```
** KUBECTL CONTEXT AND CONFIGURATION **
# Show Merged kubeconfig settings
kubectl config view
# display the first user
kubectl config view -o jsonpath='{.users[].name}'
# get a list of users
kubectl config view -o jsonpath='{.users[*].name}'
# display list of contexts
kubectl config get-contexts
# get all context names
kubectl config get-contexts -o name
# display the current-context
kubectl config current-context
# set the default context to my-cluster-name
kubectl config use-context my-cluster-name
# set a cluster entry in the kubeconfig
kubectl config set-cluster my-cluster-name
# permanently save the namespace for all subsequent kubectl commands in that
context
kubectl config set-context --current --namespace=ggckad-s2


** CREATING OBJECTS **

# create resource(s)
kubectl apply -f ./my-manifest.yaml
# create from multiple files
kubectl apply -f ./my1.yaml -f ./my2.yaml
# create resource(s) in all manifest files in dir
kubectl apply -f ./dir
# create resource(s) from url
kubectl apply -f https://example.com/manifest.yaml
# start a single instance of nginx
kubectl create deployment nginx --image=nginx
# create a Job which prints "Hello World"
kubectl create job hello --image=busybox:1.28 -- echo "Hello World"
# create a CronJob that prints "Hello World" every minute
kubectl create cronjob hello --image=busybox:1.28   --schedule="*/1 * * * *" --
echo "Hello World"
# get the documentation for pod manifests
kubectl explain pods


** VIEWING AND FINDING RESOURCES **

# 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 current namespace, with more details
kubectl get pods -o wide
# List a particular deployment
kubectl get deployment my-dep
# List all pods in the namespace
kubectl get pods
# Get a pod's YAML
kubectl get pod my-pod -o yaml
# Describe commands with verbose node output
kubectl describe nodes my-node
# Describe commands with verbose pod output
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}'
# Retrieve a base64 encoded value with dashes instead of underscores.
kubectl get secret my-secret --template='{{index .data "key-name-with-dashes"}}'
# 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}'
# 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"
# Check which nodes are ready with custom-columns
kubectl get node -o custom-
columns='NODE_NAME:.metadata.name,STATUS:.status.conditions[?
(@.type=="Ready")].status'
# Output decoded secrets without external tools
kubectl get secret my-secret -o go-template='{{range $k,$v := .data}}{{"### "}}
{{$k}}{{"\n"}}{{$v|base64decode}}{{"\n\n"}}{{end}}'
# 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
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
# List all warning events
kubectl events --types=Warning
# 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
# Produce a period-delimited tree of all keys returned for nodes
kubectl get nodes -o json | jq -c 'paths|join(".")'
# Produce a period-delimited tree of all keys returned for pods, etc
kubectl get pods -o json | jq -c 'paths|join(".")'
# Get a deployment's status subresource
kubectl get deployment nginx-deployment --subresource=status


** UPDATING RESOURCES **

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

```
# Rolling restart of the "frontend" deployment
kubectl rollout restart deployment/frontend
# 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
# Remove a label
kubectl label pods my-pod new-label-
# Overwrite an existing value
kubectl label pods my-pod new-label=new-value --overwrite
# Add an annotation
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq
# Remove annotation
kubectl annotate pods my-pod icon-url-
# 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" } }]'
# Update a deployment's replica count by patching its scale subresource
kubectl patch deployment nginx-deployment --subresource='scale' --type='merge' -
p '{"spec":{"replicas":2}}'


** EDITING RESOURCES **

# 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 **

kubectl delete -f ./pod.json                                        # Delete a pod
using the type and name specified in pod.json
kubectl delete pod unwanted --now                                   # Delete a pod
with no grace period
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 **

# dump pod logs (stdout)
kubectl logs my-pod
# dump pod logs, with label name=myLabel (stdout)
kubectl logs -l name=myLabel
# 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, with label name=myLabel (stdout)
kubectl logs -l name=myLabel -c my-container
# dump pod container logs (stdout, multi-container case) for a previous instant
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
# stream all pods logs with label name=myLabel (stdout)
kubectl logs -f -l name=myLabel --all-containers
# Run pod as interactive shell
kubectl run -i --tty busybox --image=busybox:1.28 -- sh
# Start a single instance of nginx pod in the namespace of mynamespace
kubectl run nginx --image=nginx -n mynamespace
# Generate spec for running pod nginx and write it into a file called pod.yaml
kubectl run nginx --image=nginx --dry-run=client -o yaml > pod.yaml
# 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 /
# Interactive shell access to a running pod (1 container case)
kubectl exec --stdin --tty my-pod -- /bin/sh
# Run command in existing pod (multi-container case)
kubectl exec my-pod -c my-container -- ls /
# Create an interactive debugging session within existing pod and immediately
attach to it
kubectl debug my-pod -it --image=busybox:1.28
# Create an interactive debugging session on a node and immediately attach to it
kubectl debug node/my-node -it --image=busybox:1.28
```

