Kubernetes

0. Terminology

- container runtime
- kubelet

1. Docker Basics

Steps - create docker image - docker build -t quickstart-image . - docker tag quickstart-image gcr.io/[PROJECT-ID]/quickstart-image:tag1 - docker push gcr.io/[PROJECT-ID]/quickstart-image:tag1 - docker pull gcr.io/[PROJECT-ID]/quickstart-image:tag1 - gcloud container images delete gcr.io/[PROJECT-ID]/quickstart-image:tag1 --force-delete-tags - kubernete yaml file

```
 # docker file syntax

FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]

ENTRYPOINT: set image's main command
CMD
ENV: provide environment variables
ARG
RUN
```

how to speed up imgae build? troubleshooting

```
 # docker-compose cannot kill network
docker network inspect <network_name>
docker network disconnect -f <network_name> <container_name>
```

2. Core Concepts

Goals: - Understand Kubernetes API primitives - Create and configure basic Pods

Components - api server - etcd - scheduler - kubelet - controller - container runtime

2.1 Workloads

2.1.1 Pod

the smallest unit, can have more than one containers running

```
 # kubtctl cmd

kubectl run <pod_name> --image <image_name>

kubectl create -f example.yaml

kubectl get pods -n <namespace>
kubectl get pod <pod-name> -o yaml > pod-definition.yaml  # get pod definition file
kubectl edit pod <pod-name>
kubectl get replicationcontroller

# use selector to retrieve info from matched pods

kubectl get pods -l environment=production,tier=frontend # equality-based
kubectl get pods -l 'environment in (production),tier in (frontend)' # set-based
kubectl get pod --sort-by=.metadata.creationTimestamp -n dbt-tactical-ds -o name
```

2.1.2 ReplicaSet

```
...
kind: ReplicationController
metadata:
  name:
  label:
    l_name: l_value
spec:
  template:
    <pod definition>
  replicas: 2

# replicationSet
kind: ReplicaSet
selector:
```

```
# how to scale a replicaSet?

kubectl replace -f .yaml
kubectl scale --replicas=6 -f .yaml
kubectl config set-context $(kubectl config current-context) --namespace=dev
```

2.1.3 Deployment

```
kind: Deployment
```

2.1.4 namespace

```
kind: namespaces
```

2.1.5 Service

```
kind: service
```

2.1.6 Imperative command

```
kubectl run --generator=run-pod/v1 nginx-pod --image=nginx:alpine

# expose servian from a pod
kubectl expose pod redis --port=6379 --name redis-service

# create deployment and scale
kubectl create deployment webapp --image=kodekloud/webapp-color
kubectl scale deployment/webapp --replicas=3

# flag
-o output
-l label
```

2.2 Configuration

Goals: - Understand ConfigMaps - Understand SecurityContexts - Define an application's resources requirements - Create & consume Secrets - Understand ServiceAccounts

patterns: - define command and arguments for a container - define environment variable - expose pod information via env_var - distribute credential using secrets - inject information using PodPreset

`command` ---> ENTRYPOINT

`args` --> CMD

```
spec:
  container:
    image: nginx
    args: ['run', 'a']
```

2.2.1 configMaps

```
kubectl get configmaps
kubectl create configmap <cm-name> --from-literal=<key>=<name>
```

```
# how to defind configmap in container block

      envFrom:
      - configMapRef:
          name: special-config
```

2.2.2 Secrets, SecurityContexts

```
kubectl get secrets

# create new secret

kubectl create secret generic db-secret --from-literal=DB_Host=sql01 --from-literal=DB_User=root --from-literal=DB_Password=password
```

secret type - service account - secret - Opaque

```
# use secret in image block

envFrom:
- secretRef:
    name: special-config
```

```
# security contexts

kubectl exec ubuntu-sleeper whoami
```

2.2.3 Resource Limits

Assign CPU resources ot Containers and Pods

```
# container resources and limits

apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo-2
  namespace: cpu-example
spec:
  containers:
  - name: cpu-demo-ctr-2
    image: vish/stress
    resources:
      limits:
        cpu: "100"
      requests:
        cpu: "100"
    args:
    - -cpus
    - "2"
```

2.2.4 Service Account

Usage:

> To communicate with the API server, a Pod uses a ServiceAccount containing an authentication token.

```
# service account

apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
```

```
# create via cli

kubectl apply -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-robot
EOF

# retrieve token

kubectl describe secret <dashboard-sa-secret-name>

# example

$ kubectl get secret default-token-dffkj -o yaml

apiVersion: v1
data:
 ca.crt: LS0tLS1CRU...0tLS0tCg==
 namespace: ZGVmYXVsdA==
 token: ZXlKaGJHY2...RGMUlIX2c=
kind: Secret
metadata:
 name: default-token-dffkj
 namespace: default
 ...
type: kubernetes.io/service-account-token
```

```
 # binding roles to sa

# sa
apiVersion: v1
kind: ServiceAccount
metadata:
 name: demo-sa

# role
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
 name: list-pods
 namespace: default
rules:
 - apiGroups:
   - ''
 resources:
   - pods
 verbs:
   - list

# binding
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: RoleBinding
metadata:
 name: list-pods_demo-sa
 namespace: default
roleRef:
 kind: Role
 name: list-pods
 apiGroup: rbac.authorization.k8s.io
subjects:
 - kind: ServiceAccount
   name: demo-sa
   namespace: default
```

`ca.crt` is the Base64 encoding of the cluster certificate.

