AI and Scientific Research Computing with Kubernetes
Resource scheduling

A tutorial at PEARC24

July 22, Providence, Rhode Island

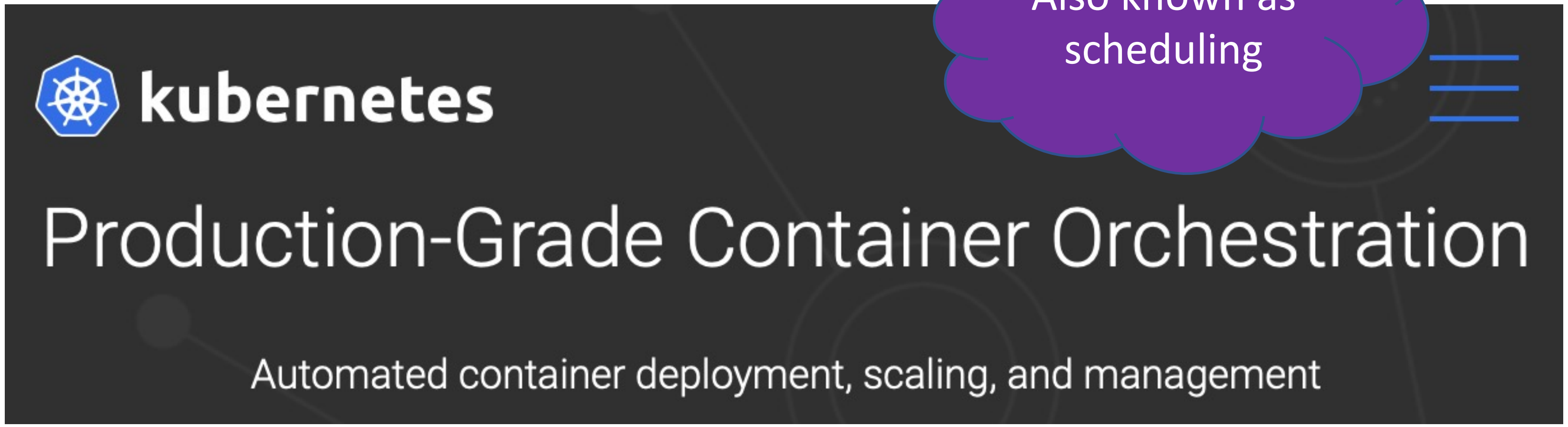
Presented by Mahidhar Tatineni and Dmitry Mishin
University of California San Diego – San Diego Supercomputer Center

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

Container Orchestration

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

Also known as scheduling



Attribution: https://kubernetes.io

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.)

- Nodes advertise what is available
- Pods specify what they require, may also limit itself to a subset of Nodes
- A Pod will start on a Node only if a match can be made

