Kubernetes

page 1 of 2

1 Templating

Helm

Template-based package mgr for k8s. “Charts” are stored in a “repository.” Eg, ArtifactHub, a public analog to DockerHub. ∃ private repos tools as well, such as ChartMuseum. Upon `helm install`*, templates are resolved into k8s manifests & sent to the API server. CLI subcmds:*

completion get package rollback template verify
create history plugin search test version
dependency install pull show uninstall
env lint repo status upgrade

Kustomize

Preferred by k8s developers (and adopted into `kubectl`*-proper) as a light-weight alternative to templates; kustomize is itself a k8s resource (per KRM) that layers k8s yaml Δs as code into subdirectories near the original manifests. View prospective Δs with* `kustomize build` *or apply them directly with* `kubectl apply -k`.

2 Docker

Images are identified by their tag, which generally looks like: <ns>/<version>. For gcloud:

```
us-central1-docker.pkg.dev/
<prjID>/<repoID>/<img>:<tag>
```

CLI

```
docker run <tag>                # local execution
docker build -t <name> .        # from Dockerfile
docker images                   # local imgs
docker ps                       # running cntrs
docker inspect <cntr>           # detailed info
docker exec -it <cntr> bash     # bash in
docker stop <cntr>              # stop execut’n
docker rm <cntr>                # fully remove
docker tag <img> <tag>          # identify
docker push <tag>               # upload to repo
docker login                    # login to repo
docker commit                   # cntr → cntr
docker logs                     #
```

Dockerfile

```
FROM ubuntu:18.04 # initial image
COPY . /app       # move files in/out
RUN make /app     # configure
CMD python /app/app.py  # main command
```

Repositories

3 kubectl

Common Flags

```
    --all-namespaces      #
    --namespace <name>#
    --field-selector <spec#
    --tail=<#>            #
    -o <style>            # output, eg wide
    -f <file>             # applys, etc
    -c <cntr>             # specific cntr
    --dry-run             # see test yaml
    --pretty              # output format
    --watch               # live Δs
    --selector            # label selector
```

Basics

```
kubectl create <>          #
kubectl get <>             #
kubectl run <>             #
kubectl expose <>          #
kubectl delete <>          #
kubectl exec --stdin --tty demo.yaml  # shell
kubectl run <name> --image=<img> # image
```

Application Deployment

```
kubectl apply <>           #
kubectl annotate <>        #
kubectl autoscale <>       #
kubectl debug <>           #
kubectl diff <>            #
kubectl edit <>            #
kubectl kustomize <>       #
kubectl label <>           #
kubectl patch <>           #
kubectl replace <>         #
kubectl rollout <>         #
kubectl scale <>           #
kubectl set <>             #
kubectl wait <>            #
```

Debugging / Inspection

```
kubectl attach <>          #
kubectl auth <>            #
kubectl cp <>              #
kubectl describe <>        #
kubectl exec <>            #
kubectl logs <>            #
kubectl port-forward <>    #
kubectl proxy <>           #
kubectl top <>             #
```

Cluster Management

```
kubectl api-versions <>    #
kubectl certificate <>     #
kubectl cluster-info <>    #
kubectl cordon <>          #
kubectl drain <>           #
kubectl taint <>           #
kubectl uncordon <>        #
```

Kubectl Settings

```
kubectl alpha <>           #
kubectl api-resources <>#
kubectl completion <>      #
kubectl config <>          #
kubectl explain <>         #
kubectl options <>         #
kubectl plugin <>          #
kubectl version <>         #
```

