Introduction to Kubernetes orchestration engine

(and how HPC can be inspired by a new generation scheduler)

Yiannis Georgiou - CTO Ryax Technologies

ryax technologies

KubeCon 2018

ryax technologies

Kubernetes Origins and Intro

- Initially developed by Google, inspired by **Borg and Omega** both proprietary software used internally at Google.
- Google open-sourced Kubernetes in 2014.
- It is a software to **deploy and manage containerized applications** while providing the **best possible utilization** of the compute platform.
- It **abstracts away the underlying infrastructure** simplifying app development and hardware management.

Kubernetes Benefits

- Application **deployment** simplification.
- Hardware **system utilization** improvement.
- Application **automatic scaling.**
- Application **development** simplification
- Fault Tolerance, High Availability and **Self Healing**

Kubernetes Architecture

- Control Plane (master)
  - **API server** : point of entry to everything
  - **Scheduler** : assigns nodes to components
  - **Controller Manager** : cluster level functions
  - **Etcd** : reliable key-value store
- Worker nodes
  - **Kubelet** : manages containers, talks to API
  - **Container Runtime** : deploys containers
  - **Kube-proxy** : load balances node traffic

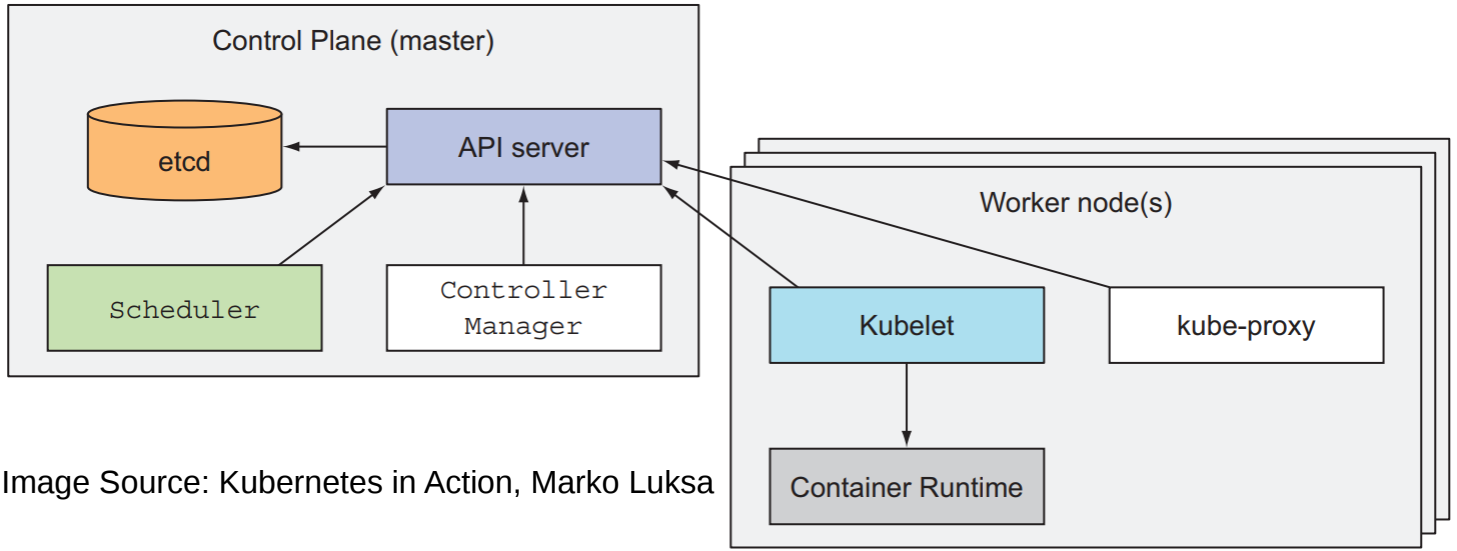


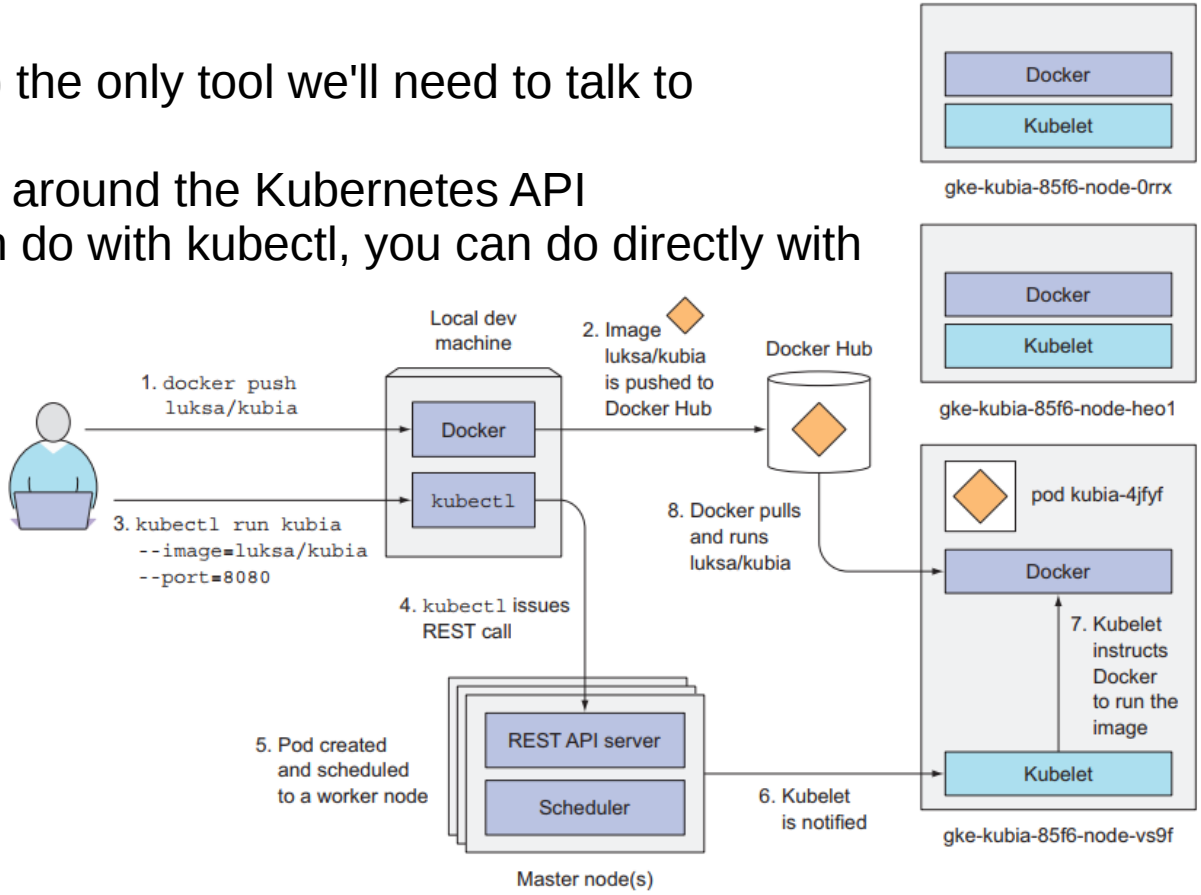
Image Source: Kubernetes in Action, Marko Luksa

Kubernetes objects

- **Pods**
  - Represent a deployment unit composed by one or more tightly coupled containers sharing resources.
  - Containers within a Pod can communicate with each other through localhost.
  - All pods reside in a single flat, shared, network-address space, no NAT gateways exist between them. Pods access each other on their unique IP address.
- **Controllers**
  - Create and manage multiple pods handling replication, rollout, self-healing.
- **Services**
  - Represent a single, constant point of entry to a group of pods providing the same service. Each service has an IP address and port that never change while the service exists.
- **Volumes**
  - Are directories accessible to the containers of a pod. Bound to pod lifecycle.
- **Namespaces**
  - Provide an abstraction that enable the usage of multiple virtual clusters backed by the same physical cluster.
- **Nodes**
  - They can be VMs or physical machines, they provide the necessary services to run pods and are managed by the master components.

