Kubernetes Tasks Documentation

Release 0.1

Petr Ruzicka

Oct 18, 2019

Contents:

This Guide Structure

This guide is designed to complement instructor-led presentations by providing step-by-step instructions for hands-on exercises.

CHAPTER 1

Kubernetes Installation

It's expected, that you will install Kubernetes to 3 VMs / hosts - to have multinode installation. The installation part is taken from these two URLs:

- https://kubernetes.io/docs/setup/independent/install-kubeadm/
- https://kubernetes.io/docs/setup/independent/create-cluster-kubeadm/

1.1 Master node installation

SSH to the first VM which will be your Master node:

```
$ ssh root@node1
```

Enable packet forwarding:

```
$ sed -i 's/^#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/' /etc/sysctl.d/99-sysctl.
↪conf
$ sysctl --quiet --system
```

Set the Kubernetes version which will be installed:

```
$ KUBERNETES_VERSION="1.10.3"
```

Set the proper CNI URL:

```
$ CNI_URL="https://raw.githubusercontent.com/coreos/flannel/v0.10.0/Documentation/
↪kube-flannel.yml"
```

For Flannel installation you need to use proper "pod-network-cidr":

```
$ POD_NETWORK_CIDR="10.244.0.0/16"
```

Add the Kubernetes repository (details):

```
$ apt-get update -qq && apt-get install -y -qq apt-transport-https curl
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
$ tee /etc/apt/sources.list.d/kubernetes.list << EOF2
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF2
```

Install necessary packages:

```
$ apt-get update -qq
$ apt-get install -y -qq docker.io kubelet=${KUBERNETES_VERSION}-00 kubeadm=$
↪{KUBERNETES_VERSION}-00 kubectl=${KUBERNETES_VERSION}-00
```

Install Kubernetes Master:

```
$ kubeadm init --pod-network-cidr=$POD_NETWORK_CIDR --kubernetes-version v$
↪{KUBERNETES_VERSION}
```

Copy the "kubectl" config files to the home directory:

```
$ test -d $HOME/.kube || mkdir $HOME/.kube
$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ chown -R $USER:$USER $HOME/.kube
```

Install CNI:

```
$ export KUBECONFIG=/etc/kubernetes/admin.conf
$ kubectl apply -f $CNI_URL
```

Your Kuberenets Master node should be ready now. You can check it using this command:

```
$ kubectl get nodes
```

1.2 Worker nodes installation

Let's connect the worker nodes now. SSH to the worker nodes and repeat these commands on all of them in paralel:

```
$ ssh root@node2
$ ssh root@node3
```

Set the Kubernetes version which will be installed:

```
$ KUBERNETES_VERSION="1.10.3"
```

Add the Kubernetes repository (details):

```
$ apt-get update -qq && apt-get install -y -qq apt-transport-https curl
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
$ tee /etc/apt/sources.list.d/kubernetes.list << EOF2
deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF2
```

Enable packet forwarding:

```
$ sed -i 's/^#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/' /etc/sysctl.d/99-sysctl.
↪conf
$ sysctl --quiet --system
```

Install necessary packages:

```
$ apt-get update -qq
$ apt-get install -y -qq docker.io kubelet=${KUBERNETES_VERSION}-00 kubeadm=$
↪{KUBERNETES_VERSION}-00 kubectl=${KUBERNETES_VERSION}-00
```

All the woker nodes are prepared now - let's connect them to master node. SSH to the master node again and generate the "joining" command:

```
$ ssh -t root@node1 "kubeadm token create --print-join-command"
```

You sould see something like:

```
$ kubeadm join <master-ip>:<master-port> --token <token> --discovery-token-ca-cert-
↪hash sha256:<hash>
```

Execute the generated command on all worker nodes:

```
$ ssh -t root@node2 "kubeadm join --token ..."
$ ssh -t root@node3 "kubeadm join --token ..."
```

SSH back to the master nodes and check the cluster status - all the nodes should appear there in "Ready" status after while:

```
$ ssh root@node1
$ # Check nodes
$ kubectl get nodes
```

Allow pods to be scheduled on the master:

```
$ kubectl taint nodes node1 node-role.kubernetes.io/master-
```

Enable routing from local machine (host) to the kubernetes pods/services/etc. Adding routes (10.244.0.0/16, 10.96.0.0/12) -> [$NODE1_IP]:

```
$ sudo bash -c "ip route | grep -q 10.244.0.0/16 && ip route del 10.244.0.0/16; ip␣
↪route add 10.244.0.0/16 via $NODE1_IP"
$ sudo bash -c "ip route | grep -q 10.96.0.0/12  && ip route del 10.96.0.0/12;  ip␣
↪route add 10.96.0.0/12  via $NODE1_IP"
```

1.3 Real installation example

CHAPTER 2

Kubernetes Basics

Create directory where the files will be stored

```
$ mkdir files
```

Enable bash-completion for kubectl (bash-completion needs to be installed)

```
$ source <(kubectl completion bash)
```

Check the cluster status (if it is healthy)

```
$ kubectl get componentstatuses
```

List all namespaces

```
$ kubectl get namespaces
```

Create namespace 'myns'

```
$ kubectl create namespace myns
```

Change default namespace for current context

```
$ kubectl config set-context $(kubectl config current-context) --namespace=myns
```

List out all of the nodes in our cluster

```
$ kubectl get pods -o wide --all-namespaces --show-labels --sort-by=.metadata.name
```

