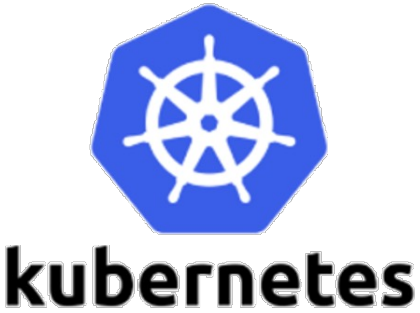


AI and Scientific Research Computing with Kubernetes

The basics

A tutorial at PEARC24

July 22, Providence, Rhode Island

Presented by Mahidhar Tatineni and Dmitry Mishin
University of California San Diego – San Diego Supercomputer Center

Ref: Tutorials at PEARC, SC, 5NRP by Igor Sfiligoi, Dmitry Mishin, and Mahidhar Tatineni

Repository for Today's talks and hands-on:

https://github.com/mahidhar/pearc24_k8s_tutorial

Agenda

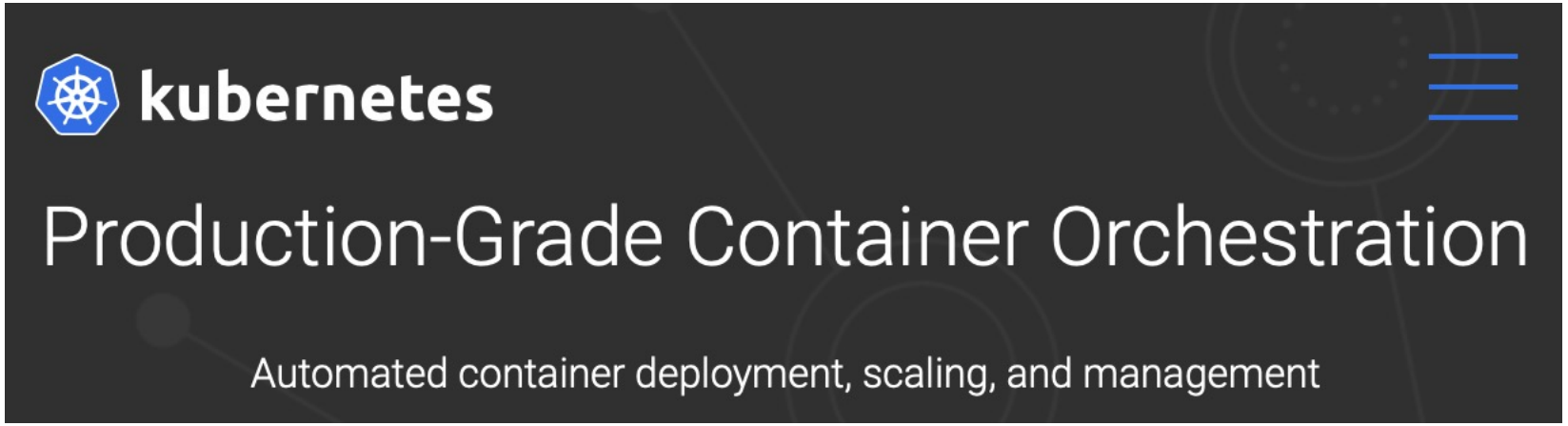
- Introduction and Welcome
- An overview of the Kubernetes architecture
  - Basic Kubernetes Hands On
- Kubernetes resource scheduling
  - Scheduling Hands On
- AI and science research applications with Kubernetes
- *Break*
- AI and scientific research applications examples with Hands On
  - AI training using PyTorch example
  - Text generation inference example
  - RAG example using Ollama
  - Helm based deployment of LLM as service (H2O)
  - LAMMPS (molecular dynamics application) example
- Storage
  - Storage hands on
- Monitoring your work
- Closeout