Kubernetes CLI and example run

- **kubectl** is (almost) the only tool we'll need to talk to Kubernetes
- It is a rich **CLI tool** around the Kubernetes API
- Everything you can do with kubectl, you can do directly with the API



Kubernetes CLI

```
ubuntu@node1:~$ kubectl get nodes
NAME      STATUS    ROLES     AGE       VERSION
node1     Ready     master    6m        v1.10.2
node2     Ready     <none>    6m        v1.10.2
node3     Ready     <none>    6m        v1.10.2
node4     Ready     <none>    6m        v1.10.2

ubuntu@node1:~$ kubectl get nodes -o wide
NAME      STATUS    ROLES     AGE       VERSION   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
node1     Ready     master    3m        v1.10.2   <none>        Ubuntu 16.04.4 LTS   4.4.0-1057-aws   docker://18.3.1
node2     Ready     <none>    3m        v1.10.2   <none>        Ubuntu 16.04.4 LTS   4.4.0-1057-aws   docker://18.3.1
node3     Ready     <none>    3m        v1.10.2   <none>        Ubuntu 16.04.4 LTS   4.4.0-1057-aws   docker://18.3.1
node4     Ready     <none>    3m        v1.10.2   <none>        Ubuntu 16.04.4 LTS   4.4.0-1057-aws   docker://18.3.1

ubuntu@node1:~$ kubectl get nodes -o json | jq ".items[] | {name:.metadata.name} + .status.capacity"
{
  "name": "node1",
  "cpu": "2",
  "ephemeral-storage": "8065444Ki",
  "hugepages-2Mi": "0",
  "memory": "4045076Ki",
  "pods": "110"
}..

ubuntu@node1:~$ kubectl describe nodes node1
...
```

ryax technologies

Kubernetes CLI

```
ubuntu@node1:~$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    5m
kube-public   Active    5m
kube-system   Active    5m
ubuntu@node1:~$ kubectl get pods
No resources found.
ubuntu@node1:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                              READY     STATUS    RESTARTS   AGE
kube-system   etcd-node1                        1/1       Running   0          5m
kube-system   kube-apiserver-node1              1/1       Running   0          5m
kube-system   kube-controller-manager-node1     1/1       Running   0          5m
kube-system   kube-dns-86f4d74b45-lw27n         3/3       Running   0          5m
kube-system   kube-proxy-5vbpz                  1/1       Running   0          5m
kube-system   kube-proxy-bjz6c                  1/1       Running   0          5m
kube-system   kube-proxy-p42pl                  1/1       Running   0          5m
kube-system   kube-proxy-q89h9                  1/1       Running   0          5m
kube-system   kube-scheduler-node1              1/1       Running   0          5m
kube-system   weave-net-b7nd2                   2/2       Running   1          5m
kube-system   weave-net-cjpnx                   2/2       Running   0          5m
kube-system   weave-net-d7mjz                   2/2       Running   0          5m
kube-system   weave-net-ndjx2                   2/2       Running   1          5m
```