4 Namespaces

Namespace

`spec` *(namespace spec)*

finalizers

`status` *(namespace status)*

conditions phase

`status.conditions` *(latest observations)*

status type lastTrans’tnTime
message reason

LimitRange

`spec` *(min/max resource usage)*

limits

`spec.limits` *(list of limit objs)*

type default def’tReq’st
max maxLim.Req.Ratio min

ResourceQuota

`spec` *(aggregate hard limits per ns)*

hard scopeSelector scopes

`status` *(RQs enforce)*

hard used

5 Workloads

Pod

`spec` *(description of a pod)*

containers initContainers imagePullSec’ts
affinity enableS’vcLinks nodeSelector
nodeName runt’mClassN’me tolerations
schedulerName topo.SpreadConst. prio.ClassName
volumes act.DeadlineSecs restartPolicy
priority setHostn.AsFQDN readinessGates
hostname term.GracePer.Secs subdomain
dnsConfig dnsPolicy hostNetwork
hostPid hostIPC shareProc.Names.
svcAcc’tName autom’dS.A.Token securityCont’xt

`spec.container` *(to run in a pod)*

name image im’gPullPolicy
command args workingDir
ports env envFrom
volumeMounts vol.Devices resources
lifecycle term.M’sgPath term.M’sgPolicy
livenessProbe readinessProbe startupProbe
securityContext stdin[Once] tty

`spec.container.port` *()*

containerPort hostIP hostPort
name protocol

`spec.container.env` *(vars & their source)*

name value valueFrom

`spec.container.volumeMounts` *(into cntr)*

mountpath m’ntPropagt’n subPath
name readOnly subPathExpr

`spec.container.resources` *(guidelines)*

limits requests

`spec.container.lifecycle` *(actions)*

postStart preStart

`spec.container.securityContext` *(config)*

runAsUser runAsNonRoot runAsGroup
readOnlyF.S. procMount priviledged
allowPriv.Escal’n capabilities seccompProfile
seLinuxOptions windowsOpt’ns

`[Handler]` *(actions to be taken)*

exec httpGet tcpSocket

`[Handler.httpGet]` *(request spec)*

port host httpHeaders
path scheme

`[[Node|Pod|PodAnti]Affinity]` *(schedul’g)*

preferredDuringSchedulingIgnoredDuringExecution
requiredDuringSchedulingIgnoredDuringExecution

`probe` *(aliveness health check)*

exec init’lDelaySec’s timeoutSec’s
httpGet term.GracePer.Sec’s periodSec’s
tcpSocket failureThresh. successThresh.

`status` *(info about a pod)*

nom’ntedNodeNamephase startTime
message podIP[s] reason
containerStatuses hostIP qosClass
initCont’rStatuses conditions

`status.[init|ephemeral]containerStatuses`

name image imageID
containerID state lastState
ready restartCount started

ReplicaSet

Replaces deprecated “ReplicationController;” frequently generated by `Deployments` *(rather than being created directly).*

`spec` *(rc specification)*

selector template replicas
minReadySec’s

`status` *()*

replicas avail.Replicas fullyLabeledRep’s
readyReplicas conditions observedGenerat’n

Job

`spec` *(Job specification)*

template parallelism completions
compl’nMode backoffLim’t act’vDeadl’nSecs
ttlSecsAftFins’d suspend selector
man’lSelector

`status` *(job’s current state)*

startTime compt’nTime active
failed succeeded complt’dIndexes
conditions uncountedTerm’dPods

`status.conditions` *(latest observed state)*

status type lastProbeTime
message reason lastTrans’nTime

CronJob

`spec` *(CJ specification)*

jobTemplate schedule startingDeadlineSecond
concurrencyPolicy suspend success’lJobsHist’yLimit

`status` *(CJ current status)*

active lastScheduleTime lastSuccessfulTime

DaemonSet

`spec` *(DS specification)*

selector template minReadySec’s
updateStrat. revis’nHist’yLim’t

`status` *(state of DS)*

conditions num’rAvail’ble curr.Num’rSched.
num’rReady collisionCount updatedNum’rSched.
num’rUnavail.num’rMissched. observedGenerat’n
desiredNum’rSche.

