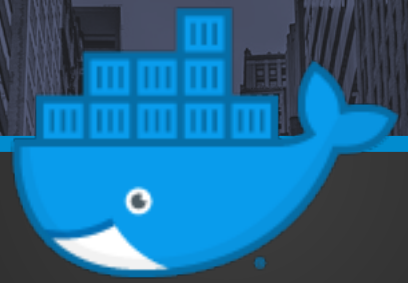
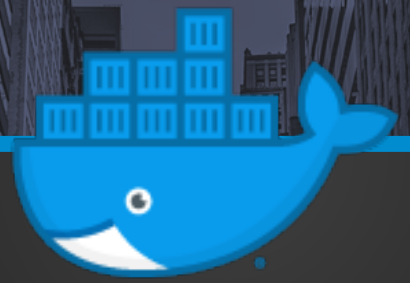
Kubernetes install



Intro to orchestration and kubernetes

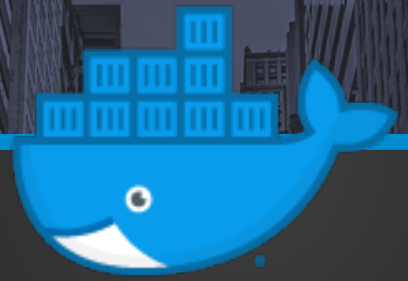
orchestration



orchestration - overview

- What's wrong with just running containers using Docker run in production? Remember: we're doing microservices now, with lots of services! And we need redundancy. So at a minimum we're talking about dozens of running containers for even a basic application!
- Why not just run Compose on servers?
- Healing and scaling
- How do you manage hundreds of servers with thousands of containers?
- What options are out there? Swarm, ECS, Kubernetes, Mesosphere.
- Discuss clusters, health monitoring, services, service discovery, geographic distribution, load balancing.
- What is a control plane or scheduler?
- Managed deployments

Kubernetes basics



kubernetes – a brief history

- 2003 – Started at Google as The Borg System to manage Google Search. They needed a sane way to manage their large-scale container clusters! But it was still very primitive compared to what we have today. It becomes big and rather messy, with many different languages and concepts due to it's organic growth.
- 2013 –Docker hits the scene and really revolutionizes computing by providing build tools, image distribution, and runtimes. This makes containers user friendly and adoption of containers explodes.
- 2014 – 3 Google engineers decide to build a next generation orchestrator that takes many lessons learned into account, built for public clouds, and open sourced. They build Kubernetes. Microsoft, Red Hat, IBM, and Docker join in.
- 2015 – Cloud Native Computing Foundation is created by Google and the Linux Foundation. More companies join in. Kubernetes 1.0 is released, followed by more major upgrades that year. KubeCon is launched.
- 2016 – K8S goes mainstream. Many supporting products are introduced including Minikube, kops, Helm. Rapid releases of big features. More companies join in and the community of passionate people explodes.
- 2017 to now – Kubernetes becomes the dominant orchestration system and de-facto standard for Docker microservices. K8S now has fully managed services by all major cloud providers. Handsome and talented instructors travel the country preaching K8S. ☺️

kubernetes – the alternative

Before cloud and Docker orchestration came along, production support and releases was a wild west full of danger and long nights.

Releases were a MAJOR nightmare for many. Spending most/all of a night on the phone with an entire ops team supporting a release and fixing issues was commonplace. The build pipeline was slow and prone to errors, and frequently took days to complete. When releases failed, you “failed forward” instead of rolling back.

Let me tell you about my own experiences before containerized workloads.

Let’s also talk through some stories the class has of painful releases. We’ll try to talk about how those issues are resolved using orchestration.

Kubernetes architecture

kubernetes – declarative model

K8S utilizes infrastructure-as-code concepts. All resources and configurations are done in declarative YAML files called resource templates or manifests and applied to your cluster.