```
# Show metrics for all pods in the default namespace
kubectl top pod
# Show metrics for a given pod and its containers
kubectl top pod POD_NAME --containers
# Show metrics for a given pod and sort it by 'cpu' or 'memory'
kubectl top pod POD_NAME --sort-by=cpu


** COPYING FILES AND DIRECTORIES TO AND FROM CONTAINERS **

# Copy /tmp/foodir local directory to /tmp/bardir in a remote pod in the current
nS
kubectl cp /tmp/foodir my-pod:/tmp/bardir
# 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 in namespace 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
# Copy /tmp/foo from a remote pod to /tmp/bar locally
kubectl exec -n my-namespace my-pod -- tar cf - /tmp/foo | tar xf - -C /tmp/bar


** INTERACTING WITH DEPLOYMENTS AND SERVICES **

# dump Pod logs for a Deployment (single-container case)
kubectl logs deploy/my-deployment
# dump Pod logs for a Deployment (multi-container case)
kubectl logs deploy/my-deployment -c my-container
# listen on local port 5000 and forward to port 5000 on Service backend
kubectl port-forward svc/my-service 5000
# listen on local port 5000 and forward to Service target port with name <my-
service-port>
kubectl port-forward svc/my-service 5000:my-service-port
# listen on local port 5000 and forward to port 6000 on a Pod created by <my-
deployment>
kubectl port-forward deploy/my-deployment 5000:6000
# run command in first Pod and first container in Deployment (single- or multi-
container cases)
kubectl exec deploy/my-deployment -- ls


** 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 all nodes
kubectl top 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
# View existing taints on which exist on current nodes.
kubectl get nodes -o='custom-
columns=NodeName:.metadata.name,TaintKey:.spec.taints[*].key,TaintValue:.spec.ta
ints[*].value,TaintEffect:.spec.taints[*].effect'
```

```
# 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 **

# All namespaced resources
kubectl api-resources --namespaced=true
# All non-namespaced resources
kubectl api-resources --namespaced=false
# All resources with simple output (only 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


** FORMATTING OUTPUT **

# Print a table using a comma separated list of custom columns
-o=custom-columns=<spec>
# Print a table using the custom columns template in the <filename> file
-o=custom-columns-file=<filename>
# Print the fields defined in a golang template
-o=go-template=<template>
# Print the fields defined by the golang template in the <filename> file
-o=go-template-file=<filename>
# Output a JSON formatted API object
-o=json
# Print the fields defined in a jsonpath expression
-o=jsonpath=<template>
# Print the fields defined by the jsonpath expression in the <filename> file
-o=jsonpath-file=<filename>
# Output a KYAML formatted API object. KYAML is an Kubernetes-specific dialect
of YAML, and can be parsed as YAML.
-o=kyaml (beta)
# Print only the resource name and nothing else
-o=name
# Output in the plain-text format with any additional information, and for pods,
the node name is included
-o=wide
# Output a YAML formatted API object
-o=yaml


** IMAGES RUNNING IN THE CLUSTER **

# All images running in a cluster
kubectl get pods -A -o=custom-columns='DATA:spec.containers[*].image'
# All images running in namespace: default, grouped by Pod
kubectl get pods --namespace default --output=custom-
columns="NAME:.metadata.name,IMAGE:.spec.containers[*].image"
# All images excluding "registry.k8s.io/coredns:1.6.2"
kubectl get pods -A -o=custom-columns='DATA:spec.containers[?(@.image!
="registry.k8s.io/coredns:1.6.2")].image'
# All fields under metadata regardless of name
kubectl get pods -A -o=custom-columns='DATA:metadata.*'


** VERBOSITY DESCRIPTION **
```

```
# Generally useful for this to always be visible to a cluster operator.
--v=0
# A reasonable default log level if you don't want verbosity.
--v=1
# State information about the service and important log messages that may
correlate to significant changes in the system.
--v=2
# Extended information about changes.
--v=3
# Debug level verbosity.
--v=4
# Trace level verbosity.
--v=5
# Display requested resources.
--v=6
# Display HTTP request headers.
--v=7
# Display HTTP request contents.
--v=8
# Display HTTP request contents without truncation of contents.
--v=9
```