Get more details about a specific node

```
$ kubectl describe node $(kubectl get node --output=jsonpath="{.items[0].metadata.
↪name}")
```

CHAPTER 3

Helm Installation

Helm installation: https://github.com/kubernetes/helm/blob/master/docs/rbac.md

```
$ curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
$ kubectl create serviceaccount tiller --namespace kube-system
$ kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
↪serviceaccount=kube-system:tiller
$ helm init --wait --service-account tiller
$ helm repo update
```

Install Traefik - Træfik is a modern HTTP reverse proxy and load balancer

```
$ helm install stable/traefik --wait --name my-traefik --namespace kube-system --set␣
↪serviceType=NodePort,dashboard.enabled=true,accessLogs.enabled=true,rbac.
↪enabled=true,metrics.prometheus.enabled=true
$ kubectl describe svc my-traefik --namespace kube-system
```

Install rook - File, Block, and Object Storage Services for your Cloud-Native Environment

```
$ helm repo add rook-master https://charts.rook.io/master
$ helm install rook-master/rook-ceph --wait --namespace rook-ceph-system --name my-
↪rook --version $(helm search rook-ceph | awk "/^rook-master/ { print \$2 }")
```

Create your Rook cluster

```
$ kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/
↪examples/kubernetes/ceph/cluster.yaml
```

Running the Toolbox with ceph commands

```
$ kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/
↪examples/kubernetes/ceph/toolbox.yaml
```

Create a storage class based on the Ceph RBD volume plugin

```
$ kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/
↪examples/kubernetes/ceph/storageclass.yaml
```

Create a shared file system which can be mounted read-write from multiple pods

```
$ kubectl create -f https://raw.githubusercontent.com/rook/rook/master/cluster/
↪examples/kubernetes/ceph/filesystem.yaml
$ sleep 150
```

Check the status of your Ceph installation

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph status
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd status
```

Check health detail of Ceph cluster

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph health detail
```

Check monitor quorum status of Ceph

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph quorum_status --format json-pretty
```

Dump monitoring information from Ceph

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph mon dump
```

Check the cluster usage status

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph df
```

Check OSD usage of Ceph

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd df
```

Check the Ceph monitor, OSD, pool, and placement group stats

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph mon stat
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd stat
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd pool stats
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph pg stat
```

List the Ceph pools in detail

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd pool ls detail
```

Check the CRUSH map view of OSDs

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd tree
```

List the cluster authentication keys

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph auth list
```

Change the size of Ceph replica for “replicapool” pool

```
$ kubectl get pool --namespace=rook-ceph replicapool -o yaml | sed "s/size: 1/size: 3/
↪" | kubectl replace -f -
```

List details for “replicapool”

```
$ kubectl describe pool --namespace=rook-ceph replicapool
```

See the manifest of the pod which should use rook/ceph

```
$ tee files/rook-ceph-test-job.yaml << EOF
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rook-ceph-test-pv-claim
spec:
  storageClassName: rook-ceph-block
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: batch/v1
kind: Job
metadata:
  name: rook-ceph-test
  labels:
    app: rook-ceph-test
spec:
  template:
    metadata:
      labels:
        app: rook-ceph-test
    spec:
      containers:
      - name: rook-ceph-test
        image: busybox
        command: [ 'dd', 'if=/dev/zero', 'of=/data/zero_file', 'bs=1M', 'count=100' ]
        volumeMounts:
          - name: rook-ceph-test
            mountPath: "/data"
      restartPolicy: Never
      volumes:
      - name: rook-ceph-test
        persistentVolumeClaim:
          claimName: rook-ceph-test-pv-claim
EOF
```

Check the ceph usage

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd status
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph df
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd df
```

Apply the manifest

```
$ kubectl apply -f files/rook-ceph-test-job.yaml
$ sleep 10
```

Check the ceph usage again

```
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd status
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph df
$ kubectl -n rook-ceph exec rook-ceph-tools -- ceph osd df
```

List the Persistent Volume Claims

```
$ kubectl get pvc
```

Delete the job

```
$ kubectl delete job rook-ceph-test
```

Install Prometheus - Prometheus Operator creates/configures/manages Prometheus clusters atop Kubernetes

```
$ helm repo add coreos https://s3-eu-west-1.amazonaws.com/coreos-charts/stable/
$ helm install coreos/prometheus-operator --wait --name my-prometheus-operator --
↪namespace monitoring
$ helm install coreos/kube-prometheus --name my-kube-prometheus --namespace␣
↪monitoring --set alertmanager.ingress.enabled=true,alertmanager.ingress.
↪hosts[0]=alertmanager.domain.com,alertmanager.storageSpec.volumeClaimTemplate.spec.
↪storageClassName=rook-block,alertmanager.storageSpec.volumeClaimTemplate.spec.
↪accessModes[0]=ReadWriteOnce,alertmanager.storageSpec.volumeClaimTemplate.spec.
↪resources.requests.storage=20Gi,grafana.adminPassword=admin123,grafana.ingress.
↪enabled=true,grafana.ingress.hosts[0]=grafana.domain.com,prometheus.ingress.
↪enabled=true,prometheus.ingress.hosts[0]=prometheus.domain.com,prometheus.
↪storageSpec.volumeClaimTemplate.spec.storageClassName=rook-block,prometheus.
↪storageSpec.volumeClaimTemplate.spec.accessModes[0]=ReadWriteOnce,prometheus.
↪storageSpec.volumeClaimTemplate.spec.resources.requests.storage=20Gi
$ GRAFANA_PASSWORD=$(kubectl get secret --namespace monitoring my-kube-prometheus-
↪grafana -o jsonpath="{.data.password}" | base64 --decode ; echo)
$ echo "Grafana login: admin / $GRAFANA_PASSWORD"
```

Install Heapster - Compute Resource Usage Analysis and Monitoring of Container Clusters