Example resource:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
  spec:
    containers:
    - name: myapp-container
      image: busybox
      command: ['sh', '-c', 'echo Hello Kubernetes! && sleep 3600']
```

kubernetes – minikube

Minikube is a small K8S cluster intended to run on your local machine. We will spend lots of time here.

It provides much of the functionality that a full cluster has. You can run a completely functional application on your machine.

Minikube commands:

- Minikube start
- Minikube stop
- Minikube ip – retrieve the IP of your cluster for use in a browser
- Minikube dashboard – bring up the web UI dashboard

kubernetes – docker kubernetes

New in Docker, you can now enable Kubernetes on your workstation with a simple checkbox. You'll find it in Preferences.

Unlike Minikube where a virtual machine is launched, you can access the services using localhost instead of the VM IP address.

You can disable and shut down your local cluster using the Docker menu.

kubernetes – kubectl

Kubectl provides the command line interaction with your cluster. This is how you'll view and manage things like pods, services, etc, and apply template files to your cluster. Commands are formatted as `kubectl <command>`

Commands:

- Get <service, pod, etc> <name> - get information about the resource
  - `kubectl get pod mypod`
- Get all – return a full page of details about your cluster's resources
- Apply – apply a template file to your cluster
  - `kubectl apply -f workloads.yaml`
- Describe – get detailed information about a resource
  - `kubectl describe service myservice`
- Logs – return console output from a pod (or a container inside it)
  - `kubectl logs mypod`

kubectl – contexts

Kubectl can be pointed to different clusters for management. If you installed minikube and also activated Kubernetes for Docker, you'll need to tell kubectl which cluster to work on. You do this with context.

- kubectl config view – shows config information, including available contexts
- kubectl config current-context – shows the current context
- kubectl config use-context docker-for-desktop – switch kubectl to send commands to the cluster identified by the name "docker-for-desktop"

You can also switch contexts using the Docker menu by expanding the Kubernetes item.

kubernetes – core components

- Master components – The controller for the cluster. Also known as the “control plane”
- Nodes – the servers in our cluster. There are master nodes and worker nodes.
- Workloads – running containers to do the work. These can be created in several ways as we’ll see.
- Services – provide network connectivity to workloads.

kubernetes – master components

- Master components are pods themselves that run in the kube-system namespace
- Master node – a special node to hold many of the master components. For fault tolerance, 3 of these should be used for production.
- Kube-apiserver – exposes the K8S API
- etcd – key-value store to hold cluster data
- kube-scheduler – places pods onto nodes

Note that the loss of the master node does not mean that your cluster is down! No management will happen until the master is back up, but it is not a gateway for traffic to your services.

kubernetes – Worker nodes

- Worker servers that run your workloads.
- Several overhead pods run on every node (these are master components):
  - Kubelet – an agent pod that ensures containers are running in a pod
  - Kube-proxy – a pod that maintains network connections/routing
  - Other optional pods and controllers (ie. Logging)
- Has the container runtime (ie. Docker)

kubernetes – pods

- The basic building block of K8S. It represents a running process in your cluster.
- It encapsulates a container. In rare cases, it can contain multiple containers that are tightly coupled and must run together.
- Provides additional configuration about how the container runs.
- K8S manages pods, not containers.
- Every pod gets a unique IP.
- If multiple containers per pod, pods can communicate with localhost.
- You will rarely interact directly with pods, except perhaps viewing logs. You will interact with Deployments or ReplicaSets instead.
- When a pod dies, no replacement is automatically created, unless it was created as part of a ReplicaSet or Deployment.
- Pods are private by default. How do you get your web server accessible? That's coming up next with a service.
- Pods get a label, which we'll learn more about later.
- Let's launch a pod and examine it.

kubernetes – pod sidecar

- Usually you want to have a pod contain just a single container.
- However, there are cases when you'd want to have multiple containers in a single pod.
- The sidecar pattern is an example of this.
- With a sidecar, you run a second container in a pod whose job is to take action and support the primary container.
- Logging is a good example, where a sidecar container sends logs from the primary container to a centralized logging system.

kubernetes – pod exercise

1. Exercise file: exercises/Day 4/pod.yaml
2. Let's apply it
3. Review all pods
4. Describe the pod
5. How do we access port 80? We can't.
6. Let's launch a bash shell inside the pod and test it.
