Chapter 4. Common kubectl Commands

The `kubectl` command-line utility is a powerful tool, and in the following chapters you will use it to create objects and interact with the Kubernetes API. Before that, however, it makes sense to go over the basic `kubectl` commands that apply to all Kubernetes objects.

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:

```
$ kubectl config set-context my-context --namespace=mystuff
```

This creates a new context, but it doesn't actually start using it yet. To use this newly created context, you can run:

```
$ kubectl config use-context my-context
```

Contexts can also be used to manage different clusters or different users for authenticating to those clusters using the `--users` or `--clusters` flags with the `set-context` command.

Viewing Kubernetes API Objects

Everything contained in Kubernetes is represented by a RESTful resource. Throughout this book, we refer to these resources as *Kubernetes objects*. Each Kubernetes object exists at a unique HTTP path; for example, *https://your-k8s.com/api/v1/namespaces/default/pods/my-pod* leads to the representation of a pod in the default namespace named `my-pod`. The `kubectl` command makes HTTP requests to these URLs to access the Kubernetes objects that reside at these paths.

The most basic command for viewing Kubernetes objects via `kubectl` is `get`. If you run *`kubectl get <resource-name>`* you will get a listing of all resources in the current namespace. If you want to get a specific resource, you can use *`kubectl get <resource-name> <object-name>`*.

By default, `kubectl` uses a human-readable printer for viewing the responses from the API server, but this human-readable printer removes many of the details of the objects to fit each object on one terminal line. One way to get slightly more information is to add the `-o wide` flag, which gives more details, on a longer line. If you want to view the complete object, you can also view the objects as raw JSON or YAML using the `-o json` or `-o yaml` flags, respectively.

A common option for manipulating the output of `kubectl` is to remove the headers, which is often useful when combining `kubectl` with Unix pipes (e.g., `kubectl … | awk …`). If you specify the `--no-headers` flag, `kubectl` will skip the headers at the top of the human-readable table.

Another common task is extracting specific fields from the object. `kubectl` uses the JSONPath query language to select fields in the returned object. The complete details of JSONPath are beyond the scope of this chapter, but as an example, this command will extract and print the IP address of the pod:

```
$ kubectl get pods my-pod -o jsonpath --template={.status.podIP}
```

If you are interested in more detailed information about a particular object, use the `describe` command:

```
$ kubectl describe <resource-name> <obj-name>
```

This will provide a rich multiline human-readable description of the object as well as any other relevant, related objects and events in the Kubernetes cluster.

Creating, Updating, and Destroying Kubernetes Objects

Objects in the Kubernetes API are represented as JSON or YAML files. These files are either returned by the server in response to a query or posted to the server as part of an API request. You can use these YAML or JSON files to create, update, or delete objects on the Kubernetes server.

Let's assume that you have a simple object stored in *obj.yaml*. You can use `kubectl` to create this object in Kubernetes by running:

```
$ kubectl apply -f obj.yaml
```

Notice that you don't need to specify the resource type of the object; it's obtained from the object file itself.

Similarly, after you make changes to the object, you can use the `apply` command again to update the object:

```
$ kubectl apply -f obj.yaml
```

> **NOTE**
>
> If you feel like making interactive edits, instead of editing a local file, you can instead use the `edit` command, which will download the latest object state, and then launch an editor that contains the definition:
>
> ```
> $ kubectl edit <resource-name> <obj-name>
> ```
>
> After you save the file, it will be automatically uploaded back to the Kubernetes cluster.

When you want to delete an object, you can simply run:

```
$ kubectl delete -f obj.yaml
```

But it is important to note that `kubectl` will not prompt you to confirm the delete. Once you issue the command, the object *will* be deleted.

Likewise, you can delete an object using the resource type and name:

```
$ kubectl delete <resource-name> <obj-name>
```

Labeling and Annotating Objects

Labels and annotations are tags for your objects. We'll discuss the differences in Chapter 6, but for now, you can update the labels and annotations on any Kubernetes object using the `annotate` and `label` commands. For example, to add the `color=red` label to a pod named `bar`, you can run:

```
$ kubectl label pods bar color=red
```

The syntax for annotations is identical.

By default, `label` and `annotate` will not let you overwrite an existing label. To do this, you need to add the `--overwrite` flag.

If you want to remove a label, you can use the `-<label-name>` syntax:

```
$ kubectl label pods bar -color
```

This will remove the `color` label from the pod named `bar`.

Debugging Commands

`kubectl` also makes a number of commands available for debugging your containers. You can use the following to see the logs for a running container:

```
$ kubectl logs <pod-name>
```

If you have multiple containers in your pod you can choose the container to view using the `-c` flag.

By default, `kubectl logs` lists the current logs and exits. If you instead want to continuously stream the logs back to the terminal without exiting, you can add the `-f` (follow) command-line flag.

You can also use the `exec` command to execute a command in a running container:

```
$ kubectl exec -it <pod-name> -- bash
```

This will provide you with an interactive shell inside the running container so that you can perform more debugging.

Finally, you can copy files to and from a container using the `cp` command:

```
$ kubectl cp <pod-name>:/path/to/remote/file /path/to/local/file
```

This will copy a file from a running container to your local machine. You can also specify directories, or reverse the syntax to copy a file from your local machine back out into the container.

Summary

`kubectl` is a powerful tool for managing your applications in your Kubernetes cluster. This chapter has illustrated many of the common uses for the tool, but `kubectl` has a great deal of built-in help available. You can start viewing this help with:

```
kubectl help
```

or:

```
kubectl help command-name
```