S1 Hands-On Introduction to

Kubernetes

Sunday, October 29th 9:00 am–12:30 pm

at #LISA17 in SF

bit.ly/lisa17-k8s

presented by @ryanj, Developer Advocate at Red Hat

Administration

1. Introduction
   - Workshop Setup
2. Kubernetes Basics
   - Learn five K8s Primitives
3. Kubernetes Architecture
4. Local Development with Minikube
5. Wrap-up

Intro Survey / Who are you?

1. Are you doing anything with containers today?
2. Do you have any experience using Kubernetes?
3. Do you consider yourself to be basically proficient with the `kubectl` cli tool?
4. Can you name five basic primitives or resource types?
5. Can you name five architectural components provided by Kubernetes?

LISA17 S1 Training Prep

Bring a laptop with the following items pre-installed:

1. kubectl
2. minikube
3. docker
4. git

Install `kubectl`

Installation on linux/amd64:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
```

Installation on macOS:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
```

For other platforms, consult the official `kubectl` setup guide

To verify `kubectl` availability, try running:

```
kubectl help
```

Install `minikube`

Installation on linux/amd64:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/min
```

Installation on macOS:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/min
```

For other platforms, see the `minikube` release notes

Optionally, customize your cluster's memory or cpu allocation:

```
minikube config set memory 4096
minikube config set cpus 2
```

to verify `minikube` availability:

```
minikube version
```

Minikube Basics

`minikube` provides an easy way to run Kubernetes locally:

```
minikube start
```

When you are done, halt the VM to free up system resources:

```
minikube stop
```

Need a fresh start? Delete your VM instance with:

```
minikube delete
```

Minikube troubleshooting

If your minikube environment does not boot correctly:

1. Minikube requires an OS virtualization back-end
2. Most OSes include some support for virtualization
3. You can use the `--vm-driver` flag to select a specific virt provider

```
minikube start --vm-driver=virtualbox
```

Check the project README for more information about supported virtualization plugins

Still stuck? Consider signing up for OpenShift Starter or GKE:
bit.ly/k8s-gcloud

Install docker

Download and install a binary from the docker store

Or, use a package manager to install:

```
brew install docker
```

To verify `docker` availability:

```
docker version
```

To reference minikube's docker daemon from your host, run:

```
eval $(minikube docker-env)
```

Install git

Install `git` using the instructions here:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

To verify `git` availability, run:

```
git version
```

Ready?

Verify that your local Kubernetes environment is ready by running:

```
kubectl version
```

The output should include your `kubectl` version info, and the release version of the kubernetes API server (when available)

Let's Go!

Kubernetes Command-Line Basics

with kubectl

Why Kubernetes?

Kubernetes is...

1. An open source platform for running container-based distributed solutions, featuring a modular, HA systems architecture
2. The best way to actively manage distributed solutions at scale, based on years of industry expertise (Google-scale experience)
3. An extensible distributed-solutions modeling language with a huge community following
4. A multi-vendor effort to eliminate cloud lock-in through the adoption of "cloud native" solutions (capable of runnning on any infrastructure)

Kubernetes provides…

An API

API object primitives include the following attributes:

```
kind
apiVersion
metadata
spec
status
```

*mostly true

Basic K8s Terminology

1. node
2. pod
3. service
4. deployment
5. replicaSet

Node

A node is a host machine (physical or virtual) where containerized processes run.

Node activity is managed via one or more Master instances.

Try using `kubectl` to list resources by type:

```
kubectl get nodes
```

Request the same info, but output the results as structured yaml:

```
kubectl get nodes -o yaml
```

Fetch an individual resource by `type/id`, output as `json`:

```
kubectl get node/minikube -o json
```

View human-readable API output:

```
kubectl describe node/minikube
```

Observations:

- Designed to exist on multiple machines (distributed system)
  - high availability of nodes
  - platform scale out
- The API ambidextriously supports both json and yaml

Pod

A group of one or more co-located containers. Pods represent your minimum increment of scale.

"Pods Scale together, and they Fail together"
@theSteve0

List resources by type:

```
kubectl get pods
```

Create a new resource based on a json object specification:

```
curl https://raw.githubusercontent.com/ryanj/metrics-k8s/master/pod.json
```

```
kubectl create -f https://raw.githubusercontent.com/ryanj/metrics-k8s/master/
```

List resources by type:

```
kubectl get pods
```

Fetch a resource by type and id, output the results as `yaml`:

```
kubectl get pod metrics-k8s -o yaml
```

Notice any changes?

