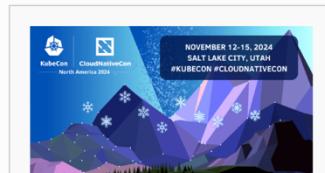
A Helm chart for deploying HuggingFace text-generation-inference. Text Generation Inference (TGI) is a toolkit for deploying and serving Large Language Models (LLMs). TGI enables high-performance text generation for the most popular open-source LLMs.

### Usage

Basic usage:

```
helm repo add substratusai https://substratusai.github.io/helm
helm install mistral-7b-instruct substratusai/text-generation-inference
# Note by default the resource limit is set to 1 GPU
helm install mistral-7b-instruct substratusai/text-generation-inference \
--set model=mistralai/Mistral-7B-Instruct-v0.1
```

- [INSTALL](#)
- [TEMPLATES](#)
- [DEFAULT VALUES](#)
- [CHANGELOG](#)



# 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 via 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](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

The screenshots demonstrate the NRP Managed LLM Services interface. The left screenshot shows the initial state of the interface, with a sidebar containing a 'Suggested' section and a main area asking 'How can I help you today?'. The right screenshot shows a chat session where a user asks 'Explain options trading in simple terms if I'm familiar with buying a stock'. The LLM responds with an explanation of what an option is, the types of options (Call and Put), and how options trading works.

How can I help you today?

Suggested

Show me a code snippet  
of a website's sticky header

Tell me a fun fact  
about the Roman Empire

Overcome procrastination  
give me tips

Explain options trading in simple terms if I'm familiar with buying a stock

As you're familiar with buying and selling stocks, let's break down options trading in simple terms:

**What is an Option?**

An option is a contract that gives you the right, but not the obligation, to buy or sell a stock at a specified price before a certain date (expiration date). Think of it as a "bet" on the direction of a stock's price.

**Types of Options:**

- Call Option**: Gives you the right to buy a stock at the strike price.
- Put Option**: Gives you the right to sell a stock at the strike price.

**How Options Trading Works:**

- Buying an Option**: You purchase a contract that gives you the right to buy or sell a stock at the strike price. This price is called the premium.
- Selling an Option**: You sell a contract to someone else, which means you're obligated to buy or sell a stock at the strike price if the buyer exercises their option.

# 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.