StatefulSet

`spec` *(SS specification)*

serviceName replicas template
revis’nHist.Lim. selector podM’gmtPolic.
vol.ClaimTempl’ts updateStrat. minReadySec’s

`spec.updateStrategy` *(for updating pods)*

type rollingUpdate

`status` *(current SS state)*

currentReplicas replicas readyReplicas
updatedReplicas conditions collisionCount
obs’vdGeneration avail.Replicas currentRevision
updatedRevision

Deployment

`spec` *(desired deployment behavior)*
selector template replicas
minReadySeconds strategy revis'nHist.Limit
prog.Deadl'nSec's paused

`spec.strategy` *(for replacing pods)*
type rollingUpdate

`status` *(most recently observed)*
replicas avail.Replicas readyReplicas
unavail.Replicas updatedReplicas collisionCount
conditions obs'vdGeneration

6 Configuration

ConfigMap

`[ConfigMap]` *(additional top-level keys)*
binaryData data immutable

Secret

`[Secret]` *(additional top-level keys)*
stringData data type immutable

7 Network

Services expose pods behind 2 layers of indirect'n: 1) a Service url, which `kube-dns` resolves to a singular svc IP; 2) an EndPoints rsc that maps the svc IP to a list of pod IPs. Ingresses (L7) and LoadBalancers (L4) can expose svcs externally by redirecting requests made against a given public IP.

Often in the below, IPs & ports contain:

`[IPAddress]` *(description of IP)*
ip hostname nodeName targetRef

`[Ports]` *(list of ports)*
port protocol name appProtocol

Service

`spec` *(attributes for creating)*
selector loadBal.S'rcRanges type
ipFamilies pub.NotReadyAddr's clusterIP[s]
externalIPs h'lthCh'kNodeP'nt loadBal.IP
ports alloc.L.B.N'deP'rts exten.Name
ipFam'yPolicy internalTrafficPol. sessionAffin.
extern.Traff.Pol. sess.Affin.Config loadBal.Class

`status` *(current service status)*
conditions loadBalancer loadBal'r.ingress

Endpoint

IPs → svcs; often generated indirectly by svcs.

`subsets` *(set of IPs, ports in a service)*
addresses notReadyAddresses ports

Ingress

`spec` *(desired ingress)*
defaultBackend ingressClassName rules tls

`spec.defaultBackend` *(how it is specified)*
resource service

`spec.rules` *(traffic routing)*
host http

`spec.tls` *(tls configuration)*
hosts secretName

`status.loadBalancer.ingress` *(status)*
hostname ip ports

IngressClass

`spec` *(ref → ingress controller)*
controller parameters

`spec.parameters` *(addit'nal ctrllr config)*
kind name apiGroup
scope namespace

NetworkPolicy

`spec` *(allowed traffic from/to pods: default open)*
podSelector policyTypes ingress egress

`spec.ingress` *(inbound rules whitelisting)*
from ports

`spec.egress` *(outbound rules whitelisting)*
to ports

`spec.[in|e]gress.[from|to]` *(rule src/dest)*
ipBlock n'mspceSelector podSelector

Nginx Ingress Controller

∄ a default ingress controller, so, to install nginx:

kubectl apply -f URL/deploy.yaml

Extensive nginx controller config fields here.

To expose a service, create a normal ingress resource (as above) with an ingress class (shown here) to specify which ingress ctllr to use. (Or just assume the default.) Customize the ingress' behavior with these nginx-specific annotations.

8 Security

ServiceAccount

`[TopLevelFields]` *(additional top level fields)*
autom'ntS.A.Token imagePullSecrets secrets

[Cluster]Role

`rules` *(top-level set of PolicyRules)*
apiGroups resources verbs
r'srcNames nonR'srcURLs

[Cluster]RoleBinding

`roleRef` *(reference → role)*
apiGroup kind name

`subjects` *(ref → user ID)*
apiGroup kind name namespace

CertificateSigningRequest

`spec` *(cert request)*
request signerName expir'tnSec's
extra groups uid
usages username

`status` *(result of request)*
certificate conditions

`status.conditions` *(reasons for rejection/etc)*
status type lastTran'tnTime
message reason lastUpd'teTime

TokenRequest

`spec` *(client parameters of request)*
audiences boundObj.Ref expirationSec's

`spec.boundObjectRef` *(token applies to this)*
apiVersion kind name uid

`status` *()*
expirationTimestamp token

TokenReview

`spec` *(attempt to authenticate)*
audiences token

`status` *(result of request)*
audiences authentication error user

9 Storage

PersistentVolume

`spec` *(spec of PV)*
accessModes capacity claimRef
m'ntOptions nodeAffinity P.V.ReclaimPol.
stor.ClassName volumeMode hostPath
local

∃ many types of PVs, eg `gcePersistentDisk`.

