Design docs, concept definitions, and references for APIs and CLIs.

See also: Kubectl Overview and JsonPath Guide.

Kubectl Autocomplete

```
$ source <(kubectl completion bash) # setup autocomplete in bash
$ source <(kubectl completion zsh)  # setup autocomplete in zsh
```

Kubectl Context and Configuration

Set which Kubernetes cluster `kubectl` communicates with and modify configuration information. See kubeconfig file 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 current-context              # Display the current-context
$ kubectl config use-context my-cluster-name  # set the default context to my-c

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

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

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

```
$ kubectl create -f ./my-manifest.yaml           # create resource(s)
$ kubectl create -f ./my1.yaml -f ./my2.yaml     # create from multiple files
$ kubectl create -f ./dir                        # create resource(s) in all ma
$ kubectl create -f https://git.io/vPieo         # create resource(s) from url
$ kubectl run nginx --image=nginx                # start a single instance of n
$ kubectl explain pods,svc                       # get the documentation for po

# 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 "s33msi4" | base64)
  username: $(echo "jane" | base64)
EOF
```

```
# Get commands with basic output
$ kubectl get services                          # List all services in the name
$ kubectl get pods --all-namespaces             # List all pods in all namespac
$ kubectl get pods -o wide                      # List all pods in the namespac
$ kubectl get deployment my-dep                 # List a particular deployment

# 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 ExternalIPs of all nodes
$ kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="Externa

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath
$ sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entri
$ echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.n

# Check which nodes are ready
$ JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@
 && kubectl get nodes -o jsonpath=$JSONPATH | grep "Ready=True"
```

Updating Resources

```
$ kubectl rolling-update frontend-v1 -f frontend-v2.json             # Rolling up
```

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

# Create a service for a replicated nginx, which serves on port 80 and connects
$ 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/' | kubec

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

Patching Resources

Patch a resource(s) with a strategic merge patch.

```
$ kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}' # Partiall

# Update a container's image; spec.containers[*].name is required because it's
$ kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-se

# Update a container's image using a json patch with positional arrays
$ kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/sp
```

Editing Resources

The edit any API resource in an editor.

```
$ kubectl edit svc/docker-registry                      # Edit the service name
$ KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an alternative ed
```

Scaling Resources

```
$ kubectl scale --replicas=3 rs/foo                                 # Scale a r
```

Deleting Resources

```
$ kubectl delete -f ./pod.json                      # Delete a pod using the ty
$ kubectl delete pod,service baz foo                # Delete pods and services
$ kubectl delete pods,services -l name=myLabel      # Delete pods and services
$ kubectl -n my-ns delete po,svc --all              # Delete all pods and servi
```

Interacting with running Pods

```
$ kubectl logs my-pod                                 # dump pod logs (stdout)
$ kubectl logs -f my-pod                              # stream pod logs (stdout
$ kubectl run -i --tty busybox --image=busybox -- sh  # Run pod as interactive
$ kubectl attach my-pod -i                            # Attach to Running Conta
$ kubectl port-forward my-pod 5000 6000               # Forward port 6000 of Po
$ kubectl port-forward my-svc 6000                    # Forward port to service
$ kubectl exec my-pod -- ls /                         # Run command in existing
$ kubectl exec my-pod -c my-container -- ls /         # Run command in existing
$ kubectl top pod POD_NAME --containers               # Show metrics for a give
```

Interacting with Nodes and Cluster

```
$ kubectl cordon my-node                                                # Mark
$ kubectl drain my-node                                                 # Drain
$ kubectl uncordon my-node                                              # Mark
$ kubectl top node my-node                                              # Show
$ kubectl cluster-info                                                  # Displ
$ kubectl cluster-info dump                                             # Dump
$ kubectl cluster-info dump --output-directory=/path/to/cluster-state   # Dump

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

Resource types

The following table includes a list of all the supported resource types and their abbreviated aliases.

`componentstatuses`	`cs`
`configmaps`	`cm`
`daemonsets`	`ds`
`deployments`	`deploy`
`endpoints`	`ep`
`event`	`ev`
`horizontalpodautoscalers`	`hpa`
`ingresses`	`ing`
`jobs`	
`limitranges`	`limits`
`namespaces`	`ns`
`networkpolicies`	
`nodes`	`no`
`petset`	
`persistentvolumeclaims`	`pvc`
`persistentvolumes`	`pv`
`pods`	`po`
`podsecuritypolicies`	`psp`
`podtemplates`	
`replicasets`	`rs`
`replicationcontrollers`	`rc`
`resourcequotas`	`quota`
`cronjob`	

Resource type	Abbreviated alias
`services`	`svc`
`storageclasses`	
`thirdpartyresources`	

Formatting output

To output details to your terminal window in a specific format, you can add either the `-o` or `-output` flags 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

Create an Issue Edit this Page

Get Started

Documentation

Blog

Partners

Case Studies

Download K8s Contribute to the K8s codebase