```
$ helm install stable/heapster --name my-heapster --set rbac.create=true
```

Install Kubernetes Dashboard - General-purpose web UI for Kubernetes clusters

```
$ helm install stable/kubernetes-dashboard --name=my-kubernetes-dashboard --namespace␣
↪monitoring --set ingress.enabled=true,rbac.clusterAdminRole=true
```

CHAPTER 4

Pods

Check 'kuard-pod.yaml' manifest which will run kuard application once it is imported to Kubernetes

```
$ tee files/kuard-pod.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
EOF
```

Start pod from the pod manifest via Kubernetes API (see the 'ContainerCreating' status)

```
$ kubectl apply --filename=files/kuard-pod.yaml; kubectl get pods
$ sleep 40
```

List pods (-o yaml will print all details)

```
$ kubectl get pods --namespace myns -o wide
```

Check pod details

```
$ kubectl describe pods kuard
```

Get IP for a kuard pod

```
$ kubectl get pods kuard -o jsonpath --template={.status.podIP}
```

Configure secure port-forwarding to access the specific pod exposed port using Kubernetes API Access the pod by opening the web browser with url: http://127.0.0.1:8080 and http://127.0.0.1:8080/fs/{etc,var,home}

```
$ kubectl port-forward kuard 8080:8080 &
```

Stop port forwarding

```
$ pkill -f "kubectl port-forward kuard 8080:8080"
```

Get the logs from pod (-f for tail) (–previous will get logs from a previous instance of the container)

```
$ kubectl logs kuard
```

Copy files to/from containers running in the pod

```
$ kubectl cp --container=kuard /etc/os-release kuard:/tmp/
```

Run commands in your container with exec (-it for interactive session). Check if I am in container

```
$ kubectl exec kuard -- cat /etc/os-release
```

Delete pod - see the status 'Terminating'

```
$ kubectl delete pods/kuard; kubectl get pods
$ sleep 30
```

Check pods - the kuard should disappear form the 'pod list'

```
$ kubectl get pods
```

CHAPTER 5

Health Checks

Check 'kuard-pod-health.yaml' manifest which will start kuard and configure HTTP health check

```
$ tee files/kuard-pod-health.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  volumes:
    - name: "kuard-data"
      hostPath:
        path: "/var/lib/kuard"
  containers:
    - image: gcr.io/kuar-demo/kuard-amd64:1
      name: kuard
      volumeMounts:
        - mountPath: "/data"
          name: "kuard-data"
      ports:
        - containerPort: 8080
          name: http
          protocol: TCP
      resources:
        requests:
          cpu: "100m"
          memory: "128Mi"
        limits:
          cpu: "1000m"
          memory: "256Mi"
      # Pod must be ready, before Kubernetes start sending traffic to it
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
```

(continues on next page)

(continued from previous page)

```
        # Check is done every 2 seconds starting as soon as the pod comes up
        periodSeconds: 2
        # Start checking once pod is up
        initialDelaySeconds: 0
        # If three successive checks fail, then the pod will be considered not ready.
        failureThreshold: 3
        # If only one check succeeds, then the pod will again be considered ready.
        successThreshold: 1
      livenessProbe:
        httpGet:
          path: /healthy
          port: 8080
        # Start probe 5 seconds after all the containers in the Pod are created
        initialDelaySeconds: 5
        # The response must be max in 1 second and status HTTP code must be between␣
↪200 and 400
        timeoutSeconds: 1
        # Repeat every 10 seconds
        periodSeconds: 10
        # If more than 3 probes failed - the container will fail + restart
        failureThreshold: 3
EOF
```

Create a Pod using this manifest and then port-forward to that pod

```
$ kubectl apply -f files/kuard-pod-health.yaml
$ sleep 30
```

Point your browser to http://127.0.0.1:8080 then click 'Liveness Probe' tab and then 'fail' link - it will cause to fail health checks

```
$ kubectl port-forward kuard 8080:8080 &
```

Stop port forwarding

```
$ pkill -f "kubectl port-forward kuard 8080:8080"
```

You will see 'unhealthy' messages in the in the following output

```
$ kubectl describe pods kuard | tail
```

Delete pod

```
$ kubectl delete pods/kuard
$ sleep 10
```

CHAPTER 6

Labels, annotations, selectors

Create app1-prod deployment with labels (creates also Deployment)

```
$ kubectl run app1-prod --image=gcr.io/kuar-demo/kuard-amd64:1 --replicas=3 --
↪port=8080 --labels="ver=1,myapp=app1,env=prod"
```

Create service (only routable inside cluster). The service is assigned Cluster IP (DNS record is automatically created) which load-balance across all of the pods that are identified by the selector

```
$ kubectl expose deployment app1-prod
```

Create app1-test deployment

```
$ kubectl run app1-test --image=gcr.io/kuar-demo/kuard-amd64:2 --replicas=1 --labels=
↪"ver=2,myapp=app1,env=test"
```

Create app2-prod

