**Kubernetes Mastery**

**Course:**[**http://www.kubernetesmastery.com**](http://www.kubernetesmastery.com/)

**Slides:**[**https://slides.kubernetesmastery.com**](https://slides.kubernetesmastery.com/)

**A brief introduction**

* **This was initially written by [Jérôme Petazzoni](https://twitter.com/jpetazzo) to support in-person, instructor-led workshops and tutorials**
* **Credit is also due to**[**multiple contributors**](https://github.com/BretFisher/kubernetes-mastery/graphs/contributors)**— thank you!**
* **I recommend using the Slack Chat to help you ...**
* **... And be comfortable spending some time reading the Kubernetes**[**documentation**](https://kubernetes.io/docs/)**...**
* **... And looking for answers on [StackOverflow](http://stackoverflow.com/questions/tagged/kubernetes) and other outlets**

**Hands on, you shall practice**

* **Nobody ever became a Jedi by spending their lives reading Wookiepedia**
* **Likewise, it will take more than merely *reading* these slides to make you an expert**
* **These slides include *tons* of exercises and examples**
* **They assume that you have access to a Kubernetes cluster**

**What and why of orchestration**

* **There are many computing orchestrators**
* **They make decisions about when and where to "do work"**
* **We've done this since the dawn of computing: Mainframe schedulers, Puppet, Terraform, AWS, Mesos, Hadoop, etc.**
* **Since 2014 we've had a resurgence of new orchestration projects because:**
  1. **Popularity of distributed computing**
  2. **Docker containers as a app package and isolated runtime**
* **We needed "many servers to act like one, and run many containers"**
* **And the Container Orchestrator was born**

**Container orchestrator**

* **Many open source projects have been created in the last 5 years to:**
  + **Schedule running of containers on servers**
  + **Dispatch them across many nodes**
  + **Monitor and react to container and server health**
  + **Provide storage, networking, proxy, security, and logging features**
  + **Do all this in a declarative way, rather than imperative**
  + **Provide API's to allow extensibility and management**

**Major container orchestration projects**

* **Kubernetes, aka K8s**
* **Docker Swarm (and Swarm classic)**
* **Apache Mesos/Marathon**
* **Cloud Foundry**
* **Amazon ECS (not OSS, AWS-only)**
* **HashiCorp Nomad**
* **Many of these tools run on top of Docker Engine**
* **Kubernetes is the *one* orchestrator with many *distributions***

## Kubernetes distributions

* **Kubernetes "vanilla upstream" (not a distribution)**
* **Cloud-Managed distros: AKS, GKE, EKS, DOK...**
* **Self-Managed distros: RedHat OpenShift, Docker Enterprise, Rancher, Canonical Charmed, openSUSE Kubic...**
* **Vanilla installers: kubeadm, kops, kubicorn...**
* **Local dev/test: Docker Desktop, minikube, microK8s**
* **CI testing: kind**
* **Special builds: Rancher k3s**
* **And**[**Many, many more...**](https://kubernetes.io/partners/#conformance)**(86 as of June 2019)**

# Kubernetes concepts

* **Kubernetes is a container management system**
* **It runs and manages containerized applications on a cluster (one or more servers)**
* **Often this is simply called "container orchestration"**
* **Sometimes shortened to Kube or K8s ("Kay-eights" or "Kates")**

## Basic things we can ask Kubernetes to do

* **Start 5 containers using image**atseashop/api:v1.3
* **Place an internal load balancer in front of these containers**
* **Start 10 containers using image**atseashop/webfront:v1.3
* **Place a public load balancer in front of these containers**
* **It's Black Friday (or Christmas), traffic spikes, grow our cluster and add containers**
* **New release! Replace my containers with the new image**atseashop/webfront:v1.4
* **Keep processing requests during the upgrade; update my containers one at a time**

## Other things that Kubernetes can do for us

* **Basic autoscaling**
* **Blue/green deployment, canary deployment**
* **Long running services, but also batch (one-off) and CRON-like jobs**
* **Overcommit our cluster and evict low-priority jobs**
* **Run services with stateful data (databases etc.)**
* **Fine-grained access control defining what can be done by whom on which resources**
* **Integrating third party services (service catalog)**
* **Automating complex tasks (operators)**

## Kubernetes architecture

## Kubernetes architecture: the nodes

* **The nodes executing our containers run a collection of services:**
  + **a container Engine (typically Docker)**
  + **kubelet (the "node agent")**
  + **kube-proxy (a necessary but not sufficient network component)**
* **Nodes were formerly called "minions"**