`token` is the Base64 encoding of the JWT used to authenticate against the API server.

what is the relationship between GSA and KSA?

2.3 multi-container pods

Goals: - Understand Multi-Container Pod design patterns

2.3.1 taint, toleration

taint and toleration are mostly used together, they are mainly used to manage resources for specfic usage cases.

what is effect in tolerations? e.g. `NoSchedule`, `NoExecute`

```
 # taint

kubectl taint nodes node01 spray=mortein:NoSchedule
```

2.3.2 Affinity

Node affinity

> node affinity allows you to constrain which nodes your pod is eligible to be scheduled on, based on labels on the node.

2.3.3 Common patterns

Patterns: - sidecar - adapter - ambassador

sidecar (logging services)

Using a sidecar container with the logging agent

```
# create sidecar container for storing logging

apiVersion: v1
kind: Pod
metadata: app
  name: app
  namesapce: elastic-stack
  labels:
    name: app
spec:
  containers:
  - name: app
    image: kodekcloud/event-simulator
    volumeMounts:
    - mountPath: /log
      name: log-volume

  - name: sidecar
    image: kodekloud/filebeat-configured
    volumeMounts:
    - mountPath: /var/log/event-simulator/
      name: log-volume

  volumes:
  - name: log-volume
    hostPath:
      path: /var/log/webapp
      type: DirectoryOrCreate
```

2.4 Observability

Goals: - Understand LivenessProbes and ReadinessProbes - Understand container logging - Understand how to monitor applications - Understand debugging in kubernetes

2.4.1 Pod Lifecycle

Pod Liftcycle

Container probes - livenessProbe - readinessProbe - startupProbe

basically it detects if the application is ready for receiving traffic

- pending
- containerCreating
- running
- Error
- completeed

2.4.1 Logging

logging architecture

- node level logging
- cluster level logging

You should remember that native kubernetes does not support extensive logging mechanism. The managed service like GKE, EKS makes life easier at some cost.

logging agent options: - stackdriver monitoring - elastic search

```
kubectl logs -f <pod-name> <container-name>
```

2.4.2 Monitoring

In GKE, cloud monitoring makes the monitoring an ease to use. Apart from that, datadog is a go-to option in the industry.

2.5 Pod Design

Goals: - Understand Deployments and how to perform rolling updates, rollbacks - Understand Jobs and CronJobs - Understand how to use Labels, Selectors and Annotations

2.5.1 labels, selectors

```
# use label to retrieve info
kubectl get pods -l <label name>

# Identify the POD which is 'prod', part of 'finance' BU and is a 'frontend' tier?
# set based syntax
kubectl get pods -l 'env in (prod), bu in(finance), tier in (frontend)'
# equality based syntax
kubectl get pods -l env=prod, bu=finance, tier=frontend

# select resources based on resource fields
kubectl get pods --field-selector status.phase=Running
```

```
# metadata block has pre-defined key, including name, labels, while within labels, fields can be defined freely.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: "{{ APP_NAME }}-{{ENVIRONMENT}}"
  labels:
    app: "{{ APP_NAME }}-{{ENVIRONMENT}}"
    env: {{ ENVIRONMENT }}
    app.kubernetes.io/name: "{{ APP_NAME }}-{{ENVIRONMENT}}"
    app.kubernetes.io/version: "{{ APP_VERSION }}"
    app.kubernetes.io/component: dataflow
    app.kubernetes.io/managed-by: chappie
# selector's label name should match labels field inside template
spec:
  selector:
      matchLabels:
        app: {{ APP_NAME }}-{{ENVIRONMENT}}
  template:
    metadata:
      labels:
        app: {{ APP_NAME }}-{{ENVIRONMENT}} # should match selector label
```

2.5.2 Rolling updates

strategy type: - RollingUpdate - 25% max unavailable, 25$ max surge - Recreate - RollingBacks

```
kubectl edit deployment frontend
kubectl set image <deployment-name> <image-name>
kubectl rollout status <deployment-name>
kubectl rollout history <deployment-name>
kubectl rollout undo <deployment-name>

# interacting with k8s pods/node
kubectl exec --namespace=<ns> curl -- sh -c '<doing something>'
```

2.5.3 Job

job is basically a task to achieve certian goals. You can set `backoffLimit` which tells it to retry up to how many times until the goal is achieved.

what kind of tasks are suitable to run as job: - non-parallel job - parallel jobs with a work queue - parallel jobs with a fixed completion count

jobs: one-off run

batch jobs

restartPolicy: Never/Always

parallelism: 3

2.5.4 CronJob

looks like job and cronjob support different fields in spec

```
# cronjob configuration

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: <container_name>
          restartPolicy: OnFailure
```

e.g. `30 19 */1 * *` schedule the job to run everyday at 19:30 UTC

cron expression validator

Field name	Mandatory?	Allowed values	Allowed special characters
Seconds	Yes	0-59	* / , -
Minutes	Yes	0-59	* / , -
Hours	Yes	0-23	* / , -
Day of month	Yes	1-31	* / , - ?
Month	Yes	1-12 or JAN-DEC	* / , -
Day of week	Yes	0-6 or SUN-SAT	* / , - ?