    1. kubectl exec -it nginx-pod -- /bin/bash
    2. curl http://localhost. Oh no! curl not found!
    3. Apt-get update
    4. Apt-get install curl
    5. Now try again
    6. We should now get the HTML for the nginx default page
7. Exit and leave the pod running
8. Side note: if I launch this pod again, will curl be there?

kubernetes – services

- Pods are mortal and unstable. Especially with ReplicaSets, pods can frequently be shuffled out of service, especially with scaling events. Services step in to provide a single, stable point of entry to your pods.
- Pods attached to a service are specified using label selectors
- Services are how we provide traffic access to pods.
- Services act as load balancers for our pods. They just distribute traffic in a round-robin pattern.
- Service types:
  - ClusterIP – default option. Provides access inside the cluster. Good for services that should remain internal to the application like a database.
  - NodePort – Provides a port outside of the cluster. Makes your service public. Note: the port you expose on the cluster is only valid from 30000-32767
  - LoadBalancer – Only valid on cloud implementations, and creates a load balancer and attaches it to your service.
- Let's give this a go with an exercise.

kubernetes – labels and selectors

- Kubernetes uses selectors to identify pods that will be included in a service
- These are name/value pairs that you define.
- The scheme you use is up to you.
- You can have multiple selectors and all must match.
- You can define elaborate schemes to enable sophisticated deployment mechanisms such as blue/green.

kubernetes – services exercise

1. Exercise file: exercises/Day 4/service.yaml
2. Review the file. Pay attention to the type and ports.
3. Apply to cluster
4. Now we can access the service from our own computer at http://localhost:30080
5. Now go delete your pod and try the URL again. Oh no! Our service is down!
6. Don’t delete the service. We’ll use it again shortly.

kubernetes – replicaset

- When we deleted the pod in the last exercise, our service went down!
- We don't want to manage pods directly, because they're mortal
- Instead we work with replicasets to manage the creation and replacement of pods.
- With a replicaset, any pods that die will be recreated to maintain the minimum number.
- This is essentially our pod declaration with some additional meta data.
- Let's go create one.

kubernetes – replicaset exercise

1. Exercise file: exercises/Day 4/replicaset.yaml
2. Review the file. Note the labels and replicas. Does the spec section look familiar?
3. Apply the file to your cluster.
4. Kubectl get all. Note that we now have pods, services, and replicasets
5. We now have our nginx pod running again! Try the URL to the service.
6. Now go delete your pod. Does the service go down? Does it go down temporarily? Do a kubectl get all and note the name of the new pod.
7. Now go increase the replicas setting in the template file and reapply.
8. Repeat the pod delete step, and note that you now have a resilient service that auto heals but also has multiple pods to serve requests!
9. When done, delete the replicaset.
10. How do we handle updates to your container images, such as new version releases? Let's talk deployments.

kubernetes – deployment

- Deployments are basically replicasets with some extra management included for managing updates to your replicasets and pods.
- They are declarative and you provide the end state. Kubernetes takes care of getting your pods to that end state in a managed way.
- On a rollout, K8S creates a new replicaset and starts moving pods to the new one in a controlled way, before removing the old replicaset.
- Note: deployments create a replicaset programmatically. Do not manage the replicaset directly!
- You can see rollout status with: kubectl rollout status deployment nginx-deployment
- You can see rollout history with: kubectl rollout history deployment nginx-deployment
- You can rollback with: kubectl rollout undo deployment nginx-deployment
- You can pause and resume rollouts
- There are tons of settings here: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

kubernetes – deployment exercise

1. Exercise file: exercises/Day 4/deployment.yaml
2. Review the file. Does it look just like a replicaset? Sure does! Note the image tag of 1.0.
3. Apply it to the cluster. Kubectl get all and note we now have pods, services, replicasets, and deployments. Your service should be back up at your cluster IP URL and will behave the same way as your replicaset did.
4. Now update the deployment file to upgrade nginx from 1.0 to 2.0. When you apply it, quickly kubectl get all, and notice how a new replicaset and pods are created. At the URL, if you keep refreshing, you'll see version 1 change to version 2 without the service ever going down.
5. When done, delete the deployment and service

kubernetes – deployment methods

- Kubernetes supports rolling deployments, which is default. But sometimes applications can’t work with that type of deployment and need a full cutover. You need another option.