Kubernetes Architecture deeper dive

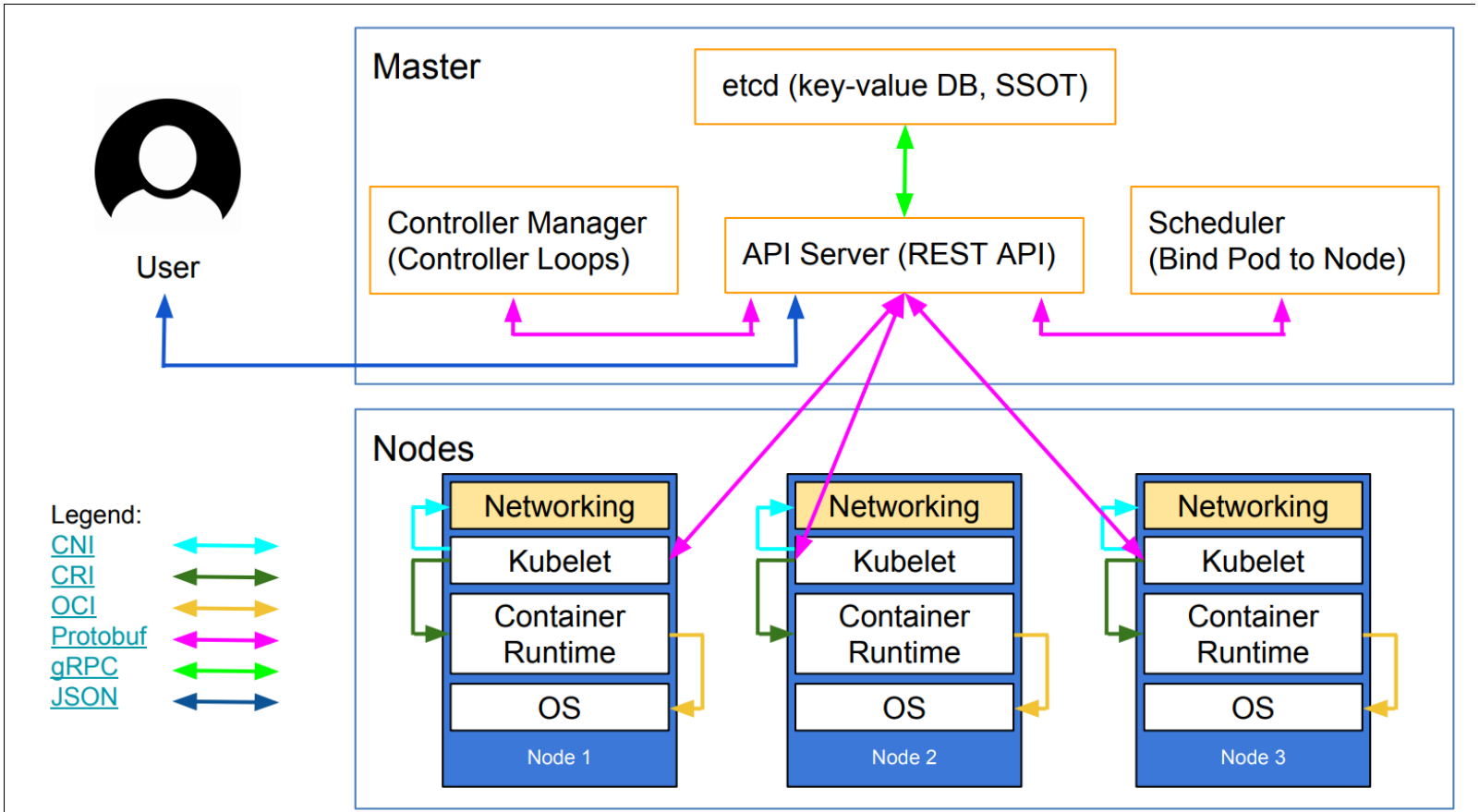


Image Source Lucas Kardlstrom: https://speakerdeck.com/luxas/kubeadm-cluster-creation-internals-from-self-hosting-to-upgradability-and-ha

Kubernetes Architecture deeper dive

- **Components communication and execution**
  - All components pass from API Server to communicate between them
  - Only kubelet runs as regular system component and can run the other components as pods
  - Components on worker nodes need to run on same node but components of master can be split across multiple nodes
- **Etcd**
  - Distributed, consistent key value store.
  - Only the API-Server talks with etcd directly. All other components talk to etcd indirectly through API-Server based on “Optimistic concurrency control”
  - Uses Raft consensus algorithm to decide on the actual state based on quorum (majority).
- **API-Server**
  - It provides a CRUD (Create, Read, Update, Delete) interface for querying and modifying the cluster state over a RESTful API.
  - Performs authentication, authorization and admission control through different plugins before accessing state in etcd
  - Watch mechanism to inform clients for modifications on objects.

