Kubectl Context and Configuration

Set which Kubernetes cluster `kubectl` communicates with and modifies configuration information. See Authenticating Across Clusters with kubeconfig documentation for detailed config file information.

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # display the first user
kubectl config view -o jsonpath='{.users[*].name}'   # get a list of users
kubectl config get-contexts                          # display list of contexts
kubectl config current-context                       # display the current-
context
kubectl config use-context my-cluster-name           # set the default context to
my-cluster-name

# add a new user to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --
password=kubepassword

# permanently save the namespace for all subsequent kubectl commands in that
context.
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete user foo
```

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
kubectl explain pods                           # get the documentation for pod
manifests

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - sleep
    - "1000"
EOF

# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
EOF
```

Viewing, Finding Resources

```
# Get commands with basic output
kubectl get services                          # List all services in the
namespace
kubectl get pods --all-namespaces             # List all pods in all namespaces
kubectl get pods -o wide                      # List all pods in the current
namespace, with more details
kubectl get deployment my-dep                 # List a particular deployment
kubectl get pods                              # List all pods in the namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Retrieve the value of a key with dots, e.g. 'ca.crt'
kubectl get configmap myconfig \
  -o jsonpath='{.data.ca\.crt}'

# Get all worker nodes (use a selector to exclude results that have a label
# named 'node-role.kubernetes.io/master')
kubectl get node --selector='!node-role.kubernetes.io/master'

# Get all running pods in the namespace
kubectl get pods --field-selector=status.phase=Running

# Get ExternalIPs of all nodes
kubectl get nodes -o
jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath, it
can be found at https://stedolan.github.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries |
.[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --
output=jsonpath={.items..metadata.name})

# Show labels for all pods (or any other Kubernetes object that supports
labelling)
kubectl get pods --show-labels

# Check which nodes are ready
```

```
JSONPATH='{range .items[*]}{@.metadata.name}:{range
@.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# 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

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of
initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range
.items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/ -
f3

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp

# 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
```

Updating Resources

```
kubectl set image deployment/frontend www=image:v2               # Rolling update
"www" containers of "frontend" deployment, updating the image
kubectl rollout history deployment/frontend                      # Check the
history of deployments including the revision
kubectl rollout undo deployment/frontend                         # Rollback to
the previous deployment
kubectl rollout undo deployment/frontend --to-revision=2         # Rollback to a
specific revision
kubectl rollout status -w deployment/frontend                    # Watch rolling
update status of "frontend" deployment until completion
kubectl rollout restart deployment/frontend                      # Rolling
restart of the "frontend" deployment


cat pod.json | kubectl replace -f -                              # Replace a pod
based on the JSON passed into std

# Force replace, delete and then re-create the resource. Will cause a service
outage.
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and connects
to the containers on port 8000
kubectl expose rc nginx --port=80 --target-port=8000
```

```
# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | kubectl
replace -f -

kubectl label pods my-pod new-label=awesome                      # Add a Label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # Add an
annotation
kubectl autoscale deployment foo --min=2 --max=10                # Auto scale a
deployment "foo"
```

Patching Resources

```
# Partially update a node
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# Update a container's image; spec.containers[*].name is required because it's a
merge key
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-
hostname","image":"new image"}]}}'

# Update a container's image using a json patch with positional arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path":
"/spec/containers/0/image", "value":"new image"}]'

# Disable a deployment livenessProbe using a json patch with positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove",
"path": "/spec/template/spec/containers/0/livenessProbe"}]'

# Add a new element to a positional array
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1",
"value": {"name": "whatever" } }]'
```

Editing Resources

Edit any API resource in your preferred editor.

```
kubectl edit svc/docker-registry                      # Edit the service named
docker-registry
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an alternative editor
```

Scaling Resources

```
kubectl scale --replicas=3 rs/foo                                 # Scale a
replicaset named 'foo' to 3
kubectl scale --replicas=3 -f foo.yaml                            # Scale a
resource specified in "foo.yaml" to 3
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # If the
deployment named mysql's current size is 2, scale mysql to 3
kubectl scale --replicas=5 rc/foo rc/bar rc/baz                   # Scale
multiple replication controllers
```

Deleting Resources

```
kubectl delete -f ./pod.json                                              #
Delete a pod using the type and name specified in pod.json
kubectl delete pod,service baz foo                                        #
Delete pods and services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel                              #
Delete pods and services with label name=myLabel
kubectl -n my-ns delete pod,svc --all                                      #
Delete all pods and services in namespace my-ns,
# Delete all pods matching the awk pattern1 or pattern2
kubectl get pods  -n mynamespace --no-headers=true | awk
'/pattern1|pattern2/{print $1}' | xargs  kubectl delete -n mynamespace pod
```