- Blue/green is a common pattern, where an entire new copy of the application is created, traffic is drained to the old deployment, and then switched over all at once.
- Blue/green is not supported by K8S. But you can still implement it yourself manually or with some scripting.

kubernetes – blue/green exercise

1. Use the deployment file in exercises/day 4/deployment.yaml
2. Ensure that it is using image vergeops/versioned-nginx:1.0
3. Make sure that there are two labels. One for the application and one for the version.
4. Apply this deployment to your cluster.
5. Now create a second deployment, but with image vergeops/versioned-nginx: 2.0. Make sure this one is using a different set of labels.
6. Now using the services.yaml file, ensure that the label selector matches your first deployment.
7. Apply the service to your cluster and test in a browser. Verify that the UI shows “version 1”
8. Now apply the second deployment.
9. With two deployments running, modify the labels in your service to match the second deployment.
10. Apply the service and validate that the UI changed to ”version 2”

kubernetes – volume

- The filesystem in a container is ephemeral, because containers are immutable. Volumes provide persistent storage on the host system.
- Volumes can be a variety of types, from the host hard drive to cloud storage volumes like AWS EBS.
- To read up on the details: https://kubernetes.io/docs/concepts/storage/volumes/
- K8S volumes include lots of management automatically, such as mounting your EBS volumes to pods and unmounting them.
- This topic can get messy. We're going to dive into more depth on day 5 when we create our infrastructure on AWS.

kubernetes – namespace

- These are a way to create virtual clusters on the same physical clusters.
- For example, you could create dev and test environments on the same K8S cluster hardware.
- You likely won't need to worry about this much.
- You should know about the kube-system namespace. System level stuff goes here.
- The default namespace where all your stuff goes if you don't tell K8S otherwise.
- Everything we've done so far was in the default. Later we'll add a few resources to the kube-system namespace.

kubernetes – probes

- Kubelet uses liveness probes to know when to restart a container.
- Kubelet uses readiness probes to know when to start sending traffic to a container. A pod is ready when all containers are ready. You can use this to prevent pods from accepting traffic before they're fully initialized.
- You can combine liveness and readiness probes for a robust pod.
- Liveness probes can be different types: TCP, HTTP, filesystem check
- You have some options to fine tune (they all have defaults):
  - initialDelaySeconds
  - periodSeconds – default 10
  - timeoutSeconds – default 1
  - successThreshold – default 1
  - failureThreshold – default 3

kubernetes – probe exercise

In this exercise, a nginx container starts, but after 30 seconds the index.html file is removed, which causes nginx to no longer return a 200 status code. The probe will pick up on that and restart the container, starting the process over.

1. Exercise file: exercises/Day 4/probe.yaml
2. Review the template file. Notice there are two resources in this one!
3. Apply to your cluster.
4. Open a browser and verify that you can access it.
5. Kubectl get all. Notice the pod's restarts column.
6. Kubectl describe pod liveness-http, and notice the output shows the restarts
7. When done, delete the resources using the template file. Kubectl delete –f probe.yaml

kubernetes – Configmap

- Containers often need several environment variables to function properly so that configuration is externalized. But passing in variables directly to containers is messy.
- ConfigMaps allow you to decouple configurations at the cluster level and have containers refer to them.
- This also allows you to reuse configurations.

kubernetes – Configmap exercise

1. Exercise file: exercises/Day 4/configmap.yaml
2. Review the file and apply to your cluster
3. Take a look at the log output for the MongoDB pod
4. Also describe the MongoDB pod and notice the output references the configmap
5. When done, delete the resources

kubernetes – secret

- How do you store sensitive information? Should you include it in a Docker image? How about in a pod spec? Never!
- Secrets are small pieces of sensitive information that your pods can access at runtime. Think passwords, SSH keys, etc.
- Secrets are stored as volumes that your containers can access. Alternatively, they can be exposed as environment variables.
- When using kubectl get, you won't see the contents of a secret.
- Note that secrets are still accessible to those with access directly to the cluster. They are meant to protect from including them in Docker images which are more portable. It is best to have secrets managed by a limited set of people who know how to keep them safe. And don't just check them into source control alongside your resources.
- When secrets are updated, the containers automatically pick up the changes immediately.

kubernetes – secret exercise

1. Exercise file: exercises/Day 4/secret.yaml
2. Review the file and apply to your cluster
