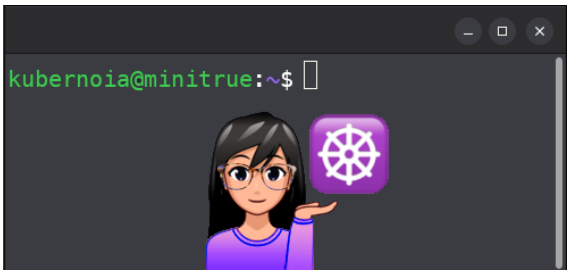
GKE CHEATSHEET

BY PAULOBA



kubectl (kubecontrol)

Kubernetes provides a command line tool for communicating with a Kubernetes cluster's control plane, using the Kubernetes API. This tool is named kubectl.
Official cheat sheet guide → kubectl Cheat Sheet

Kubectl short names

```
$ kubectl --namespace=production describe horizontalpodautoscalers
```

Even with shell autocompletion, it's a long command to type. Thankfully, the k8s community maintains short identifiers for common resource types.

That same command can be shortened to:

```
$ kubectl -n production describe hpa
```

To list short names for k8s commands, run:

```
$ kubectl api-resources
NAME                SHORTNAMES   APIVERSION     NAMESPACED   KIND
bindings                         v1             true         Binding
componentstatuses   cs           v1             false        ComponentStatus
configmaps          cm           v1             true         ConfigMap
endpoints           ep           v1             true         Endpoints
events              ev           v1             true         Event
[...]
```

Cluster management

How to connect to a gke cluster

1. Cloud sdk installed in the machine
    a. linux→ Install the gcloud CLI | Google Cloud
    b. mac → Install the gcloud CLI | Google Cloud
    c. windows → Install the gcloud CLI | Google Cloud

2. Kubectl tool
    a. linux → Install and Set Up kubectl on Linux
    b. mac→ Install and Set Up kubectl on macOS
    c. windows→ Install and Set Up kubectl on Windows

3. Set GCP project where the cluster lives
    a. gcloud config set project my-gcp-project

4. Acquire new user credentials to use for Application Default Credentials
    a. gcloud auth application-default login (go to point 5)
    b. Install the gke auth plugin:
        i. gcloud is the recommended way to install the binary on Windows and OS X
            1. gcloud components install gke-gcloud-auth-plugin
        ii. gke-gcloud-auth-plugin --version
        If you install the plugin prior to the release of 1.26 and experience issues with it, you can revert to using the provider-specific code by setting export USE_GKE_GCLOUD_AUTH_PLUGIN=True in ~/.bashrc (or in Environment variables for Windows) → Kubectl auth changes in GKE v1.26 : gke-gcloud-auth-plugin | Google Cloud Blog

5. Connect to the cluster
    a. gcloud container clusters get-credentials my-cluster --zone europe-west2-a --project my-gcp-project
    b. Check that you are really connected, running any kubectl command
        i. kubectl get po --all-namespaces
    c. *If you cannot connect to the cluster, go to point 4.b. above*

Check to what cluster are you connected to

$ kubectl config get-contexts | grep "*"

Basic commands

- Appending --all-namespaces happens frequently enough where you should be aware of the shorthand for --all-namespaces:
    - $ kubectl -A
- Get the list of pods and the status associated with each pod
    - $ kubectl get po -w --output-watch-events -A
- Get all resources across a namespace
    - $ kubectl get all -n my-namespace

Get pod metrics

The command below queries the Metrics API for the metrics and presents them to you. In most clusters, especially those provided by cloud services, the Metrics API will already be installed.
$ kubectl top po -n my-namespace

Scale deployment up/down

$ kubectl -n my-namespace scale --replicas=1 deployment/my-deployment

Order pod events by timestamp

$ kubectl -n my-namespace get events --sort-by='{.lastTimestamp}'

$ kubectl get events --sort-by=.metadata.creationTimestamp -A

Check pod resources consumption

$ kubectl top po

Check node metrics

```
$ kubectl top node
```

```
NAME                CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
gke-my-node         338m         4%     1662Mi          10%
```

How to read the output from kubectl top?