Kubernetes Architecture deeper dive

- **Controller manager**
  - It combines a multitude of controllers performing various reconciliation tasks.
  - Each controller watches the API server for changes to resources (Deployments, Services, and so on) and perform operations for each change
  - It reconciles the actual state with the desired state (specified in the resource's spec section)
- **Scheduler**
  - It updates pods definition and through the API-server watch mechanism the kubelet is notified to execute a pod.
  - The default scheduling algorithm determines acceptable nodes and selects the best one for the pod based on various configurable parameters.
  - Multiple schedulers can run simultaneously in the cluster and a pod can use whichever is more adapted.

Kubernetes Architecture deeper dive

- **Kubelet**
  - Component responsible for everything running on a worker node
  - Registers the node it's running on by creating a Node resource in the API server.
  - Continuously monitor the API server for Pods that have been scheduled to the node
  - Start the pod's containers using the configured container runtime
- **Kube-proxy**
  - makes sure connections to the service IP and port end up at one of the pods backing that service
  - when a service is backed by more than one pod, the proxy performs load balancing across those pods.
  - Not real proxy but uses iptables rules to redirect packets handled in kernel space for better performance
- **Network Model**
  - Nodes and pods share one big flat IP network
  - All nodes and pods can reach each other directly without NAT
  - The network is usually set up by a Container Network Interface (CNI) plugin (more than 15 official implementations exist)