3. Take a look at the log output for the MongoDB pod
4. Also describe the MongoDB pod and notice the output references the secret. Also notice that it was mounted as a volume to the container for you.
5. When done, delete the resources

Note: for simplicity, the secret and deployment are together. Don’t do this in a real world scenario.

kubernetes – daemonset

- Runs a copy of the pod on every node in the cluster
- Any new node will get a new copy of the pod
- Any node removal cleans up the copy of the pod
- Useful for system level resources such as monitoring, logging, etc.
- This is how the master pods on the worker nodes run, such as the kube-proxy and kubelet.

kubernetes – statefulset

- Works like a deployment, but provides guarantees about the order and uniqueness of pods
- Pods get a consistent naming scheme that is ordered. For example, pod-0, pod-1, pod-2, etc.
- The spec is identical, except for the Kind statement
- Stable, persistent storage
- Not typical. You should aim for stateless components if possible and use Deployments instead.
- Useful when you have a group of servers that work together and need to know each others' names ahead of time. For example, we will create an ElasticSearch cluster later that uses StatefulSets.

kubernetes – job

- Jobs are short-lived processes that create pods to fulfill the work and then cleanup the pods when done.
- You can create jobs programmatically, or have timed jobs with CronJobs.
- Great for batch operations. Think daily import processes or perhaps an inventory management process that runs periodically. Maybe a backup job.
- Keep in mind that in certain situations the schedule can create multiple jobs based on one CronJob, so design accordingly.

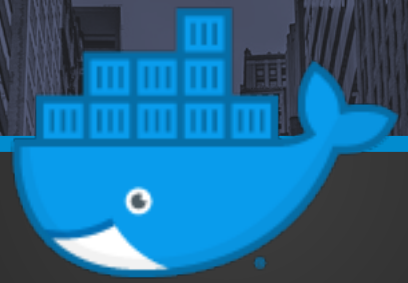
kubernetes – job exercise

1. Exercise file: exercises/Day 4/job.yaml
2. Review the file and apply to your cluster
3. Review the pods in the cluster, look at output, etc.
4. Delete when done

kubernetes – ingress

- Provides external access to services in your cluster. Usually HTTP
- Can provide load balancing, SSL, and name based virtual hosting.
- Can specify complex routing rules
- You can choose many different controllers to provide the functionality, including:
  - Nginx (maintained by K8S project)
  - Contour/Envoy
- Popular use is to centralize many microservices under one name using routing rules (AKA edge service or API gateway). For example:
  - Example.com/account points to account service
  - Example.com/orders points to order service
- Other solutions exist for this. For example, cloud providers have their own L7 load balancers such as Application Load Balancer from AWS. Ingress is just the K8S solution. You can also do it internally with your own app like we will do in the hackathon portion.

Cloud services



Cloud services – overview

- The three major cloud providers (AWS, GCP, and Azure) all have their own Kubernetes services
- They work by providing the “control plane” or master components for you rather than managing your own master nodes.
- They provide additional functionality and integration with their services such as autoscaling of nodes. They also give you some online console components.

schedulers

schedulers – overview

- Kubernetes ships with a default scheduler called “kube-scheduler”. Most of the time it will meet your needs. It is built to spread your workloads out across nodes in the most fault-tolerant way.
- But this scheduler doesn’t always fit your needs. For most uses it’ll be fine. But you may want to fine-tune pod placement depending on your use case.
- For example, you may want to use a mix of spot and reserved instances in your cluster. You want to make sure that a certain percentage of pods in a deployment get put onto reserved instances and the remainder get put on spot instances that could get removed with little notice. You could write a custom scheduler that adds some extra intelligence to the process of scheduling.
- So you can run your own scheduler. You can even run multiple schedulers and tell Kubernetes which one to use on specific workloads. In the example above, you may have some non-critical deployments that can withstand some instability, such as a reporting service. This you could use the default scheduler or write a custom scheduler that ONLY puts these pods on spot instances.

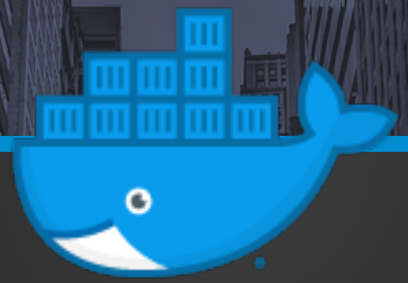
schedulers – overview

- Schedulers run as pods (deployment for resiliency) just like any other workload.
- You specify which scheduler to use in your workload. If you don’t choose one, the default scheduler will be used.

schedulers – lifecycle

- Schedulers have one job: find a suitable node for every pod and let the Kubernetes apiserver know, which will take care of placing and starting the pod.
- The lifecycle goes like this:
  - A pod is created and the information is stored in etcd. The node name is not filled in.
  - The scheduler notices there's a pod with no assigned node.
  - The scheduler finds a node that is suitable to the pod according to it's own internal rules. This is the main part you'd be customizing.