```
$ kubectl run app2-prod --image=gcr.io/kuar-demo/kuard-amd64:2 --replicas=2 --
↪port=8080 --labels="ver=2,myapp=app2,env=prod"
```

Create service

```
$ kubectl expose deployment app2-prod
```

Check if the DNS record was properly created for the Cluster IPs. app2-prod [name of the service], myns [namespace that this service is in], svc [service], cluster.local. [base domain name for the cluster]

```
$ kubectl run nslookup --rm -it --restart=Never --image=busybox -- nslookup app2-prod
$ kubectl run nslookup --rm -it --restart=Never --image=busybox -- nslookup app2-prod.
↪myns
```

Create app2-staging

```
$ kubectl run app2-staging --image=gcr.io/kuar-demo/kuard-amd64:2 --replicas=1 --
↪labels="ver=2,myapp=app2,env=staging"
```

Show deployments

```
$ kubectl get deployments -o wide --show-labels
```

Change labels

```
$ kubectl label deployments app1-test "canary=true"
```

Add annotation - usually longer than labels

```
$ kubectl annotate deployments app1-test description="My favorite deployment with my␣
↪app"
```

List 'canary' deployments (with canary label)

```
$ kubectl get deployments -o wide --label-columns=canary
```

Remove label

```
$ kubectl label deployments app1-test "canary-"
```

List pods including labels

```
$ kubectl get pods --sort-by=.metadata.name --show-labels
```

List pods ver=2 using the –selector flag

```
$ kubectl get pods --selector="ver=2" --show-labels
```

List pods with 2 tags

```
$ kubectl get pods --selector="myapp=app2,ver=2" --show-labels
```

List pods where myapp=(app1 or app2)

```
$ kubectl get pods --selector="myapp in (app1,app2)" --show-labels
```

Label multiple pods

```
$ kubectl label pods -l canary=true my=testlabel
```

List all services

```
$ kubectl get services -o wide
```

Get service details

```
$ kubectl describe service app1-prod
```

Get service endpoints

```
$ kubectl describe endpoints app1-prod
```

List IPs belongs to specific pods

```
$ kubectl get pods -o wide --selector=myapp=app1,env=prod --show-labels
```

Cleanup all deployments

```
$ kubectl delete services,deployments -l myapp
```

CHAPTER 7

ReplicaSet

Show minimal ReplicaSet definition

```
$ tee files/kuard-rs.yaml << EOF
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  name: kuard
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kuard
      version: "2"
  template:
    metadata:
      labels:
        app: kuard
        version: "2"
    spec:
      containers:
        - name: kuard
          image: "gcr.io/kuar-demo/kuard-amd64:2"
EOF
```

Create ReplicaSet

```
$ kubectl apply -f files/kuard-rs.yaml
```

Check pods

```
$ kubectl get pods
```

Check ReplicaSet details

```
$ kubectl describe rs kuard
```

The pods have the same labels as ReplicaSet

```
$ kubectl get pods -l app=kuard,version=2 --show-labels
```

Check if pod is part of ReplicaSet

```
$ kubectl get pods -l app=kuard,version=2 -o json | jq ".items[].metadata"
```

Scale up ReplicaSet

```
$ kubectl scale replicasets kuard --replicas=4
```

New pods are beeing created

```
$ kubectl get pods -l app=kuard --show-labels
```

Delete ReplicaSet

```
$ kubectl delete rs kuard
```

CHAPTER 8

DaemonSets and NodeSelector

Add labels to your nodes (hosts)

```
$ kubectl label nodes node2 ssd=true
```

Filter nodes based on labels

```
$ kubectl get nodes --selector ssd=true
```

Check 'nginx-fast-storage.yaml' which will provision nginx to ssd labeled nodes only. By default a DaemonSet will create a copy of a Pod on every node

```
$ tee files/nginx-fast-storage.yaml << EOF
apiVersion: extensions/v1beta1
kind: "DaemonSet"
metadata:
  labels:
    app: nginx
    ssd: "true"
  name: nginx-fast-storage
spec:
  template:
    metadata:
      labels:
        app: nginx
        ssd: "true"
    spec:
      nodeSelector:
        ssd: "true"
      containers:
        - name: nginx
          image: nginx:1.10.0
EOF
```

Create daemonset from the nginx-fast-storage.yaml

```
$ kubectl apply -f files/nginx-fast-storage.yaml
```

Check the nodes where nginx was deployed

```
$ kubectl get pods -o wide
```

Add label ssd=true to the node3 - nginx should be deployed there automatically

```
$ kubectl label nodes node3 ssd=true
```

Check the nodes where nginx was deployed (it should be also on node3 with ssd=true label)

```
$ kubectl get pods -o wide
```

Check the nodes where nginx was deployed

```
$ kubectl delete ds nginx-fast-storage
```

CHAPTER 9

Jobs

One-shot Jobs provide a way to run a single Pod once until successful termination. Pod is restarted in case of failure

```
$ kubectl run -it oneshot --image=gcr.io/kuar-demo/kuard-amd64:1 --restart=OnFailure -
↪- --keygen-enable --keygen-exit-on-complete --keygen-num-to-gen 5
```

List all jobs

```
$ kubectl get jobs -o wide
```

Delete job

```
$ kubectl delete jobs oneshot
```

Show one-shot Job configuration file

```
$ tee files/job-oneshot.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: oneshot
  labels:
    chapter: jobs
spec:
  template:
    metadata:
      labels:
        chapter: jobs
    spec:
      containers:
      - name: kuard
        image: gcr.io/kuar-demo/kuard-amd64:1
        imagePullPolicy: Always
        args:
        - "--keygen-enable"
```

(continues on next page)

(continued from previous page)