The output from `kubectl top node` gives you information about CPU(cores), CPU%, memory, and memory%. Let's see what these terms mean:

- ***CPU (cores)***
  `338m` means 338 millicpu. `1000m` is equal to 1 CPU, hence 338m means 33.8% of 1 CPU.
- ***CPU%***
  It is displayed only for nodes, and it stands for the total CPU usage percentage of that node.
- ***Memory***
  Memory being used by that node
- ***Memory%***
  It is also displayed only for nodes, and it stands for total memory usage percentage of that node.

Set application image

```
$ kubectl set image --record deployment/my-deployment
'my-deployment=us.gcr.io/my-gcp-project-name/my-image-name:2.0.0' --namespace my-namespace
```

Watch pods

You can use --watch/-w flag to watch pods similar to the watch command. This is useful for example when deploying an aplication so you can see if containers get up, or if the pod status is error, crashLoopBackoff etc.

```
$ kubectl -n my-namespace get po --watch |  grep pod-name
```

```
$ kubectl -n my-namespace get po -w |  grep pod-name
```

Add label to pod

Labels can be used to select objects and to find collections of objects that satisfy certain conditions. You can use labels for example to deploy a pod on specific node.

$ kubectl label pods <your-pod-name> <label-name>

e.g. $ kubectl label pods test-pod-rsyslog app=prod

Remove label from pod

kubectl label node <nodename> <labelname>-

Is the same command that adds a label but with a minus sign with the label name.

Get pods with labels

$ kubectl get po --show-labels

Get pods and the node where the pod is running

$ kubectl get po -owide

Check the health of API endpoints

$ kubectl get --raw='/readyz?verbose'

previously $ kubectl get cs

Decode k8s secrets

$ kubectl get secret datadog-secret -o jsonpath='{.data}' | base64 -d

$ kubectl get secret datadog-secret -o jsonpath='{.data.app-key}' | base64 -d

Useful: $ echo [BASE64-ENCODED-TEXT]] | base64 --decode

Deploy a temporary curl pod for debugging

```
$ kubectl run curlpod --image=curlimages/curl -i --tty -- sh
```

```
$ kubectl exec -it curl-pod -n default -- sh
```

Deploy a temporary pod with a Mysql client

You can use this pod to debug issues or perform tests, for example when connecting with the MySQLclient to CloudSQL instance within a GCP project.

```
$ kubectl run mysql-client --image=mysql:5.7 -it --rm --restart=Never -- /bin/bash
```

```
$ kubectl exec -it mysql-client bash
```

```
$ mysql --host=XXX.XXX.XXX.XXX --user=[DATABASE-USER] --password=[DATABASE-PASSWORD]
```

```
show databases;
```

```
use DATABASE-NAME;
```

```
show tables;
```

```
select * from TABLE-NAME;
```

Cleanup pods with undesired states

Some development environments have clusters that have spot vms in the nodepools, therefore from time to time you can find pods with the status *NodeAffinity* that are left from the VM being deleted and recreated.

- You can set up a cronjob that removes pods daily
- Alternatively you can delete those and other unwanted pods by running this bash script, whenever you want manually: https://github.com/pauloba/k8s-scripts/blob/main/del-pods.sh

Operate carefully, some pods with state *Completed* may not be of your interest to be deleted, since you may want to check the events of those pods for troubleshooting purposes.

kubectl commands useful for debugging

This interesting post from Thomas Stringer to dump pod logs and events from a cluster and search for errors with grep.

