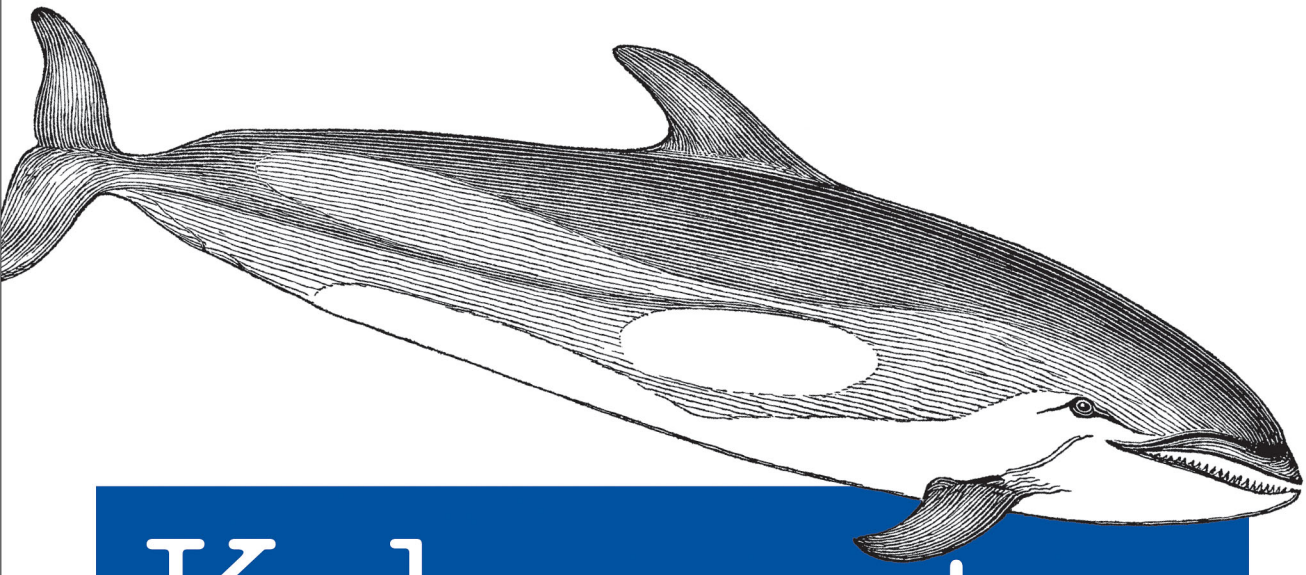
O'REILLY®



Kubernetes Up & Running

DIVE INTO THE FUTURE OF INFRASTRUCTURE

Kelsey Hightower,
Brendan Burns & Joe Beda

Kubernetes: Up and Running

Dive into the Future of Infrastructure

This Excerpt contains Chapter 4 of *Kubernetes: Up and Running*. The final book is available on Safari and through other retailers.

Kelsey Hightower, Brendan Burns, and Joe Beda

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Kubernetes: Up and Running
by Kelsey Hightower, Brendan Burns, and Joe Beda

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Angela Rufino
Production Editor: Melanie Yarbrough
Copyeditor: Christina Edwards
Proofreader: Rachel Head
Indexer: Kevin Broccoli
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

September 2017: First Edition

Revision History for the First Edition
2017-09-05: First Release
2018-02-02: Second Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781491935675* for release details.

978-1-491-93567-5

[LSI]

Table of Contents

CHAPTER 1

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
```

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
```

About the Authors

Kelsey Hightower has worn every hat possible throughout his career in tech, and enjoys leadership roles focused on making things happen and shipping software. Kelsey is a strong open source advocate focused on building simple tools that make people smile. When he is not slinging Go code, you can catch him giving technical workshops covering everything from programming to system administration.

Joe Beda started his career at Microsoft working on Internet Explorer (he was young and naive). Throughout 7 years at Microsoft and 10 at Google, Joe has worked on GUI frameworks, real-time voice and chat, telephony, machine learning for ads, and cloud computing. Most notably, while at Google, Joe started the Google Compute Engine and, along with Brendan and Craig McLuckie, created Kubernetes. Joe is now CTO of Heptio, a startup he founded along with Craig. Joe proudly calls Seattle home.

Brendan Burns began his career with a brief stint in the software industry followed by a PhD in Robotics focused on motion planning for human-like robot arms. This was followed by a brief stint as a professor of computer science. Eventually, he returned to Seattle and joined Google, where he worked on web search infrastructure with a special focus on low-latency indexing. While at Google, he created the Kubernetes project with Joe and Craig McLuckie. Brendan is currently a Director of Engineering at Microsoft Azure.

Colophon

The animal on the cover of *Kubernetes: Up and Running* is an Atlantic white-sided dolphin (*Lagenorhynchus acutus*). As its name suggests, the white-sided dolphin has light patches on its sides and a light gray strip that runs above the eye to below the dorsal fin. It is among the largest species of oceanic dolphins, and ranges throughout the north Atlantic Ocean. It prefers open water, so it is not often seen from the shore, but will readily approach boats and perform various acrobatic feats.

White-sided dolphins are social animals commonly found in large groups (known as pods) of about 60 individuals, though the size will vary depending on location and the availability of food. Dolphins often work as a team to harvest schools of fish, but they also hunt individually. They primarily search for prey using echolocation, which is similar to sonar. The bulk of this marine mammal's diet consists of herring, mackerel, and squid.

The average lifespan of the white-sided dolphin is between 22–27 years. Females only mate every 2–3 years, and the gestation period is 11 months. Calves are typically born in June or July, and are weaned after 18 months. Dolphins have very great intelligence

and display complex social behaviors like grieving, cooperation, and problem-solving, due to their high brain-to-body ratio (the highest among aquatic mammals).

Many of the animals on O'Reilly covers are endangered; all of them are important to the world. To learn more about how you can help, go to *animals.oreilly.com*.

The cover image is from *British Quadrupeds*. The cover fonts are URW Typewriter and Guardian Sans. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.