Kubernetes Control Plane HA

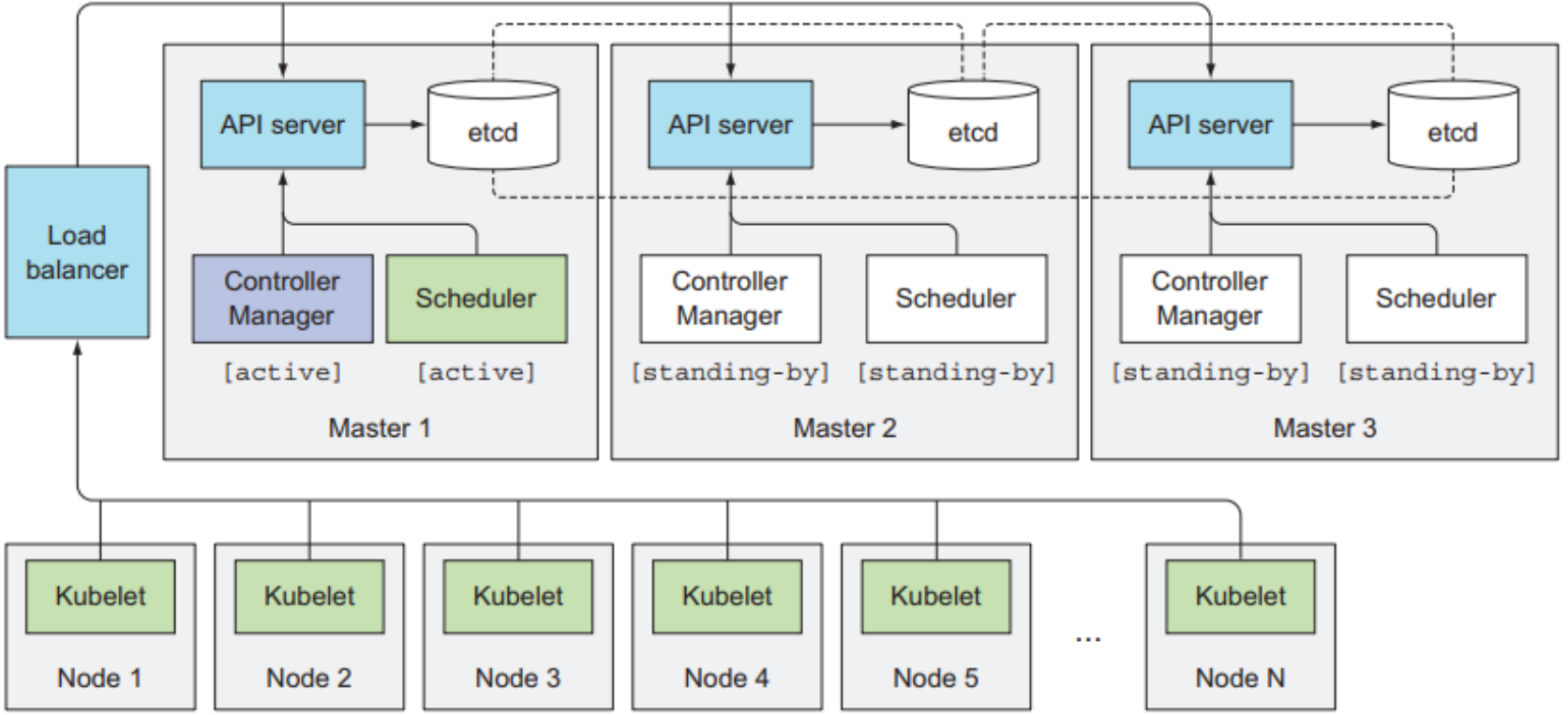


Image Source: Kubernetes in Action, Marko Luksa

Setting up Kubernetes Cluster

- Various deployment options (non-exhaustive list)
  - AKS on Azure, EKS or kops on AWS, GKE on Google Cloud
  - Minikube on local machine
  - Kubicorn or kubespray for hybrid multi-cloud solutions
  - And plenty of commercial (open-source or not) options with different levels of offered support RancherOS, RedHat Openshift, Suse, Ubuntu, etc
- For our tutorial we have used kubeadm on freshly installed VM instances running Ubuntu 16.04 LTS deployed on AWS
  - Install Docker
  - Install Kubernetes packages
  - Run kubeadm init on the master node
  - Set up Weave (the overlay network)
  - Run kubeadm join on the other nodes (with the token produced by kubeadm init)
  - Copy the configuration file generated by kubeadm init

Kubernetes Features

- Automatic scheduling and allocation
- Automatic scaling
- Resources Monitoring
- Run everywhere
- Self Healing
- Storage Orchestration
- Automatic rollouts and rollbacks

Tutorial Contents

- The tutorial provides steps to get to know Kubernetes through various exercises.
  - We have explicitly abstracted the creation of the Kubernetes cluster to focus more on interesting functionalities and exercises.