dump_pod_logs.sh

```
#!/bin/bash

ROOT_OUTPUT_DIR="/tmp"
OUTPUT_DIR_NAME="pod_logs_$(kubectl config current-context)_$(date +%s)"
OUTPUT_DIR="${ROOT_OUTPUT_DIR}/${OUTPUT_DIR_NAME}"
EXTENSION="log"
echo "Using output dir $OUTPUT_DIR"
mkdir "$OUTPUT_DIR"

# Get all pod logs and describe
kubectl get po -A --no-headers | while read -r line; do
    NAMESPACE=$(echo "$line" | awk '{print $1}')
    POD_NAME=$(echo "$line" | awk '{print $2}')
    FILENAME="${OUTPUT_DIR}/${NAMESPACE}.${POD_NAME}.describe"
    kubectl describe pod -n "$NAMESPACE" "$POD_NAME" > "$FILENAME"
    for CONTAINER in $(kubectl get po -n "$NAMESPACE" "$POD_NAME" -o
jsonpath="{.spec.containers[*].name}"); do
        FILENAME_PREFIX="${OUTPUT_DIR}/${NAMESPACE}.${POD_NAME}.${CONTAINER}"
        FILENAME="${FILENAME_PREFIX}.current.${EXTENSION}"
        echo "$FILENAME"
        kubectl logs -n "$NAMESPACE" "$POD_NAME" "$CONTAINER" > "$FILENAME"
        FILENAME="${FILENAME_PREFIX}.previous.${EXTENSION}"
        echo "$FILENAME"
        kubectl logs -p -n "$NAMESPACE" "$POD_NAME" "$CONTAINER" > "$FILENAME" 2> /dev/null
    done
done

# Dump all events
FILENAME="${OUTPUT_DIR}/events.log"
kubectl get events -A > "$FILENAME"

CWD=$(pwd)
cd $ROOT_OUTPUT_DIR || exit 1

TARBALL_FILE_NAME="${OUTPUT_DIR_NAME}.tar.gz"
tar -czvf "./${TARBALL_FILE_NAME}" "./${OUTPUT_DIR_NAME}"
mv "./${TARBALL_FILE_NAME}" "$OUTPUT_DIR"
```

```
echo
echo "Files located at $OUTPUT_DIR"
echo "Tarball located at ${OUTPUT_DIR}/${TARBALL_FILE_NAME}"
echo
echo "Search for errors:"
echo "  $ grep -Ei \"fail|err\" ${OUTPUT_DIR}/*.log"

cd "$CWD" || exit 1
```

You can execute it with `$ ./dump_pod_logs.sh`
and search for errors with `$ grep -Ei "fail|err" /tmp/pod_logs_kind-kind_1636422622/*.log`

- Get all pods from a given namespace and the node where the pod is running
  - kubectl get po -n namespace -o=custom-columns=NODE:.spec.nodeName,POD:.metadata.name
- Get all events with a warning
  - $ kubectl get events -w --field-selector=type=Warning -A
- Get documentation on the terminal
  - $ kubectl explain pods
  - $ kubectl explain deployment
  - $ kubectl explain replicaset
- Describe commands with verbose output
  - $ kubectl describe nodes my-node
  - $ kubectl describe pods my-pod
- List PersistentVolumes sorted by capacity
  - $ kubectl get pv --sort-by=.spec.capacity.storage
- Show labels for all pods (or any other Kubernetes object that supports labeling)
  - $ kubectl get pods --show-labels
- List all secrets currently in use by a pod

$ kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq

- Compares the current state of the cluster against the state that the cluster would be in if the manifest was applied.
  - $ kubectl diff -f ./my-manifest.yaml
- Wait for pods with a given label to be on a specific status. This could be useful for scripting, jenkins jobs and any automation.
  - $ kubectl -n my-namespace wait --for=condition=ready pod -l security.istio.io/tlsMode=istio
- Show pods sorted by memory use

    - $ kubectl top pods -n my-namespace --sort-by='memory'
- Show pods sorted by cpu use
    - $ kubectl top pods -n my-namespace --sort-by='cpu'
- Live trail logs for a specified pod. Use --follow or -f to stream to you terminal)
    - kubectl logs -n namespace pod-name-54c45f97f7-sghwt -f

kubectl commands useful for configuration

- Getting json/yaml config. files
    - kubectl get deploy -n my-namespace application-frontend -ojson
        - get the json file for the application named application-frontend
    - kubectl get cm -n my-namespace application-backend -oyaml
        - get the yaml file for the ConfigMag named application-backend