Observations:

- pods are scheduled to be run on nodes
- asyncronous fulfilment of requests
- declarative specifications
- automatic health checks, lifecycle management for containers (processes)

Service

Services (svc) establish a single endpoint for a collection of replicated pods, distributing inbound traffic based on label selectors

In our K8s modeling language they represent a load balancer. Their implementation often varies per cloud provider

Contacting your App

Expose the pod by creating a new `service` (or "loadbalancer"):

```
kubectl expose pod/metrics-k8s --port 2015 --type=NodePort
```

Contact your newly-exposed pod using the associated service id:

```
minikube service metrics-k8s
```

Schedule a pod to be deleted:

```
kubectl delete pod metrics-k8s
```

Contact the related service. What happens?:

```
minikube service metrics-k8s
```

Delete the service:

```
kubectl delete service metrics-k8s
```

Observations:

- *"service"* basically means *"loadbalancer"*
- Pods and Services exist independently, have disjoint lifecycles

Deployment

A `deployment` helps you specify container runtime requirements (in terms of pods)

Create a specification for your `deployment`:

```
kubectl run metrics-k8s --image=quay.io/ryanj/metrics-k8s \
--expose --port=2015 --service-overrides='{ "spec": { "type": "NodePort" } }'
--dry-run -o yaml > deployment.yaml
```

View the generated deployment spec file:

```
cat deployment.yaml
```

Bug?: You may need to edit this file, adding " - - - " (on it's own line) between resource 1 and resource 2 for a workaround.

Can you think of another way to fix this issue? json compatible?

Create a new resource based on your yaml specification:

```
kubectl create -f deployment.yaml
```

List resources by type:

```
kubectl get po,svc
```

Connect to your new deployment via the associated service id:

```
minikube service metrics-k8s
```

Replication

Scale up the `metrics-k8s` deployment to 3 replicas:

```
kubectl scale deploy/metrics-k8s --replicas=3
```

List pods:

```
kubectl get po
```

Edit `deploy/metrics-k8s`, setting `spec.replicas` to `5`:

```
kubectl edit deploy/metrics-k8s -o json
```

Save and quit. What happens?

```
kubectl get pods
```

AutoRecovery

Watch for changes to pod resources:

```
kubectl get pods --watch
```

In another terminal, delete several pods by id:

```
kubectl delete pod $(kubectl get pods | grep ^metrics-k8s | cut -f1 -s -d' '
```

What happend? How many pods remain?

```
kubectl get pods
```

Observations:

- Use the `--dry-run` flag to generate new resource specifications
- A deployment spec contains a pod spec

ReplicaSet

A `replicaset` provides replication and lifecycle management for a specific image release

Watch deployments (leave this running until the 'cleanup' section):

```
kubectl get deploy --watch
```

View the current state of your deployment:

```
minikube service metrics-k8s
```

Rollouts

Update your deployment's image spec to rollout a new release:

```
kubectl set image deploy/metrics-k8s metrics-k8s=quay.io/ryanj/metrics-k8s:v1
```

Reload your browser to view the state of your deployment

```
kubectl get rs,deploy
```

Rollbacks

View the list of previous rollouts:

```
kubectl rollout history deploy/metrics-k8s
```

Rollback to the previous state:

```
kubectl rollout undo deployment metrics-k8s
```

Reload your browser to view the state of your deployment

Cleanup

Cleanup old resources if you don't plan to use them:

```
kubectl delete service,deployment metrics-k8s
```

Close any remaining `--watch` listeners

Observations:

- The API allows for watch operations (in addition to get, set, list)
- ReplicaSets provide lifecycle management for pod resources
- Deployments create ReplicaSets to manage pod replication per rollout (per change in podspec: image:tag, environment vars)

break 1

Kubernetes Architecture

adapted for minikube

Kubernetes is designed ...

1. for managing distributed solutions at scale, based on years of industry expertise (Google-scale experience)
2. for high availabilty of the control plane and user workloads (when using pod replication), avoiding most single points of failure
3. with a modular control plane architecture, allowing many peices to be replaced without disrupting workload availability
4. to persist all of it's internal platform state within an etcd database

etcd

- distributed key-value store
- implements the RAFT consensus protocol

CAP theorum

1. Consistency
2. Availability
3. Partition tolerance

etcd is "CA"

Degraded Performance

Fault tolerance sizing chart:

CLUSTER SIZE	MAJORITY	FAILURE TOLERANCE
1	1	0
3	2	1
4	3	1
5	3	**2**
6	4	2
7	4	**3**
8	5	3
9	5	**4**

play.etcd.io

play.etcd.io/play

Kubernetes API

- gatekeeper for etcd (the only way to access the db)
- not required for pod uptime

API outage simulation

Example borrowed from Brandon Philips' "Fire Drills" from OSCON 2016:

https://github.com/philips/2016-OSCON-containers-at-scale-with-Kubernetes#fire-drills

Create a pod and a service. Verify that the service is responding.

```
kubectl run metrics-k8s --image=quay.io/ryanj/metrics-k8s \
--expose --port=2015 --service-overrides='{ "spec": { "type": "NodePort" } }'
```

```
minikube service metrics-k8s
```

ssh into minikube, kill the control plane:

```
minikube ssh
ps aux | grep "localkube"
sudo killall localkube
logout
```

Use kubectl to list pods:

```
kubectl get pods
The connection to the server mycluster.example.com was refused - did you spec
```

The API server is down!

Reload your service. Are your pods still available?

Kubelet

Runs on each node, listens to the API for new items with a matching `NodeName`

Kubernetes Scheduler

Assigns workloads to Node machines

Bypass the Scheduler

Create two pods:

```
kubectl create -f https://raw.githubusercontent.com/ryanj/metrics-k8s/master/
kubectl create -f https://gist.githubusercontent.com/ryanj/893e0ac5b3887674f8
```

View events:

```
kubectl get events
```

Did both pods get scheduled? run?