```
        - "--keygen-exit-on-complete"
        - "--keygen-num-to-gen=5"
      restartPolicy: OnFailure
EOF
```

Create one-shot Job using a configuration file

```
$ kubectl apply -f files/job-oneshot.yaml
$ sleep 30
```

Print details about the job

```
$ kubectl describe jobs oneshot
```

Get pod name of a job called 'oneshot' and check the logs

```
$ POD_NAME=$(kubectl get pods --selector="job-name=oneshot" -o=jsonpath="{.items[0].
↪metadata.name}")
$ kubectl logs ${POD_NAME}
```

Remove job oneshot

```
$ kubectl delete jobs oneshot
```

Show one-shot Job configuration file. See the keygen-exit-code parameter - nonzero exit code after generating three keys

```
$ tee files/job-oneshot-failure1.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: oneshot
  labels:
    chapter: jobs
spec:
  template:
    metadata:
      labels:
        chapter: jobs
    spec:
      containers:
      - name: kuard
        image: gcr.io/kuar-demo/kuard-amd64:1
        imagePullPolicy: Always
        args:
        - "--keygen-enable"
        - "--keygen-exit-on-complete"
        - "--keygen-exit-code=1"
        - "--keygen-num-to-gen=3"
      restartPolicy: OnFailure
EOF
```

Create one-shot Job using a configuration file

```
$ kubectl apply -f files/job-oneshot-failure1.yaml
$ sleep 60
```

Get pod status - look for CrashLoopBackOff/Error indicating pod restarts

```
$ kubectl get pod -l job-name=oneshot
```

Remove the job

```
$ kubectl delete jobs oneshot
```

Show Parallel Job configuration file - generate (5x10) keys generated in 5 containers

```
$ tee files/job-parallel.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  name: parallel
  labels:
    chapter: jobs
spec:
  # 5 pods simlutaneously
  parallelism: 5
  # repeat task 10 times
  completions: 10
  template:
    metadata:
      labels:
        chapter: jobs
    spec:
      containers:
      - name: kuard
        image: gcr.io/kuar-demo/kuard-amd64:1
        imagePullPolicy: Always
        args:
        - "--keygen-enable"
        - "--keygen-exit-on-complete"
        - "--keygen-num-to-gen=5"
      restartPolicy: OnFailure
EOF
```

Create Parallel Job using a configuration file

```
$ kubectl apply -f files/job-parallel.yaml
```

Check the pods and list changes as they happen

```
$ kubectl get pods --watch -o wide &
$ sleep 10
```

Stop getting the pods

```
$ pkill -f "kubectl get pods --watch -o wide"
```

Remove the job

```
$ kubectl delete jobs parallel
```

9.1 Queue job example

Memory-based work queue system: Producer -> Work Queue -> Consumers diagram

```
$ tee /tmp/producer_queue_consumer-diagram.txt << EOF
                                                    +--------------+
                                                    |              |
                                                +-> |   Consumer   |
                                                |   |              |
                                                |   +--------------+
                                                |
+--------------+          +-----------------+   |   +--------------+
|              |          |                 |   |   |              |
|   Producer   | +------> |   Work Queue    | +--+-> |   Consumer   |
|              |          |                 |   |   |              |
+--------------+          +-----------------+   |   +--------------+
                                                |
                                                |   +--------------+
                                                |   |              |
                                                +-> |   Consumer   |
                                                    |              |
                                                    +--------------+
EOF
```

Create a simple ReplicaSet to manage a singleton work queue daemon

```
$ tee files/rs-queue.yaml << EOF
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
  labels:
    app: work-queue
    component: queue
    chapter: jobs
  name: queue
spec:
  replicas: 1
  selector:
    matchLabels:
      app: work-queue
      component: queue
      chapter: jobs
  template:
    metadata:
      labels:
        app: work-queue
        component: queue
        chapter: jobs
    spec:
      containers:
      - name: queue
        image: "gcr.io/kuar-demo/kuard-amd64:1"
        imagePullPolicy: Always
EOF
```

Create work queue using a configuration file

```
$ kubectl apply -f files/rs-queue.yaml
$ sleep 30
```

Configure port forwarding to connect to the 'work queue daemon' pod

```
$ QUEUE_POD=$(kubectl get pods -l app=work-queue,component=queue -o jsonpath="{.
↪items[0].metadata.name}")
$ kubectl port-forward $QUEUE_POD 8080:8080 &
```

Expose work queue - this helps consumers+producers to locate the work queue via DNS

```
$ tee files/service-queue.yaml << EOF
apiVersion: v1
kind: Service
metadata:
  labels:
    app: work-queue
    component: queue
    chapter: jobs
  name: queue
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: work-queue
    component: queue
EOF
```

Create the service pod using a configuration file

```
$ kubectl apply -f files/service-queue.yaml
$ sleep 20
```

Create a work queue called 'keygen'

```
$ curl -X PUT 127.0.0.1:8080/memq/server/queues/keygen
```

Create work items and load up the queue

```
$ for WORK in work-item-{0..20}; do curl -X POST 127.0.0.1:8080/memq/server/queues/
↪keygen/enqueue -d "$WORK"; done
```

Queue should not be empty - check the queue by looking at the 'MemQ Server' tab in Web interface (http://127.0.0.1:8080/-/memq)