  - The scheduler tells apiserver to bind the node to the pod. apiserver updates etcd with the node name.
  - The kubelet on the node is watching the apiserver for any bound pods. When it sees it, then it starts the pod on the node.

helm



Helm – overview

- Helm is like a package manager for Kubernetes
- Whereas Docker Hub is a place to find third party Docker images, Helm kind of acts the same way.
- But Helm is there to help you manage, install, upgrade complex Kubernetes applications.

Helm – tiller

- Tiller is the Helm server side component that runs in your Kubernetes cluster.
- When you use Charts to manage your application, it uses Tiller to handle the cluster side of things.
- You install Tiller into your cluster with the Helm CLI:
  - `helm init --history-max 200`
- Tiller runs as standard Kubernetes resources in the kube-system namespace. We can view the deployment, replicaset, and pod by looking at that namespace:
  - `kubectl get all --all-namespaces`

Helm – charts

- Charts are the core concept in Helm
- For complex applications, you may have a whole collection of resource files. Deployments, Pods, ReplicaSets, Secrets, Volumes, DaemonSets, etc etc etc.
- To manage all this by hand means installing and managing lots of files and resources.
- Charts let you package everything together into one versioned archive for deployment.
- The files for a Chart are organized into a directory with the Chart name. It has a predetermined file structure:
  - Chart.yaml – the Chart information
  - Values.yaml – default configuration values
  - Charts/ - a directory of dependent charts
  - Templates/ - a directory of templates that, when combined with values, will generate valid Kubernetes resource files.
  - Other optional files like notes, readme, license, etc.

Helm – chart repositories

- After a chart is developed, it can be packaged into a .tgz archive.
- From here, it can be pushed up to Helm Hub, or your own repository.
- A repository works like Docker Hub.
- You can also stand up your own private Helm repository.
- Helm Hub is the public repository at hub.helm.sh

Helm – cli

- When you install Helm on your workstation, it comes with it's own CLI.
- All commands start with `helm`
- If you already have kubectl installed and configured, them the Helm CLI will pick up on that configuration to know what cluster to point to.
- We can use the following common commands:
  - Helm install <chart name>
  - Helm list – show charts installed in our cluster
  - Helm get <chart name> - get details on an installed chart
  - Helm delete <chart name>

Helm – installing a chart

- It's a good idea to update the repo before installing any charts.
  - `helm repo update`
- Then to install, just use:
  - `helm install <chart name>`

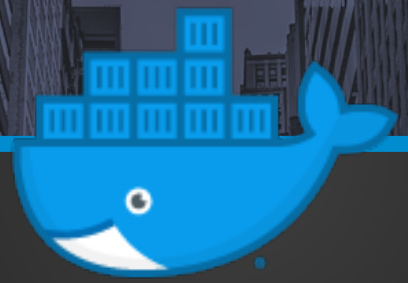
Helm – mysql walkthrough

- Let's use the example of Mysql. Mysql has many components to be able to run on Kubernetes:
  - ConfigMaps
  - Secrets
  - PersistentVolumes
  - Pods
  - ReplicaSets
  - Deployments
  - Services
- Rather than put together the resource files for all these ourselves, which is time consuming and requires knowledge of the inner workings of Mysql, we can instead just install the Helm Chart for Mysql

Helm – mysql walkthrough

- Search Helm Hub for mysql
- Find the stable/mysql Chart
- Review the Chart page
- Install it with `helm install --name mysql stable/mysql`
- Review the output, which gives instructions for forwarding the port, getting the root password, etc.
  - I will grab the root password, forward the port, and connect using Mysql Workbench

kops



kops – overview

- Kubernetes Operations
- https://github.com/kubernetes/kops
- The easiest way to get a production grade Kubernetes cluster running and maintain them. It has built-in support for AWS.
- I prefer to create a dedicated machine in AWS for doing cluster related work. Cloud9 fits the bill perfectly for this because it shuts down when you're done.
- Installation has some manual steps, but then you're all set.
- Kops is for creating the physical cluster itself and installing Kubernetes. Once created, you'll use kubectl to manage the Kubernetes cluster remotely.

kops – Initial install

- https://github.com/kubernetes/kops/blob/master/docs/aws.md
- You need kubectl installed first
  - `curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl`
  - `chmod +x ./kubectl`
  - `sudo mv ./kubectl /usr/local/bin/kubectl`
- Then install kops
  - `curl -LO https://github.com/kubernetes/kops/releases/download/$(curl -s https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f 4)/kops-linux-amd64`
  - `chmod +x kops-linux-amd64`
  - `sudo mv kops-linux-amd64 /usr/local/bin/kops`
- Get the code
  - `git clone https://github.com/VergeOps/k8s-rvstore.git`

kops – cluster setup

- During cluster creation, you'll get an error about a public SSH key. You need to create one.
  - `ssh-keygen -o` (leave the passphrase empty)
  - `kops create secret --name nwm-tim.k8s.local sshpublickey admin -i ~/.ssh/id_rsa.pub` (change the name to the name you chose)
- At this point you should be able to finish the instructions to get the cluster built

review and wrap up