****

**kubectl**

kubectl is a command line interface for running commands against Kubernetes clusters

The kubectl command-line utility is a powerful tool, and you will use it to create objects and interact with the Kubernetes API

List of general purpose commands for Kubernetes management:

* PODS
* Create Deployments
* Scaling PODs
* POD Upgrade / History
* Services
* Volumes
* Secrets
* Scheduler
* Troubleshooting
* Role Based Access Control (RBAC)
* Security Contexts
* Pod Security Policies
* Network Policies

kubectl help

**Basic Commands**

create

get

run

expose

delete

**Application Management**

apply

annotate

autoscale

convert

edit

label

patch

replace

rollout

scale

set

wait

Working with Apps

attach

auth

cp

describe

exec

logs

port-forward

proxy

top

Cluster Management

api-versions

certificate

cluster-info

cordon

drain

taint

uncordon

KUBECTL SETTINGS AND USAGE

alpha

api-resources

completion

config

explain

options

plugin

version

**Syntax**

Use the following syntax to run kubectl commands from your terminal window:

kubectl[command][TYPE][NAME][flags]

where command, TYPE, NAME, and flags are:

* command: Specifies the operation that you want to perform on one or more resources, for example create, get, describe, delete.
* TYPE: Specifies the resource type. Resource types are case-insensitive and you can specify the singular, plural, or abbreviated forms. For example, the following commands produce the same output:
* $ kubectl get pod pod1
* $ kubectl get pods pod1

$ kubectl get po pod1

* NAME: Specifies the name of the resource. Names are case-sensitive. If the name is omitted, details for all resources are displayed, for example $ kubectl get pods.

When performing an operation on multiple resources, you can specify each resource by type and name or specify one or more files:

* To specify resources by type and name:
  + To group resources if they are all the same type: TYPE1 name1 name2 name<#>.  
    Example: $ kubectl get pod example-pod1 example-pod2
  + To specify multiple resource types individually: TYPE1/name1 TYPE1/name2 TYPE2/name3 TYPE<#>/name<#>.  
    Example: $ kubectl get pod/example-pod1 replicationcontroller/example-rc1
* To specify resources with one or more files: -f file1 -f file2 -f file<#>
  + Use YAML rather than JSON since YAML tends to be more user-friendly, especially for configuration files.  
    Example: $ kubectl get pod -f ./pod.yaml
* flags: Specifies optional flags. For example, you can use the -s or --server flags to specify the address and port of the Kubernetes API server.  
  **Important**: Flags that you specify from the command line override default values and any corresponding environment variables.

If you need help, just run kubectl help from the terminal window.

## Operations

The following table includes short descriptions and the general syntax for all of the kubectl operations:

| **Operation** | **Syntax** | **Description** |
| --- | --- | --- |
| annotate | kubectl annotate (-f FILENAME \| TYPE NAME \| TYPE/NAME) KEY\_1=VAL\_1 ... KEY\_N=VAL\_N [--overwrite] [--all] [--resource-version=version] [flags] | Add or update the annotations of one or more resources. |
| api-versions | kubectlapi-versions [flags] | List the API versions that are available. |
| apply | kubectl apply -f FILENAME [flags] | Apply a configuration change to a resource from a file or stdin. |
| attach | kubectl attach POD -c CONTAINER [-i] [-t] [flags] | Attach to a running container either to view the output stream or interact with the container (stdin). |
| autoscale | kubectlautoscale (-f FILENAME \| TYPE NAME \| TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [flags] | Automatically scale the set of pods that are managed by a replication controller. |
| cluster-info | kubectl cluster-info [flags] | Display endpoint information about the master and services in the cluster. |
| config | kubectlconfig SUBCOMMAND [flags] | Modifies kubeconfig files. See the individual subcommands for details. |
| create | kubectl create -f FILENAME [flags] | Create one or more resources from a file or stdin. |
| delete | kubectl delete (-f FILENAME \| TYPE [NAME \| /NAME \| -l label \| --all]) [flags] | Delete resources either from a file, stdin, or specifying label selectors, names, resource selectors, or resources. |
| describe | kubectl describe (-f FILENAME \| TYPE [NAME\_PREFIX \| /NAME \| -l label]) [flags] | Display the detailed state of one or more resources. |
| edit | kubectl edit (-f FILENAME \| TYPE NAME \| TYPE/NAME) [flags] | Edit and update the definition of one or more resources on the server by using the default editor. |
| exec | kubectl exec POD [-c CONTAINER] [-i] [-t] [flags] [-- COMMAND [args...]] | Execute a command against a container in a pod, |
| explain | kubectl explain [--include-extended-apis=true] [--recursive=false] [flags] | Get documentation of various resources. For instance pods, nodes, services, etc. |
| expose | kubectl expose (-f FILENAME \| TYPE NAME \| TYPE/NAME) [--port=port] [--protocol=TCP\|UDP] [--target-port=number-or-name] [--name=name] [----external-ip=external-ip-of-service] [--type=type] [flags] | Expose a replication controller, service, or pod as a new Kubernetes service. |
| get | kubectl get (-f FILENAME \| TYPE [NAME \| /NAME \| -l label]) [--watch] [--sort-by=FIELD] [[-o \| --output]=OUTPUT\_FORMAT] [flags] | List one or more resources. |
| label | kubectl label (-f FILENAME \| TYPE NAME \| TYPE/NAME) KEY\_1=VAL\_1 ... KEY\_N=VAL\_N [--overwrite] [--all] [--resource-version=version] [flags] | Add or update the labels of one or more resources. |
| logs | kubectl logs POD [-c CONTAINER] [--follow] [flags] | Print the logs for a container in a pod. |
| patch | kubectl patch (-f FILENAME \| TYPE NAME \| TYPE/NAME) --patch PATCH [flags] | Update one or more fields of a resource by using the strategic merge patch process. |
| port-forward | kubectl port-forward POD [LOCAL\_PORT:]REMOTE\_PORT [...[LOCAL\_PORT\_N:]REMOTE\_PORT\_N] [flags] | Forward one or more local ports to a pod. |
| proxy | kubectl proxy [--port=PORT] [--www=static-dir] [--www-prefix=prefix] [--api-prefix=prefix] [flags] | Run a proxy to the Kubernetes API server. |
| replace | kubectl replace -f FILENAME | Replace a resource from a file or stdin. |
| rolling-update | kubectl rolling-update OLD\_CONTROLLER\_NAME ([NEW\_CONTROLLER\_NAME] --image=NEW\_CONTAINER\_IMAGE \| -f NEW\_CONTROLLER\_SPEC) [flags] | Perform a rolling update by gradually replacing the specified replication controller and its pods. |
| run | kubectl run NAME --image=image [--env="key=value"] [--port=port] [--replicas=replicas] [--dry-run=bool] [--overrides=inline-json] [flags] | Run a specified image on the cluster. |
| scale | kubectl scale (-f FILENAME \| TYPE NAME \| TYPE/NAME) --replicas=COUNT [--resource-version=version] [--current-replicas=count] [flags] | Update the size of the specified replication controller. |
| stop | kubectl stop | Deprecated: Instead, see kubectl delete. |
| version | kubectl version [--client] [flags] | Display the Kubernetes version running on the client and server. |