```
$ curl --silent 127.0.0.1:8080/memq/server/stats | jq
```

Show consumer job config file allowing start up five pods in parallel. Once the first pod exits with a zero exit code, the Job will not start any new pods (none of the workers should exit until the work is done)

```
$ tee files/job-consumers.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  labels:
```

(continues on next page)

(continued from previous page)

```
    app: message-queue
    component: consumer
    chapter: jobs
  name: consumers
spec:
  parallelism: 5
  template:
    metadata:
      labels:
        app: message-queue
        component: consumer
        chapter: jobs
    spec:
      containers:
      - name: worker
        image: "gcr.io/kuar-demo/kuard-amd64:1"
        imagePullPolicy: Always
        args:
        - "--keygen-enable"
        - "--keygen-exit-on-complete"
        - "--keygen-memq-server=http://queue:8080/memq/server"
        - "--keygen-memq-queue=keygen"
      restartPolicy: OnFailure
EOF
```

Create consumer job from config file

```
$ kubectl apply -f files/job-consumers.yaml
$ sleep 30
```

Five pods should be created to run until the work queue is empty. Open the web browser to see changing queue status (http://127.0.0.1:8080/-/memq)

```
$ kubectl get pods -o wide
```

Check the queue status - especially the 'dequeued' and 'depth' fields

```
$ curl --silent 127.0.0.1:8080/memq/server/stats | jq
```

Stop port-forwarding

```
$ pkill -f "kubectl port-forward $QUEUE_POD 8080:8080"
```

Clear the resources

```
$ kubectl delete rs,svc,job -l chapter=jobs
```

CHAPTER 10

ConfigMaps

Show file with key/value pairs which will be available to the pod

```
$ tee files/my-config.txt << EOF
# This is a sample config file that I might use to configure an application
parameter1 = value1
parameter2 = value2
EOF
```

Create a ConfigMap with that file (environment variables are specified with a special valueFrom member)

```
$ kubectl create configmap my-config --from-file=files/my-config.txt --from-
↪literal=extra-param=extra-value --from-literal=another-param=another-value
```

Show ConfigMaps

```
$ kubectl get configmaps
```

Show ConfigMap details

```
$ kubectl describe configmap my-config
```

See the YAML ConfigMap object

```
$ kubectl get configmaps my-config -o yaml
```

Prepare config file for ConfigMap usage

```
$ tee files/kuard-config.yaml << \EOF
apiVersion: v1
kind: Pod
metadata:
  name: kuard-config
spec:
  containers:
```

(continues on next page)

(continued from previous page)

```
    - name: test-container
      image: gcr.io/kuar-demo/kuard-amd64:1
      imagePullPolicy: Always
      command:
        - "/kuard"
        - "$(EXTRA_PARAM)"
      env:
        - name: ANOTHER_PARAM
          valueFrom:
            configMapKeyRef:
              name: my-config
              key: another-param
        # Define the environment variable
        - name: EXTRA_PARAM
          valueFrom:
            configMapKeyRef:
              # The ConfigMap containing the value you want to assign to ANOTHER_PARAM
              name: my-config
              # Specify the key associated with the value
              key: extra-param
      volumeMounts:
        - name: config-volume
          mountPath: /config
  volumes:
    - name: config-volume
      configMap:
        name: my-config
  restartPolicy: Never
EOF
```

Apply the config file

```
$ kubectl apply -f files/kuard-config.yaml
$ sleep 20
```

{EXTRA_PARAM,ANOTHER_PARAM} variable has value from configmap my-config/{extra-param,another-param} and file /config/my-config.txt exists in container

```
$ kubectl exec kuard-config -- sh -xc "echo EXTRA_PARAM: \$EXTRA_PARAM; echo ANOTHER_
↪PARAM: \$ANOTHER_PARAM && cat /config/my-config.txt"
```

Go to http://localhost:8080 and click on the 'Server Env' tab, then 'File system browser' tab (/config) and look for ANOTHER_PARAM and EXTRA_PARAM values

```
$ kubectl port-forward kuard-config 8080:8080 &
```

Stop port forwarding

```
$ pkill -f "kubectl port-forward kuard-config 8080:8080"
```

Remove pod"

```
$ kubectl delete pod kuard-config
```

CHAPTER 11

Secrets

Download certificates

```
$ wget -q -c -P files https://storage.googleapis.com/kuar-demo/kuard.crt https://
↪storage.googleapis.com/kuar-demo/kuard.key
```

Create a secret named kuard-tls

```
$ kubectl create secret generic kuard-tls --from-file=files/kuard.crt --from-
↪file=files/kuard.key
```

Get details about created secret

```
$ kubectl describe secrets kuard-tls
```

Show secrets

```
$ kubectl get secrets
```

Update secrets - generate yaml and then edit the secret 'kubectl edit configmap my-config'

```
$ kubectl create secret generic kuard-tls --from-file=files/kuard.crt --from-
↪file=files/kuard.key --dry-run -o yaml | kubectl replace -f -
```

Create a new pod with secret attached

```
$ tee files/kuard-secret.yaml << EOF
apiVersion: v1
kind: Pod
metadata:
  name: kuard-tls
spec:
  containers:
    - name: kuard-tls
      image: gcr.io/kuar-demo/kuard-amd64:1
```

(continues on next page)

(continued from previous page)