2.6 Service & Networking

Goals: - Understand Services - Basic understanding of network policies

Difficulties - Define ingress/egress rules, protocol, IP, port

Services basically give pod a static address so that another pods can work upon on these backend pods.

2.6.1 Services

Multi-Port Services

Discovering Services - Env variables - DNS (preferred)

Publishing Services - ClusterIP - NodePort - LoadBalancer - ExternalName

Supported protocols - TCP - UDP - HTTP - PROXY protocol - SCTP

```
# sample yaml

apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp  # should be the name of target deployment
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```
# expose deployments as service in a namespace
# question: how to specify the app name to be exposed via selector?

kubectl expose deployment -n ingress-space ingress-controller --type=NodePort --port=80 --name=ingress --dry-run -o yaml >ingress.ya
```

2.6.2 Network Policies

ingress type: - Single Service Ingress - Simple fanout - Name based virtual hosting - TLS - Load balancing

egress

How to access deployments in other namespaces

Nginx Ingress rewrite

```
# create a new service

apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  namespace: critical-space
  annotations:
    # why we need annotations here?
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /pay
        backend:
          serviceName: pay-service
          servicePort: 8282
```

```
# > Create a network policy to allow traffic from the 'Internal' application only to the 'payroll-service' and 'db-service'

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: <policy-name>
spec:
  podSelector:
    matchLabel:
      name: internal
  policyTypes:
  - Egress
  - Ingress
  ingress:
    - {}
  egress:
  - to:
    - podSelector:
        matchLabels:
          name: <pod_name_1>
    ports:
    - protocol: TCP
      port: <port>

  - to:
    - podSelector:
        matchLabels:
          name: <pod_name_2>
    ports:
    - protocol: TCP
      port: <port>
```

```
# example ingress resources

master $ kubectl describe ingresses.networking.k8s.io ingress-wear-watch -n app-space
Name:             ingress-wear-watch
Namespace:        app-space
Address:
Default backend:  default-http-backend:80 (<none>)
Rules:
  Host  Path  Backends
  ----  ----  --------
  *
        /wear    wear-service:8080 (10.32.0.2:8080)
        /watch   video-service:8080 (10.32.0.3:8080)
Annotations:
  nginx.ingress.kubernetes.io/rewrite-target:  /
  nginx.ingress.kubernetes.io/ssl-redirect:    false
Events:
  Type    Reason  Age    From                      Message
  ----    ------  ----   ----                      -------
  Normal  CREATE  6m21s  nginx-ingress-controller  Ingress app-space/ingress-wear-watch
  Normal  UPDATE  6m20s  nginx-ingress-controller  Ingress app-space/ingress-wear-watch
```

2.7 State Persistence

Goal: Understand PersistentVolumeClaims for storage

doc link

Concepts: - Reclaim Policy - Access Mode

```
# Task 1. create a volumn for log storage for in a pod
spec:
  containers:
  - name: event-simulator
    image: kodekloud/event-simulator
  volumes:

  - name: log-volume
    hostPath:
      path: /var/log/webapp
      type: Directorymaster $

# Task 2. create pvc to bound to pv

# Task 3. use pvc in application pods
spec:
  # emit details here
  volumes:
  - name: log-volume
    persistentVolumeClaim:
      claimName: claim-log-1master $

# Task 4. try delete pvc? try delete app and then pvc?
```

```
# Persistent Volume

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2

# Persistent Volume Claims

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

2.8 Optional topics

- static provisioning
- dynamic provisioning
- Stateful sets

FAQ

how to create a cluster?

how to create a pod?

what's the best access pattern

how to use template?

yaml syntax practing

how to switch context?

what does `READY` column stand for in `get pods` output?

```
# create pod

apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox

# change image name in pod
```

Reading List

lucassha-CKAD #k8s-tips-service-account CKAD candidate book

Common Errors

ReplicaSet

> The ReplicaSet "replicaset-2" is invalid: spec.template.metadata.labels: Invalid value: map[string]string{"tier":"nginx"}: `selector` does not match template `labels`

Errors occuring during the provising of pods - ImagePullBackOff - CrashLoopBackOff

Cmd reference

```
kubectl api-versions
kubectl api-resources

# remember shortnames
pv
pvc
ing
...

kk explain pod.spec.nodeSelector
# update resources
kk replace -f example.yaml
kk scale deployment/d1 --replicas=2
kk create --edit

# connect to running container
kubectl -n <namespace> exec -it <pods_name> -- /bin/bash
```