- Creating objects
  Kubernetes manifests can be defined in YAML or JSON. The file extension .yaml, .yml, and .json can be used.
    - $ kubectl apply -f ./my-manifest.yaml
        - # create resource(s)
    - $ kubectl apply -f ./my1.yaml -f ./my2.yaml
        - # create from multiple files
    - $ kubectl apply -f ./dir
        - # create resource(s) in all manifest files in dir
    - $ kubectl apply -f https://git.io/vPieo
        - # create resource(s) from url
    - $ kubectl create deployment nginx --image=nginx
        - # start a single instance of nginx

- Forwarding ports
    - kubectl port-forward svc/<service_name> XXXX:YYYY
    - kubectl port-forward <pod_name> XXXX:YYYY

- Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote pod (current ns)

    - $ kubectl cp /tmp/foo_dir my-pod:/tmp/bar_dir
- Copy /tmp/foo local file to /tmp/bar in a remote pod in a specific container
    - $ kubectl cp /tmp/foo my-pod:/tmp/bar -c my-container
- Copy /tmp/foo local file to /tmp/bar in a remote pod (ns=my-namespace)
    - $ kubectl cp /tmp/foo my-namespace/my-pod:/tmp/bar
- Copy /tmp/foo from a remote pod to /tmp/bar locally
    - $ kubectl cp my-namespace/my-pod:/tmp/foo /tmp/bar

gcloud CLI

The Google gcloud CLI is a set of tools to create and manage Google Cloud resources. You can use these tools to perform many common platform tasks from the command line or through scripts and other automation.

There is an official gcloud CLI cheatsheet.

Authenticate with your account to a cluster

To be able to set up certain configurations within a cluster it's needed to be authenticated with your account.

$ gcloud auth login

$ gcloud container clusters get-credentials cluster-name --zone us-east5-a --project gcp-project-name

ASM HUB membership

$ gcloud container hub mesh describe

$ gcloud container hub mesh disable

$ gcloud container hub memberships list

$ gcloud container hub memberships delete [MEMBERSHIP-NAME]

Delete nodepool

$ gcloud container node-pools list --cluster [CLUSTER] --region [REGION]

$ gcloud container node-pools delete NAME [--async] [--cluster=CLUSTER] [--region=REGION | --zone=ZONE, -z ZONE] [GCLOUD_WIDE_FLAG ...]

E.g.
$ gcloud container node-pools list --cluster my-cluster --region us-east5-a

$ gcloud container node-pools delete my-node-pool --cluster my-cluster --region us-east5-a

Scale nodepool to 0

$ gcloud container node-pools list --cluster [CLUSTER] --region [REGION]

$ gcloud container clusters resize CLUSTER_NAME --node-pool NAME_OF_THE_POOL --num-nodes 0

E.g. $ gcloud container node-pools list --cluster my-cluster --region us-east5-a

$ gcloud container clusters resize my-cluster --node-pool example-node-pool --num-nodes 0 --zone us-east5-a

Cluster access

You can add a couple of bash functions to a config. file to simplify accessing clusters.

1. Edit your ~/.bashrc or your ~/.zshrc
2. Add these 2 bash functions

```
function cluster( ) {
  gcloud container clusters get-credentials $1 --zone us-east5-a --project $2
}
function cluster-eu( ) {
  gcloud container clusters get-credentials $1 --zone europe-west12-b --project $2
}
```

3. Reload your bash configuration by running
   $ source ~/.bashrc

or depending on the shell you use

```
$ source ~/.zshrc
```

Now you can log into clusters located in *us-east5-a* by running:

```
$ cluster CLUSTER-NAME PROJECT-NAME
```

and you can log into clusters located in *europe-west12-b* running:

```
$ cluster-eu CLUSTER-NAME PROJECT-NAME
```