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

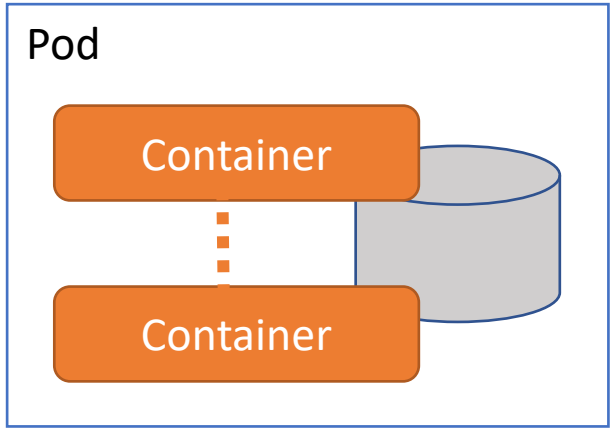
https://kubernetes.io/docs/concepts/workloads/pods/pod/

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



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/

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

Easy but pedantic

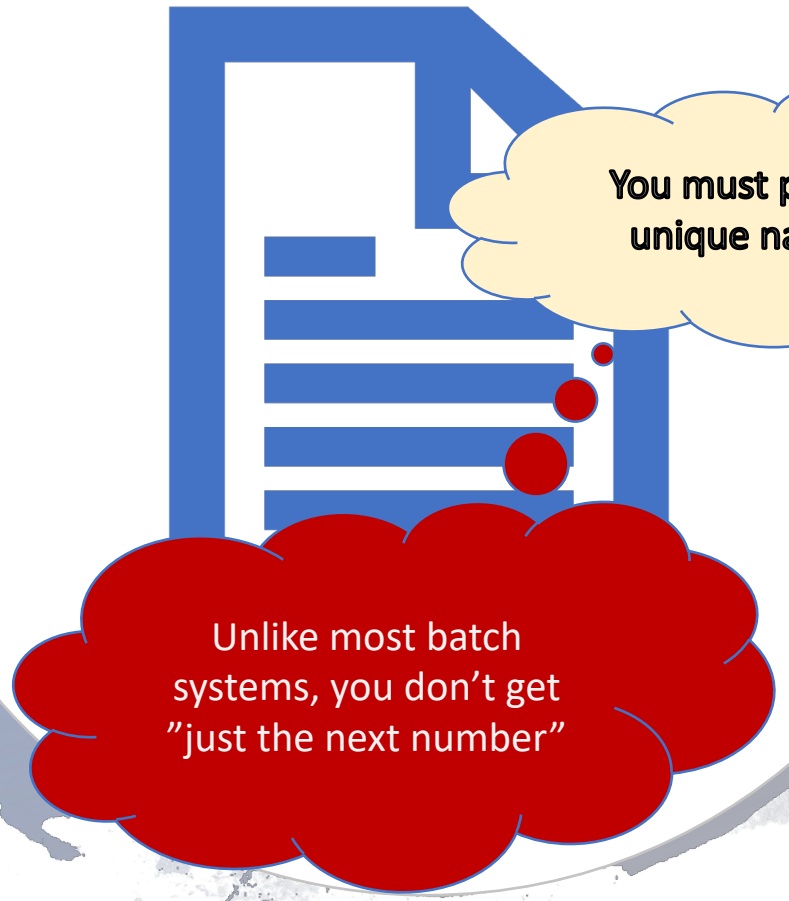
```
apiVersion: batch/v1
kind: Job
metadata:
  labels:
    k8s-app: pilot
  name: pilot-Oct22
spec:
  completions: 3
  parallelism: 3
  template:
    metadata:
      labels:
        k8s-app: pilot
    spec:
      containers:
      - env:
        - name: USE_SINGULARITY
          value: "no"
        image: sfiligoi/pilot:v1
        name: htcondor
        resources:
          limits:
            cpu: 2.5
            memory: 6Gi
          requests:
            cpu: 1.5
            memory: 4Gi
        volumeMounts:
        - mountPath: /data
          name: s1
      volumes:
      - name: s1
        persistentVolumeClaim:
          claimName: igor-durable
```

Just a simple example

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

Command to execute
(including any arguments)

```
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)

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

A word about networking

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

And now the hands-on session

Repository for Today's talks and hands-on:

https://github.com/mahidhar/pearc24_k8s_tutorial

Acknowledgents

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