kubectl - Cheat Sheet

With thanks for the thorough job done at https://kubernetes.io/docs/reference/kubectl/cheatsheet .

Kubectl Autocomplete

BASH

```
# setup autocomplete in bash into the current shell,
# bash-completion package should be installed first.
source <(kubectl completion bash)

# add autocomplete permanently to your bash shell.
echo "source <(kubectl completion bash)" >> ~/.bashrc
```

ZSH

```
# setup autocomplete in zsh into the current shell
source <(kubectl completion zsh)

# add autocomplete permanently to your zsh shell
echo "if [ $commands[kubectl] ]; then source <(kubectl completion zsh); fi" \
  >>  ~/.zshrc
```

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}'

# Display the current-context
kubectl config current-context

# set the default context to my-cluster-name
kubectl config use-context my-cluster-name

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

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

Creating Objects

Kubernetes manifests can be defined in json or yaml. The file extension .yaml, .yml, and .json can be used.

```
# create resource(s)
kubectl create -f ./my-manifest.yaml

# create from multiple files
kubectl create -f ./my1.yaml -f ./my2.yaml

# create resource(s) in all manifest files in dir
kubectl create -f ./dir

# create resource(s) from url
kubectl create -f https://git.io/vPieo

# start a single instance of nginx
kubectl run nginx --image=nginx

# get the documentation for pod and svc manifests
kubectl explain pods,svc

# Create multiple YAML objects from stdin
cat <<EOF | kubectl create -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 create -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64)
  username: $(echo -n "jane" | base64)
EOF
```

Viewing, Finding Resources

```
# Get commands with basic output

# List all services in the namespace
kubectl get services

# List all pods in all namespaces
kubectl get pods --all-namespaces

# List all pods in the namespace, with more details
kubectl get pods -o wide

# List a particular deployment
kubectl get deployment my-dep

# List all pods in the namespace, including uninitialized ones
kubectl get pods --include-uninitialized

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

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

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

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

# 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})

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

Updating Resources

```
# Rolling update pods of frontend-v1
kubectl rolling-update frontend-v1 -f frontend-v2.json

# Change the name of the resource and update the image
kubectl rolling-update frontend-v1 frontend-v2 --image=image:v2

# Update the pods image of frontend
kubectl rolling-update frontend --image=image:v2

# Abort existing rollout in progress
kubectl rolling-update frontend-v1 frontend-v2 --rollback

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

# 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 -

# Add a Label
kubectl label pods my-pod new-label=awesome

# Add an annotation
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq

# Auto scale a deployment "foo"
kubectl autoscale deployment foo --min=2 --max=10
```

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

The edit any API resource in an editor.

```
# Edit the service named docker-registry
kubectl edit svc/docker-registry

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

Scaling Resources

```
# Scale a replicaset named 'foo' to 3
kubectl scale --replicas=3 rs/foo

# Scale a resource specified in "foo.yaml" to 3
kubectl scale --replicas=3 -f foo.yaml

# If the deployment named mysql's current size is 2, scale mysql to 3
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql

# Scale multiple replication controllers
kubectl scale --replicas=5 rc/foo rc/bar rc/baz
```

Deleting Resources

```
# Delete a pod using the type and name specified in pod.json
kubectl delete -f ./pod.json

# Delete pods and services with same names "baz" and "foo"
kubectl delete pod,service baz foo

# Delete pods and services with label name=myLabel
kubectl delete pods,services -l name=myLabel

# Delete pods and services, including uninitialized ones, with label name=myLabel
kubectl delete pods,services -l name=myLabel --include-uninitialized

# Delete all pods and services, including uninitialized ones, in namespace my-ns
kubectl -n my-ns delete po,svc --all
```

Interacting with running Pods

```
# dump pod logs (stdout)
kubectl logs my-pod

# dump pod logs (stdout) for a previous instantiation of a container
kubectl logs my-pod --previous

# dump pod container logs (stdout, multi-container case)
kubectl logs my-pod -c my-container

# dump pod container logs (stdout, multi-container case)
# for a previous instantiation of a container
kubectl logs my-pod -c my-container --previous

# stream pod logs (stdout)
kubectl logs -f my-pod

# stream pod container logs (stdout, multi-container case)
kubectl logs -f my-pod -c my-container

# Run pod as interactive shell
kubectl run -i --tty busybox --image=busybox -- sh

# Attach to Running Container
kubectl attach my-pod -i

# Listen on port 5000 on the local machine and forward to port 6000 on my-pod
kubectl port-forward my-pod 5000:6000

# Run command in existing pod (1 container case)
kubectl exec my-pod -- ls /

# Run command in existing pod (multi-container case)
kubectl exec my-pod -c my-container -- ls /

# Show metrics for a given pod and its containers
kubectl top pod POD_NAME --containers
```

Interacting with Nodes and Cluster

```
# Mark my-node as unschedulable
kubectl cordon my-node

# Drain my-node in preparation for maintenance
kubectl drain my-node

# Mark my-node as schedulable
kubectl uncordon my-node

# Show metrics for a given node
kubectl top node my-node

# Display addresses of the master and services
kubectl cluster-info

# Dump current cluster state to stdout
kubectl cluster-info dump

# Dump current cluster state to /path/to/cluster-state
kubectl cluster-info dump --output-directory=/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:

```
# All namespaced resources
kubectl api-resources --namespaced=true

# All non-namespaced resources
kubectl api-resources --namespaced=false

# All resources with simple output (just the resource name)
kubectl api-resources -o name

# All resources with expanded (aka "wide") output
kubectl api-resources -o wide

# All resources that support the "list" and "get" request verbs
```

```
kubectl api-resources --verbs=list,get

# All resources in the "extensions" API group
kubectl api-resources --api-group=extensions
```