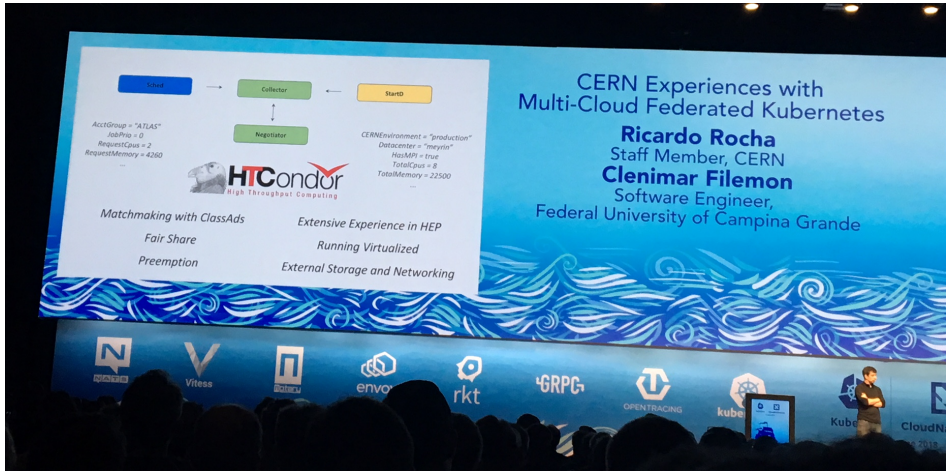
1) Launch VMs on AWS with your account and deploy a kubernetes cluster on those VMs using a preconfigured script
2) Connect on your VMs to start playing with your Kubernetes cluster, deploy a web dashboard and activate monitoring
3) Execute Big Data jobs based on Spark
4) Make use of multi-schedulers functionality, activate a new scheduling policy and use it
5) Activate and use exclusive CPU Management policy
6) Enable and use pod auto-scaling functionality

Hands-On Time

- Connect on this link and follow the steps described there:

https://github.com/RyaxTech/kube-tutorial

Kubernetes and HPC



kubernetes

Documentation Blog Partners Community Case Studies

Kubernetes Blog

Tuesday, August 22, 2017

Kubernetes Meets High-Performance Computing

Editor's note: today's post is by Robert Lalonde, general manager at Univa, on supporting mixed HPC and containerized applications

Anyone who has worked with Docker can appreciate the enormous gains in efficiency achievable with containers. While Kubernetes excels at orchestrating containers, high-performance computing (HPC) applications can be tricky to deploy on Kubernetes.

In this post, I discuss some of the challenges of running HPC workloads with Kubernetes, explain how organizations approach these challenges today, and suggest an approach for supporting mixed workloads on a shared Kubernetes cluster. We will also provide information and links to a case study on a customer, IHME, showing how Kubernetes is extended to service their HPC workloads seamlessly while retaining scalability and interfaces familiar to HPC users.

HPC workloads unique challenges

In Kubernetes, the base unit of scheduling is a Pod: one or more Docker containers scheduled to a cluster host. Kubernetes assumes that workloads are containers. While Kubernetes has the notion of Cron Jobs and Jobs that run to completion, applications deployed on Kubernetes are typically long-running services, like web servers, load balancers or data stores and while they are highly dynamic with pods coming and going, they differ greatly from HPC application

ryax technologies

Kubernetes and HPC

- Kubernetes has not been designed for HPC, hence it will never be as optimized as Slurm, however:
  - In the effort to enable support of Big Data and Artificial Intelligence the software is being enhanced with functionalities that will eventually interest HPC community.
  - The convergence of HPC and Big Data will further motivate the usage of Kubernetes in HPC.

Some interesting development efforts and functionalities already in place:

- CRI-O lightweight container runtime + rootless usage
- CPU Pinning and Isolation
- Device Plugins (GPUs, Infiniband, FPGA, etc)
- NUMA Management
- SR-IOV Networking plugin
- Spark 2.3.0 with Kubernetes scheduler instead of Yarn
- Pods Priorities and Preemption

Kubernetes and HPC

Some interesting development efforts and communities to follow:

- Container Runtime
  - CRI-O – OCI stable runtime follow rootless
  - KataContainers – OCI VM Containers
  - Singularity – Syllabs seems to be working on support of Kubernetes
  - Gvisor – Google sandboxed containers
- Kubernetes cluster federation – for offloading on hybrid clouds
- Kubeflow to make Machine Learning simple, portable and scalable

Some Kubernetes Resources

- KubeCon 2018 presentations with slides:

https://events.linuxfoundation.org/events/kubecon-cloudnativecon-europe-2018/program/schedule/

- KubeCon 2018 videos:

https://www.youtube.com/channel/UCvqbFHwN-nwalWPjPUKpvTA/videos

ryax
technologies