`status` *(current status of volume)*
message phase reason

PersistentVolumeClaim

`spec` *()*
accessModes selector resources
volumeName stor.ClassName volumeMode
dataSource dataSourceRef

`status` *(current status, PVC)*
accessModes capacity conditions phase

StorageClass

`[TopLevel]` *(additional top-level fields)*
provisioner allowVol.Expansion all'dTopologies
mountOptions parameters reclaimPolicy
vol.BindingMode

10 Cluster

HorizontalPodAutoscaler

`spec` *(HPA spec)*
maxReplicas minReplicas scaleTargetRef
targ.CPUUtil.%

`status` *()*
currentReplicas desiredReplicas curr.CPUUtil.%
lastScaleTime obs'vdGeneration

PodDisruptionBudget

`spec` *()*
maxUnavail. minAvail. selector

`status` *(PDB status)*
currentHealthy desiredHealthy disruptionsAll'd
expectedPods conditions disruptedPods
obs'vdGeneration

Node

`spec` *()*
configSource externalID podCIDR[s]
providerID taints unschedulable

`taints` *(affecting all pods on node)*
effect key timeAdded value

`status` *(current node status)*
addresses allocatable capacity
conditions config daemonEndP'ts
images nodeInfo phase
vol'sAttached vol'sInUse

ComponentStatus

`conditions` *(comp conditions observed)*
status type error message

Event

`[TopLevel]` *(additional top-level fields)*
eventTime action deprecatedCount
dep.FirstT'stmp dep.Source reason
dep.LastT'stmp note regarding
report'gC'trllr relate rept'gInstance
series type

11 Kubernetes Internals

kubeadm

To build your own k8s cluster, install node machines / OSes, with relevant network config and container runtimes, then run:

kubeadm init # run on master node

Controllers (& k8s "Control Plane")

These are a bevy of subscribers to API server, each listening for events of interest. They affect cluster Δs by writing back to API server. Egs:
Repl'n Mgr ReplicaSet C. DaemonSet C.
StatefulSet C. Node C. Deployment C.
Job C. Service C. Endpoints C.
Namespace C. Persist.Vol. C.

Scheduler

Assigns pod → node by Δ-ing pod definition thru API server, which then notifies `kubelet` of the Δ.

kubelet

Monitors API server for pods assigned to the local node; then starts new pods' containers by invoking the cluster's configured container runtime (most often Docker).

kube-dns

A pod that ∃ (∈ `kube-system`) to resolve cluster service URLs to service IPs ∀ pods ∈ the cluster. DNS inquiries are routed here by default; each each pod's `/etc/resolv.conf` file points to `kube-dns`.

kube-proxy

Each node has a `kube-proxy` that writes `EndPoints` contents to its node's `iptables` file.

API

API adheres to REST principals, employing HTTP (or protobufs) to transfer data. REST's "resources" map to k8s "objects" (eg, pods, namespaces, etc) via its "resource" URL. `kubectl` CLI wraps HTTP calls. Most/all resources have the following:

`[top level]` *(top level fields)*
apiVersion kind metadata spec status

`[resource].metadata` *(generic, ∀ resource)*
name generateName namespace
labels annotations finalizers
manag'dFields ownerRef's creat'nTim'p
del.Time'p del.Gr'cPer.Sec's generation
rs'ceVersion selfLink uid

`[<rsrcetype>List]` *(collect'n, <rsrcetype>s)*
apiVersion kind metadata items

Eg, for a ChronJob (other fields are obj-specific):

apiVersion: batch/v1; kind: ChronJob