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/

Driving Kubernetes

The simple pod YAML

```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

```yaml
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”
```

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

Interacting with Kubernetes

Still using kubectl

- kubectl create -f <filename> - Create new object
- kubectl get <type> - Query existing objects
- kubectl logs <id> - Fetch stdout (result)
- kubectl delete -f <filename> - Delete existing object

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

Finding the resources

All nodes advertise their state

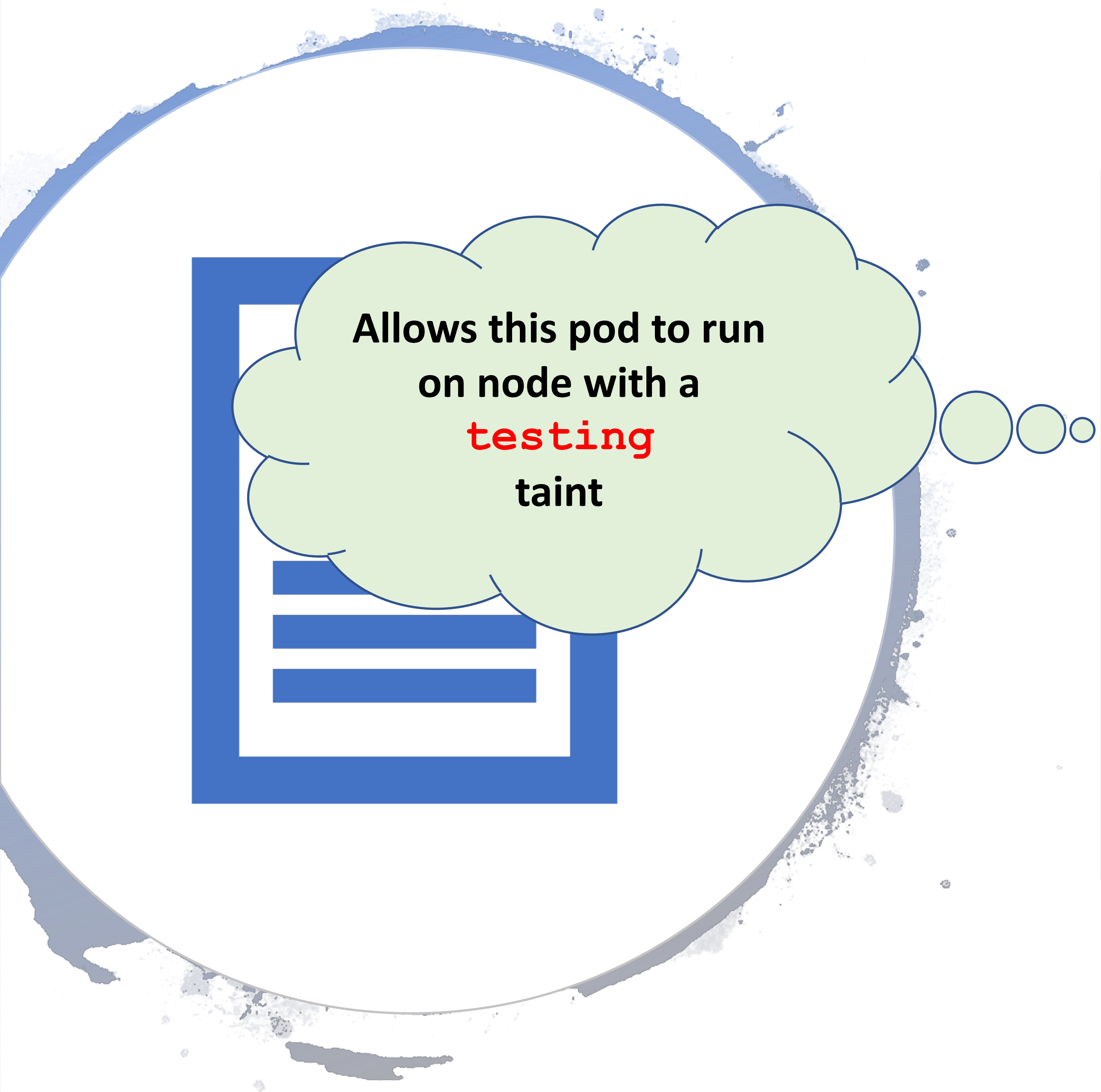
- Just like all pods do

Use kubectl to list them

- `kubectl get nodes`
- Use `-o wide` or `-o yaml` for detailed view

Advanced topics

An example of toleration




```yaml
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/

An example of priority

System wide setup

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

```yaml
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/

MPI jobs

- Kubernetes does not normally support MPI jobs
  - External projects help getting a cluster to support them
  - SDSC is using Kubeflow in Voyager: https://www.kubeflow.org/docs/components/training/mpi/

- Gang-scheduling is generally not available, either
  - An external scheduler plugins can be used: https://github.com/kubernetes-sigs/scheduler-plugins/blob/master/doc/install.md#install-release-v0257-and-use-coscheduling

Hands-on time

Acknowledgents

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