**(You might see that word in older articles or documentation)**

## Kubernetes architecture: the control plane

* **The Kubernetes logic (its "brains") is a collection of services:**
  + **the API server (our point of entry to everything!)**
  + **core services like the scheduler and controller manager**
  + etcd**(a highly available key/value store; the "database" of Kubernetes)**
* **Together, these services form the control plane of our cluster**
* **The control plane is also called the "master"**

## Running the control plane on special nodes

* **It is common to reserve a dedicated node for the control plane**

**(Except for single-node development clusters, like when using minikube)**

* **This node is then called a "master"**

**(Yes, this is ambiguous: is the "master" a node, or the whole control plane?)**

* **Normal applications are restricted from running on this node**

**(By using a mechanism called**[**"taints"**](https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/)**)**

* **When high availability is required, each service of the control plane must be resilient**
* **The control plane is then replicated on multiple nodes**

**(This is sometimes called a "multi-master" setup)**

## Running the control plane outside containers

* **The services of the control plane can run in or out of containers**
* **For instance: since**etcd**is a critical service, some people deploy it directly on a dedicated cluster (without containers)**

**(This is illustrated on the first "super complicated" schema)**

* **In some hosted Kubernetes offerings (e.g. AKS, GKE, EKS), the control plane is invisible**

**(We only "see" a Kubernetes API endpoint)**

* **In that case, there is no "master node"**

**For this reason, it is more accurate to say "control plane" rather than "master."**

## Do we need to run Docker at all?

**No!**

* **By default, Kubernetes uses the Docker Engine to run containers**
* **Or leverage other pluggable runtimes through the Container Runtime Interface**
* **(deprecated)**
* [**containerd**](https://github.com/containerd/containerd/blob/master/README.md)**: maintained by Docker, IBM, and community**
* **Used by Docker Engine, microK8s, k3s, GKE, and standalone; has**ctr**CLI**
* [**CRI-O**](https://github.com/cri-o/cri-o/blob/master/README.md)**: maintained by Red Hat, SUSE, and community; based on containerd**
* **Used by OpenShift and Kubic, version matched to Kubernetes**
* [**And more**](https://kubernetes.io/docs/setup/production-environment/container-runtimes/)

## Do we need to run Docker at all?

**Yes!**

* **In this course, we'll run our apps on a single node first**
* **We may need to build images and ship them around**
* **We can do these things without Docker  
  (and get diagnosed with NIH¹ syndrome)**
* **Docker is still the most stable container engine today  
  (but other options are maturing very quickly)**

## Do we need to run Docker at all?

* **On our development environments, CI pipelines ... :**

**Yes, almost certainly**

* **On our production servers:**

**Yes (today)**

**Probably not (in the future)**

## Interacting with Kubernetes

* **We will interact with our Kubernetes cluster through the Kubernetes API**
* **The Kubernetes API is (mostly) RESTful**
* **It allows us to create, read, update, delete resources**
* **A few common resource types are:**
  + **node (a machine — physical or virtual — in our cluster)**
  + **pod (group of containers running together on a node)**
  + **service (stable network endpoint to connect to one or multiple containers)**

**Pods**

* **Pods are a new abstraction!**
* **A *pod* can have multiple containers working together**
* **(But you usually only have on container per pod)**
* **Pod is our smallest deployable unit; Kubernetes can't mange containers directly**
* **IP addresses are associated with *pods*, not with individual containers**
* **Containers in a pod share**localhost**, and can share volumes**
* **Multiple containers in a pod are deployed together**
* **In reality, Docker doesn't know a pod, only containers/namespaces/volumes**

**Credits**

* **The first diagram is courtesy of Lucas Käldström, in**[**this presentation**](https://speakerdeck.com/luxas/kubeadm-cluster-creation-internals-from-self-hosting-to-upgradability-and-ha)
  + **it's one of the best Kubernetes architecture diagrams available!**
* **The second diagram is courtesy of Weaveworks**
  + **a *pod* can have multiple containers working together**
  + **IP addresses are associated with *pods*, not with individual containers**

**Both diagrams used with permission.**

shpod**: For a consistent Kubernetes experience ...**

* **You can use [shpod](https://github.com/bretfisher/shpod) for examples**
* shpod**provides a shell running in a pod on the cluster**
* **It comes with many tools pre-installed (helm, stern, curl, jq...)**
* **These tools are used in many exercises in these slides**
* shpod**also gives you shell completion and a fancy prompt**
* **Create it with**kubectl apply -f https://k8smastery.com/shpod.yaml
* **Attach to shell with**kubectl attach --namespace=shpod -ti shpod
* **After finishing course**kubectl delete -f https://k8smastery.com/shpod.yaml