Kube DNS

Kube Proxy

CNI

- flannel
- canal

CRI

- containerd (docker)
- cri-o
- rkt

compatible with OCI image spec., runtime

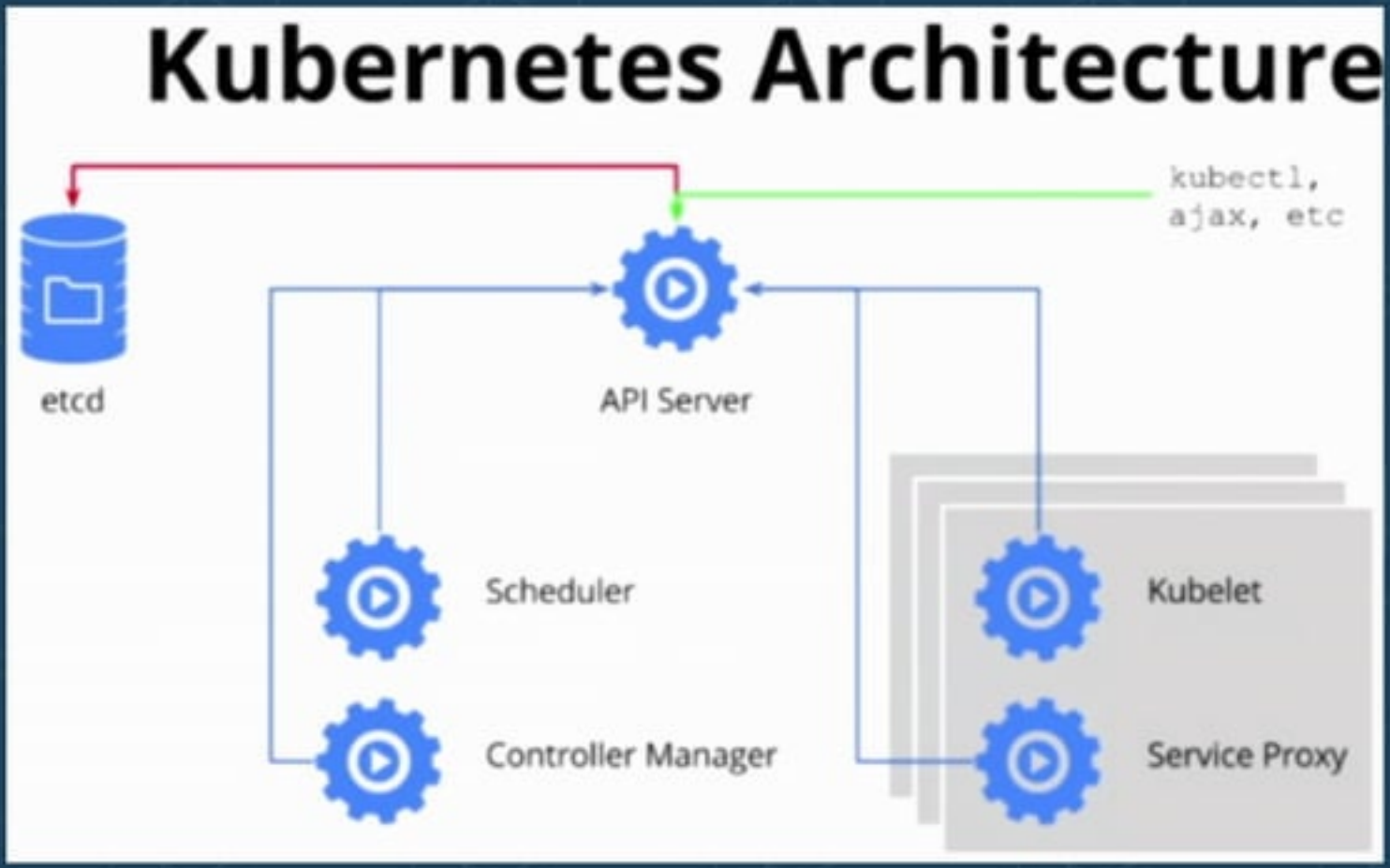
K8s Controllers

Controllers work to regulate the declarative nature of the platform state, reconsiling imbalances via a basic control loop

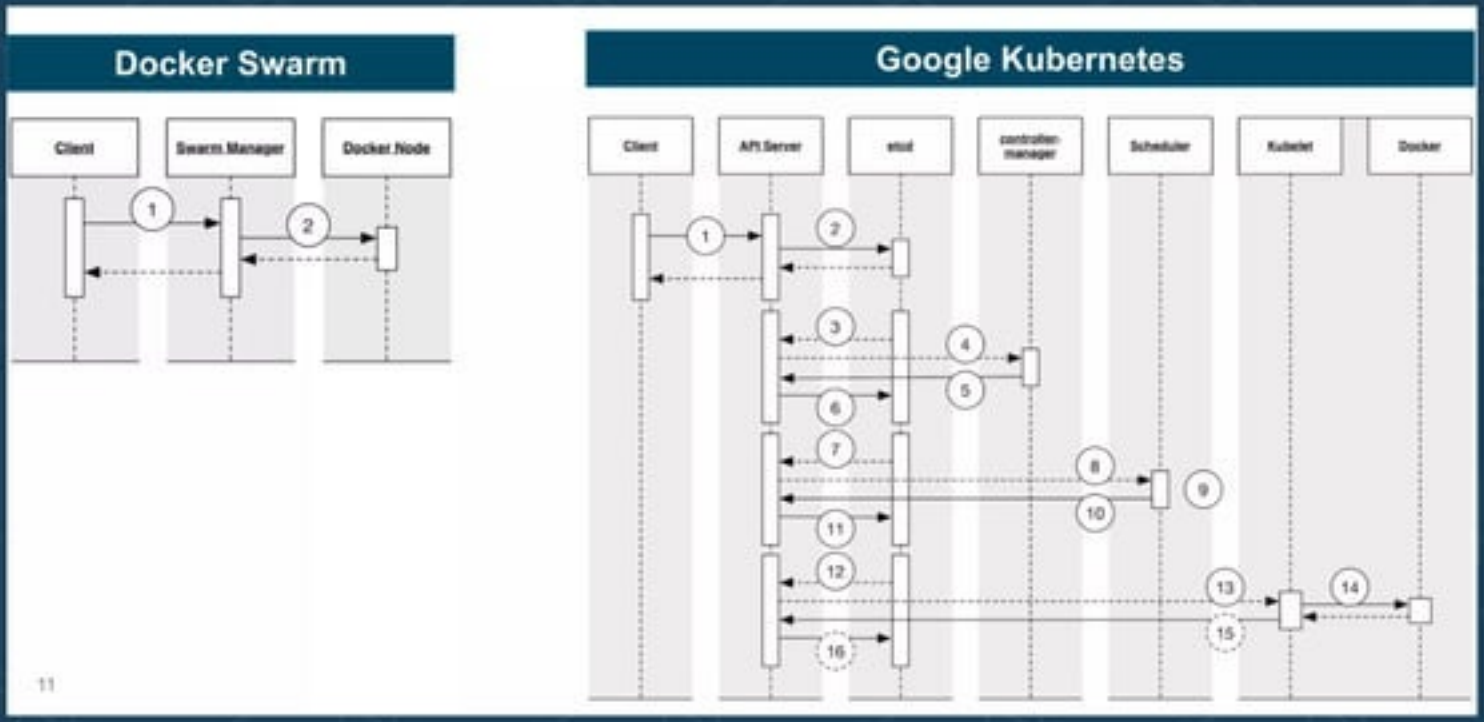
https://kubernetes.io/docs/admin/kube-controller-manager/

Kubernetes allows you to introduce your own custom controllers!