```
      imagePullPolicy: Always
      volumeMounts:
      - name: tls-certs
        mountPath: "/tls"
        readOnly: true
  volumes:
    - name: tls-certs
      secret:
        secretName: kuard-tls
EOF
```

Apply the config file

```
$ kubectl apply -f files/kuard-secret.yaml
$ sleep 20
```

Set port-forwarding. Go to https://localhost:8080, check the certificate and click on "File system browser" tab (/tls)

```
$ kubectl port-forward kuard-tls 8443:8443 &
```

Stop port forwarding

```
$ pkill -f "kubectl port-forward kuard-tls 8443:8443"
```

Delete pod

```
$ kubectl delete pod kuard-tls
```

CHAPTER 12

Deployments

Show nginx deployment definition

```
$ tee files/nginx-deployment.yaml << EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
EOF
```

Create nginx deployment

```
$ kubectl create -f files/nginx-deployment.yaml
```

List deployments

```
$ kubectl get deployments -o wide
```

Get deployment details

```
$ kubectl describe deployment nginx-deployment
```

Show deployment YAML file (look for: 'nginx:1.7.9')

```
$ kubectl get deployment nginx-deployment -o wide
```

Change deployment image (version 1.7.9 -> 1.8) - you can do the change by running 'kubectl edit deployment nginx-deployment' too...

```
$ kubectl set image deployment nginx-deployment nginx=nginx:1.8
```

See what is happening during the deployment change

```
$ kubectl rollout status deployment nginx-deployment
```

Get deployment details (see: 'nginx:1.8')

```
$ kubectl get deployment nginx-deployment -o wide
```

Show details for deployment

```
$ kubectl describe deployment nginx-deployment
```

See the deployment history (first there was version nginx:1.7.9, then nginx:1.8)

```
$ kubectl rollout history deployment nginx-deployment --revision=1
$ kubectl rollout history deployment nginx-deployment --revision=2
```

Rollback the deployment to previous version (1.7.9)

```
$ kubectl rollout undo deployment nginx-deployment
$ kubectl rollout status deployment nginx-deployment
```

Get deployment details - see the image is now again 'nginx:1.7.9'

```
$ kubectl get deployment nginx-deployment -o wide
```

Rollback the deployment back to version (1.8)

```
$ kubectl rollout undo deployment nginx-deployment --to-revision=2
$ kubectl rollout status deployment nginx-deployment
```

Get deployment details - see the image is now again 'nginx:1.8'

```
$ kubectl get deployment nginx-deployment -o wide
```

Check the utilization of pods

```
$ kubectl top pod --heapster-namespace=myns --all-namespaces --containers
```

CHAPTER 13

Endpoints

Show external service DNS definition

```
$ tee files/dns-service.yaml << EOF
kind: Service
apiVersion: v1
metadata:
  name: external-database
spec:
  type: ExternalName
  externalName: database.company.com
EOF
```

Create DNS name (CNAME) that points to the specific server running the database

```
$ kubectl create -f files/dns-service.yaml
```

Show services

```
$ kubectl get service
```

Remove service

```
$ kubectl delete service external-database
```

CHAPTER 14

Self-Healing

Get pod details

```
$ kubectl get pods -o wide
```

Get first nginx pod and delete it - one of the nginx pods should be in 'Terminating' status

```
$ NGINX_POD=$(kubectl get pods -l app=nginx --output=jsonpath="{.items[0].metadata.
↪name}")
$ kubectl delete pod $NGINX_POD; kubectl get pods -l app=nginx -o wide
$ sleep 10
```

Get pod details - one nginx pod should be freshly started

```
$ kubectl get pods -l app=nginx -o wide
```

Get deployement details and check the events for recent changes

```
$ kubectl describe deployment nginx-deployment
```

Halt one of the nodes (node2)

```
$ vagrant halt node2
$ sleep 30
```

Get node details - node2 Status=NotReady

```
$ kubectl get nodes
```

Get pod details - everything looks fine - you need to wait 5 minutes

```
$ kubectl get pods -o wide
```

Pod will not be evicted until it is 5 minutes old - (see Tolerations in 'describe pod'). It prevents Kubernetes to spin up the new containers when it is not necessary

```
$ NGINX_POD=$(kubectl get pods -l app=nginx --output=jsonpath="{.items[0].metadata.
↪name}")
$ kubectl describe pod $NGINX_POD | grep -A1 Tolerations
```

Sleeping for 5 minutes

```
$ sleep 300
```

Get pods details - Status=Unknown/NodeLost and new container was started

```
$ kubectl get pods -o wide
```

Get depoyment details - again AVAILABLE=3/3

```
$ kubectl get deployments -o wide
```

Power on the node2 node

```
$ vagrant up node2
$ sleep 70
```

Get node details - node2 should be Ready again

```
$ kubectl get nodes
```

Get pods details - 'Unknown' pods were removed

```
$ kubectl get pods -o wide
```

CHAPTER 15

Persistent Storage

Install and configure NFS on node1

```
$ ssh $SSH_ARGS vagrant@node1 "sudo sh -xc \" apt-get update -qq; DEBIAN_
↪FRONTEND=noninteractive apt-get install -y nfs-kernel-server > /dev/null; mkdir /
↪nfs; chown nobody:nogroup /nfs; echo /nfs *\(rw,sync,no_subtree_check\) >> /etc/
↪exports; systemctl restart nfs-kernel-server \""
```

Install NFS client to other nodes