**First contact with**kubectl

* 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)**

**First contact with**kubectl

* 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)**

* **On our machines, there is a**~/.kube/config**file with:**
  + **the Kubernetes API address**
  + **the path to our TLS certificates used to authenticate**
* **You can also use the**--kubeconfig**flag to pass a config file**
* **Or directly**--server**,**--user**, etc.**
* kubectl**can be pronounced "Cube C T L", "Cube cuttle", "Cube cuddle"...**
* **I'll be using the official name "Cube Control"**

kubectl**is the new SSH**

* **We often start managing servers with SSH**

**(installing packages, troubleshooting ...)**

* **At scale, it becomes tedious, repetitive, error-prone**
* **Instead, we use config management, central logging, etc.**
* **In many cases, we still need SSH:**
  + **as the underlying access method (e.g. Ansible)**
  + **to debug tricky scenarios**
  + **to inspect and poke at things**

**The parallel with**kubectl

* **We often start managing Kubernetes clusters with**kubectl

**(deploying applications, troubleshooting ...)**

* **At scale (with many applications or clusters), it becomes tedious, repetitive, error-prone**
* **Instead, we use automated pipelines, observability tooling, etc.**
* **In many cases, we still need**kubectl**:**
  + **to debug tricky scenarios**
  + **to inspect and poke at things**
* **The Kubernetes API is always the underlying access method**

kubectl get

* **Let's look at our**Node**resources with**kubectl get**!**
* **Look at the composition of our cluster:**

kubectl get node

* **These commands are equivalent:**

kubectl get no

kubectl get node

kubectl get nodes

**Obtaining machine-readable output**

* kubectl get**can output JSON, YAML, or be directly formatted**
* **Give us more info about the nodes:**

kubectl get nodes -o wide

* **Let's have some YAML:**

kubectl get no -o yaml

**See that**kind: List**at the end? It's the type of our result!**

**(Ab)using**kubectl**and**jq

* **It's super easy to build custom reports**
* **Show the capacity of all our nodes as a stream of JSON objects:**

kubectl get nodes -o json |

jq ".items[] | {name:.metadata.name} + .status.capacity"

## Viewing details

* **We can use**kubectl get -o yaml**to see all available details**
* **However, YAML output is often simultaneously too much and not enough**
* **For instance,**kubectl get node node1 -o yaml**is:**
  + **too much information (e.g.: list of images available on this node)**
  + **not enough information (e.g.: doesn't show pods running on this node)**
  + **difficult to read for a human operator**
* **For a comprehensive overview, we can use**kubectl describe**instead**

**Exploring types and definitions**

* **We can list all available resource types by running**kubectl api-resources **(In Kubernetes 1.10 and prior, this command used to be**kubectl get**)**
* **We can view the definition for a resource type with:**

kubectl explain type

* **We can view the definition of a field in a resource, for instance:**

kubectl explain node.spec

* **Or get the list of all fields and sub-fields:**

kubectl explain node --recursive

## Introspection vs. documentation

* **We can access the same information by reading the**[**API documentation**](https://kubernetes.io/docs/reference/#api-reference)
* **The API documentation is usually easier to read, but:**
  + **it won't show custom types (like Custom Resource Definitions)**
  + **we need to make sure that we look at the correct version**
* kubectl api-resources**and**kubectl explain**perform introspection**

**(they communicate with the API server and obtain the exact type definitions)**

**Type names**

* **The most common resource names have three forms:**
  + **singular (e.g.**node**,**service**,**deployment**)**
  + **plural (e.g.**nodes**,**services**,**deployments**)**
  + **short (e.g.**no**,**svc**,**deploy**)**
* **Some resources do not have a short name**
* Endpoints**only have a plural form**

**(because even a single**Endpoints**resource is actually a list of endpoints)**

## More get commands: Services

* **A service is a stable endpoint to connect to "something"**

**(In the initial proposal, they were called "portals")**

* **List the services on our cluster with one of these commands:**

kubectl get services

kubectl get svc

## More get commands: Listing running containers

* **Containers are manipulated through pods**
* **A pod is a group of containers:**
  + **running together (on the same node)**
  + **sharing resources (RAM, CPU; but also network, volumes)**
* **List pods on our cluster:**

kubectl get pods

## Namespaces

* **Namespaces allow us to segregate resources**
* **List the namespaces on our cluster with one of these commands:**

kubectl get namespaces

kubectl get namespace

kubectl get ns

## Namespaces

* **Namespaces allow us to segregate resources**
* **List the namespaces on our cluster with one of these commands:**

kubectl get namespaces

kubectl get namespace

kubectl get ns

**You know what ... This***kube-system***thing looks suspicious.**

**In fact, I'm pretty sure it showed up earlier, when we did:**

kubectl describe node <node-name>