Architecture Diagram



Interaction Diagram



(copied from blog.docker.com)

break 2

Local Development

with

minikube

Kubernetes provides portable abstractions for working with distributed solitions:

1. standardized packaging (containers, volumes, pods)
2. load balancing (services)
3. scaling automation (replica sets)

Need any of these for local development?

Why run K8s locally?

As web development is increasingly being carried out using container-based microservices:

1. ability to offer reproducible development environments
   - reduce onboarding time for new devs
2. minimize deltas between dev and prod environments
   - fewer surprises when promoting code leads to faster velocity
3. decentralize your release pipeline, allow CI test suites to be run locally
   - provide functional / systems-integration feedback earlier in the dev lifecycle
4. potenial for fully offline development
   - <expanding brain meme>

Local Development Checklist:

1. onboarding - show someone new how to run the `:latest` release
2. preview changes - review changes and iterate on a solution
3. test changes - build and deploy
4. promote changes - git push

Onboarding

Onboarding - Yesterday's Jam

1. `git clone https://github.com/ryanj/metrics-k8s`
2. `cd metrics-k8s`
3. ~~`npm install`~~
4. ~~`npm start`~~

Onboarding - Add K8s

Generate kubernetes `deployment` and `service` specifications, both named `metrics-review`:

```
kubectl run metrics-review --image=quay.io/ryanj/metrics-k8s \
--expose --port=2015 --service-overrides='{ "spec": { "type": "NodePort" } }'
--dry-run -o yaml > metrics-review.yaml
```

Onboarding - deploy :latest

Test your generated spec:

```
kubectl create -f metrics-review.yaml
```

Minikube users will be able to open the resulting service in their browser by running:

```
minikube service metrics-review
```

Preview Changes

Preview - local files

First, share your local clone of `metrics-k8s` with minikube:

```
minikube mount $(pwd):/var/www/html
```

Preview - hostPath

Next, produce a new deployment spec that includes (minimal) support for live development workflows:

1. `cp metrics-review.yaml metrics-dev.yaml`
2. replace `metrics-review` with `metrics-dev` (global)
3. Add a `hostPort` volume to access your local repo:

```
      spec:
        containers:
        - image: quay.io/ryanj/metrics-k8s
          name: metrics-dev
          ports:
          - containerPort: 2015
          resources: {}
+         volumeMounts:
+         - mountPath: /var/www/html
+           name: metrics-src
+       volumes:
+       - name: metrics-src
+         hostPath:
```

Preview

The resulting file should look just like the included metrics-dev.yaml file from the `metrics-k8s` git repo.

Try launching it with:

```
kubectl create -f metrics-dev.yaml
```

Preview

Verify that any changes written to your local repo become immediately visible when reloading your browser window:

1. view your latest

```
minikube service metrics-dev
```

2. make a change to `index.html`
3. reload your browser

Test

Test - Rollout

1. Verify that your `docker-env` is configured for minikube
2. Run a build

```
docker build . -t yourname/metrics-k8s:v1
```

3. Update `metrics-review.yaml`, setting the container `image` to:

```
yourname/metrics-k8s:v1
```

4. Apply the changes locally:

```
kubectl apply -f metrics-review.yaml
```

5. Check your latest before promoting:

```
minikube service metrics-review
```

Promote Changes

Promoting Changes

1. `git push`?
2. send PR?
3. next steps TBD / handled by CI suite

Wrap Up

Resources

1. Training materials to-go: bit.ly/k8s-workshops
2. Home: kubernetes.io
3. Docs: K8s documentation
4. Community: K8s Special Interest Groups (SIGs)
5. eBook: Kubernetes: Scheduling the Future at Cloud Scale
6. eBook: Docker Security: Using Containers Safely in Production
7. eBook: Microservices vs. Service-Oriented Architecture
8. eBook: OpenShift for Developers

Exit Survey

1. Have you ever developed using containers?
2. Do you have any experience using Kubernetes?
3. Do you consider yourself to be basically proficient with the `kubectl` cli tool?
4. Can you name five basic primitives or resource types?
5. Can you name five pieces of k8s architecture?
6. Are you prepared to onboard a new web dev?

Congratulations!

on completing the

[S1] Hands-on Intro to Kubernetes

tutorial at #LISA17 in SF

bit.ly/lisa17-k8s

Please remember to complete your tutorial evaluation!