```
$ for COUNT in {2..4}; do ssh $SSH_ARGS vagrant@node${COUNT} "sudo sh -xc \"apt-get␣
↪update -qq; DEBIAN_FRONTEND=noninteractive apt-get install -y nfs-common > /dev/
↪null\""; done
```

Show persistent volume object definition

```
$ tee files/nfs-volume.yaml << EOF
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
  labels:
    volume: nfs-volume
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 1Gi
  nfs:
    server: node1
    path: "/nfs"
EOF
```

Create persistent volume

```
$ kubectl create -f files/nfs-volume.yaml
```

Check persistent volumes

```
$ kubectl get persistentvolume
```

Show persistent volume claim object definition

```
$ tee files/nfs-volume-claim.yaml << EOF
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      volume: nfs-volume
EOF
```

Claim the persistent volume for our pod

```
$ kubectl create -f files/nfs-volume-claim.yaml
```

Check persistent volume claims

```
$ kubectl get persistentvolumeclaim
```

Show replicaset definition

```
$ tee files/nfs-test-replicaset.yaml << EOF
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nfs-test
  # labels so that we can bind a service to this pod
  labels:
    app: nfs-test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nfs-test
  template:
    metadata:
      labels:
        app: nfs-test
    spec:
      containers:
      - name: nfs-test
        image: busybox
        command: [ 'sh', '-c', 'date >> /tmp/date && sleep 3600' ]
        volumeMounts:
```

(continues on next page)

(continued from previous page)

```
          - name: nfs-test
            mountPath: "/tmp"
      volumes:
      - name: nfs-test
        persistentVolumeClaim:
          claimName: nfs-pvc
      securityContext:
        runAsUser: 65534
        fsGroup: 65534
EOF
```

Create replicaset

```
$ kubectl create -f files/nfs-test-replicaset.yaml
$ sleep 20
```

You can see the /tmp is mounted to both pods containing the same file 'date'

```
$ NFS_TEST_POD2=$(kubectl get pods --no-headers -l app=nfs-test -o custom-
↪columns=NAME:.metadata.name | head -1); echo $NFS_TEST_POD2
$ NFS_TEST_POD1=$(kubectl get pods --no-headers -l app=nfs-test -o custom-
↪columns=NAME:.metadata.name | tail -1); echo $NFS_TEST_POD1
$ kubectl exec -it $NFS_TEST_POD1 -- sh -xc "hostname; echo $NFS_TEST_POD1 >> /tmp/
↪date"
$ kubectl exec -it $NFS_TEST_POD2 -- sh -xc "hostname; echo $NFS_TEST_POD2 >> /tmp/
↪date"
```

Show files on NFS server - there should be 'nfs/date' file with 2 dates

```
$ ssh $SSH_ARGS vagrant@node1 "set -x; ls -al /nfs -ls; ls -n /nfs; cat /nfs/date"
```

CHAPTER 16

Node replacement

Move all pods away from node3

```
$ kubectl drain --delete-local-data --ignore-daemonsets node3
```

Get pod details

```
$ kubectl get pods -o wide --all-namespaces | grep node3
```

Destroy the node node3

```
$ vagrant destroy -f node3
```

Wait some time for Kubernetes to catch up...

```
$ sleep 40
```

The node3 shoult be in 'NotReady' state

```
$ kubectl get pods -o wide --all-namespaces
```

Remove the node3 from the cluster

```
$ kubectl delete node node3
```

Generate command which can add new node to Kubernetes cluster

```
$ KUBERNETES_JOIN_CMD=$(ssh $SSH_ARGS root@node1 "kubeadm token create --print-join-
↪command"); echo $KUBERNETES_JOIN_CMD
```

Start new node

```
$ vagrant up node3
```

Install Kubernetes repository to new node

```
$ ssh $SSH_ARGS vagrant@node3 "sudo sh -xc \" apt-get update -qq; DEBIAN_
↪FRONTEND=noninteractive apt-get install -y apt-transport-https curl > /dev/null;␣
↪curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -; echo␣
↪deb https://apt.kubernetes.io/ kubernetes-xenial main > /etc/apt/sources.list.d/
↪kubernetes.list \""
```

Install Kubernetes packages

```
$ ssh $SSH_ARGS vagrant@node3 "sudo sh -xc \" apt-get update -qq; DEBIAN_
↪FRONTEND=noninteractive apt-get install -y docker.io kubelet=${KUBERNETES_VERSION}-
↪00 kubeadm=${KUBERNETES_VERSION}-00 kubectl=${KUBERNETES_VERSION}-00 > /dev/null \""
```

Join node3 to the Kuberenets cluster

```
$ ssh $SSH_ARGS vagrant@node3 "sudo sh -xc \" $KUBERNETES_JOIN_CMD \""
$ sleep 40
```

Check the nodes - node3 should be there

```
$ kubectl get nodes
```

CHAPTER 17

Notes

Show logs from specific docker container inside pod

```
$ kubectl logs --namespace=kube-system $(kubectl get pods -n kube-system -l k8s-
↪app=kube-dns -o name) --container=dnsmasq --tail=10
$ kubectl logs --namespace=kube-system $(kubectl get pods -n kube-system -l k8s-
↪app=kube-dns -o name) --container=kubedns --tail=10
```

See the logs directly on the Kubernetes node

```
$ ssh $SSH_ARGS vagrant@node1 "ls /var/log/containers/"
```

Show all

```
$ kubectl get all
```