Interacting with running Pods

```
kubectl logs my-pod                                 # dump pod logs (stdout)
kubectl logs -l name=myLabel                        # dump pod logs, with label
name=myLabel (stdout)
kubectl logs my-pod --previous                      # dump pod logs (stdout) for
a previous instantiation of a container
kubectl logs my-pod -c my-container                 # dump pod container logs
(stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container        # dump pod logs, with label
name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous      # dump pod container logs
(stdout, multi-container case) for a previous instantiation of a container
kubectl logs -f my-pod                              # stream pod logs (stdout)
kubectl logs -f my-pod -c my-container              # stream pod container logs
(stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers    # stream all pods logs with
label name=myLabel (stdout)
kubectl run -i --tty busybox --image=busybox -- sh  # Run pod as interactive
shell
kubectl run nginx --image=nginx -n
mynamespace                                         # Run pod nginx in a specific
namespace
kubectl run nginx --image=nginx                     # Run pod nginx and write its
spec into a file called pod.yaml
--dry-run=client -o yaml > pod.yaml
```

```
kubectl attach my-pod -i                            # Attach to Running Container
kubectl port-forward my-pod 5000:6000               # Listen on port 5000 on the
local machine and forward to port 6000 on my-pod
kubectl exec my-pod -- ls /                         # Run command in existing pod
(1 container case)
kubectl exec my-pod -c my-container -- ls /         # Run command in existing pod
(multi-container case)
kubectl top pod POD_NAME --containers               # Show metrics for a given
pod and its containers
```

Interacting with Nodes and Cluster

```
kubectl cordon my-node                                                # Mark my-
node as unschedulable
kubectl drain my-node                                                 # Drain my-
node in preparation for maintenance
kubectl uncordon my-node                                              # Mark my-
node as schedulable
kubectl top node my-node                                              # Show
metrics for a given node
kubectl cluster-info                                                  # Display
addresses of the master and services
kubectl cluster-info dump                                             # Dump
current cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state   # Dump
current cluster state to /path/to/cluster-state

# If a taint with that key and effect already exists, its value is replaced as
specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule
```

Resource types

List all supported resource types along with their shortnames, API group, whether they are namespaced, and Kind:

```
kubectl api-resources
```

Other operations for exploring API resources:

```
kubectl api-resources --namespaced=true      # All namespaced resources
kubectl api-resources --namespaced=false     # All non-namespaced resources
kubectl api-resources -o name                # All resources with simple output
(just the resource name)
kubectl api-resources -o wide                # All resources with expanded (aka
"wide") output
kubectl api-resources --verbs=list,get       # All resources that support the
"list" and "get" request verbs
kubectl api-resources --api-group=extensions # All resources in the "extensions"
API group
```

Formatting output

To output details to your terminal window in a specific format, add the `-o` (or `--output`) flag to a supported `kubectl` command.

Output format	Description
`-o=custom-columns=<spec>`	Print a table using a comma separated list of custom columns
`-o=custom-columns-file=<filename>`	Print a table using the custom columns template in the `<filename>` file
`-o=json`	Output a JSON formatted API object
`-o=jsonpath=<template>`	Print the fields defined in a jsonpath expression
`-o=jsonpath-file=<filename>`	Print the fields defined by the jsonpath expression in the `<filename>` file
`-o=name`	Print only the resource name and nothing else
`-o=wide`	Output in the plain-text format with any additional information, and for pods, the node name is included
`-o=yaml`	Output a YAML formatted API object

Examples using `-o=custom-columns`:

```
# All images running in a cluster
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'

 # All images excluding "k8s.gcr.io/coredns:1.6.2"
kubectl get pods -A -o=custom-
columns='DATA:spec.containers[?(@.image!="k8s.gcr.io/coredns:1.6.2")].image'

# All fields under metadata regardless of name
kubectl get pods -A -o=custom-columns='DATA:metadata.*'
```

Kubectl output verbosity and debugging

Kubectl verbosity is controlled with the `-v` or `--v` flags followed by an integer representing the log level. General Kubernetes logging conventions and the associated log levels are described here.

Verbosity	Description
`--v=0`	Generally useful for this to *always* be visible to a cluster operator.
`--v=1`	A reasonable default log level if you don't want verbosity.

Verbosity	Description
`--v=2`	Useful steady state information about the service and important log messages that may correlate to significant changes in the system. This is the recommended default log level for most systems.
`--v=3`	Extended information about changes.
`--v=4`	Debug level verbosity.
`--v=6`	Display requested resources.
`--v=7`	Display HTTP request headers.
`--v=8`	Display HTTP request contents.
`--v=9`	Display HTTP request contents without truncation of contents.