## Namespaces

Kubernetes uses namespaces to organize objects in the cluster. You can think of each namespace as a folder that holds a set of objects. By default, the kubectl command-line tool interacts with the default namespace. If you want to use a different namespace, you can pass kubectl the --namespace flag. For example, kubectl --namespace=mystuff references objects in the mystuff namespace.

## Contexts

If you want to change the default namespace more permanently, you can use a context. This gets recorded in a kubectl configuration file, usually located at $HOME/.kube/config. This configuration file also stores how to both find and authenticate to your cluster. For example, you can create a context with a different default namespace for your kubectl commands using:

There are three types of management techniques most commonly used in kubectl; these are known as Imperative Commands, Imperative Object Configuration, and Declarative Object Configuration.

Each management technique has its advantages and disadvantages, but it is highly recommended that a Kubernetes object is managed by a single method only. The use of multiple techniques on a single object at the same time will cause undefined behavior.

* **Imperative Commands** refers to "easier to use" commands that are easily remembered. They also require only one step to make a change to a cluster. Imperative commands are recommended for use in development projects, but they do also have downsides. These commands do not provide a great option for keeping records or integrating commands into change review processes, as the user is operating directly on live clusters. They also won’t provide a template for new objects.

eg: kubectl run podalabeltest --image=nginx --port=80 --labels="app=web,env=dev" --generator=run-pod/v1

* **Imperative Object Configuration** are kubectl configurations which specify operations or actions, i.e., replace, create, etc. This type of kubectl command will also specify optional flags and a file name. Each file must contain a complete definition of each object in the appropriate format to use imperative object configuration format. Imperative object configuration can be stored in a source control system and can also be integrated into change review processes. Additionally, these configurations will provide you with a template for new objects. This technique is simpler and more mature than declarative object configuration, but it is not great for working on directories.

eg: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

* **Declarative Object Configuration:** Users can adapt the live configuration values of a Kubernetes object, as part of a Kubernetes cluster. These are held in local Kubernetes cluster storage, usually etcd. The operations of create, update, and delete will be detected per-object by kubectl, enabling the ability to work on directories that may require different operations for different objects. Declarative object configuration uses kubectl apply only to repetitively create and update objects as necessary. This technique is preferred to imperative object configuration due to the fact that changes are retained, even when they are not merged with the configuration files. One disadvantage of this technique would be that it is harder to debug and understand results if they don’t come out as expected.

#### Generators

You can generate the following resources in kubectl run using --generator flag:

* Pod - use run-pod/v1.
* Replication controller - use run/v1.
* Deployment - use extensions/v1beta1 and for an endpoint - use deployment/v1beta1 (default).
* Deployment - use apps/v1beta1 and for an endpoint - use deployment/apps.v1beta1 (recommended).
* Job - use job/v1.
* CronJob - use batch/v1beta1and for an endpoint - use cronjob/v1beta1(default).
* CronJob - usebatch/v2alpha1 and for an endpoint - use cronjob/v2alpha1 (deprecated).