Opensource.com

Getting started with Kubernetes

Find out how Kubernetes is an elegant solution to a wide range of essential business problems.

ABOUT OPENSOURCE.COM

What is Opensource.com?

OPENSOURCE.COM publishes stories about creating, adopting, and sharing open source solutions. Visit Opensource.com to learn more about how the open source way is improving technologies, education, business, government, health, law, entertainment, humanitarian efforts, and more.

Submit a story idea: https://opensource.com/story

Email us: open@opensource.com

Supported by Red Hat

SCOTT MCCARTY

SCOTT MCCARTY AT RED HAT, SCOTT MCCARTY IS TECHNICAL PRODUCT MANAGER for the container subsystem team, which enables key product capabilities in OpenShift Container Platform and Red Hat Enterprise Linux. Focus areas includes container runtimes, tools, and images. Working closely with engineering teams, at both a product and upstream project level, he combines personal experience with customer and partner feedback to enhance and tailor strategic container features and capabilities.

Scott is a social media start-up veteran, an e-commerce old timer, and a weathered government research technologist, with experience across a variety of companies and organizations, from seven person startups to 12,000 employee technology companies. This has culminated in a unique perspective on open source software development, delivery, and maintenance.

FOLLOW SCOTT MCCARTY

Twitter: https://twitter.com/fatherlinux

Kubernetes is a dump truck: Here's why

Dump trucks are an elegant solution to a wide range of essential business problems.

DUMP TRUCKS ARE ELEGANT. Seriously, stay with me for a minute. They solve a wide array of technical problems in an elegant way. They can move dirt, gravel, rocks, coal, construction material, or road barricades. They can even pull trailers with other pieces of heavy equipment on them. You can load a dump truck with five tons of dirt and drive across the country with it. For a nerd like me, that's elegance.

But, they're not easy to use. Dump trucks require a special driver's license. They're also not easy to equip or maintain. There are a ton of options when you buy a dump truck and a lot of maintenance. But, they're elegant for moving dirt.

You know what's not elegant for moving dirt? A late-model, compact sedan. They're way easier to buy. Easier to drive. Easier to maintain. But, they're terrible at carrying dirt. It would take 200 trips to carry five tons of dirt, and nobody would want the car after that.

Alright, you're sold on using a dump truck, but you want to build it yourself. I get it. I'm a nerd and I love building things. But…

If you owned a construction company, you wouldn't build your own dump trucks. You definitely wouldn't maintain the supply chain to rebuild dump trucks (that's a big supply chain). But you would learn to drive one.

OK, my analogy is crude but easy to understand. Ease of use is relative. Ease of maintenance is relative. Ease of configuration is relative. It really depends on what you are trying to do. Kubernetes [1] is no different.

Building Kubernetes once isn't too hard. Equipping Kubernetes? Well, that gets harder. What did you think of KubeCon? How many new projects were announced? Which ones are "real"? Which ones should you learn? How deeply do you understand Harbor, TikV, NATD, Vitess, Open Policy Agent? Not to mention Envoy, eBPF, and a bunch of the underlying technologies in Linux? It feels a lot like building dump trucks in 1904 with the industrial revolution in full swing. Figuring out what screws, bolts, metal, and pistons to use. (Steampunk, anyone?)

Building and equipping Kubernetes, like a dump truck, is a technical problem you probably shouldn't be tackling if you are in financial services, retail, biological research, food services, and so forth. But, learning how to drive Kubernetes is definitely something you should be learning.

Kubernetes, like a dump truck, is elegant for the wide variety of technical problems it can solve (and the ecosystem it drags along). So, I'll leave you with a quote that one of my computer science professors told us in my first year of college. She said, "one day, you will look at a piece of code and say to yourself, 'now that's elegant!'"

Kubernetes is elegant.

Links

[1] https://kubernetes.io/

How to navigate the **Kubernetes learning curve**

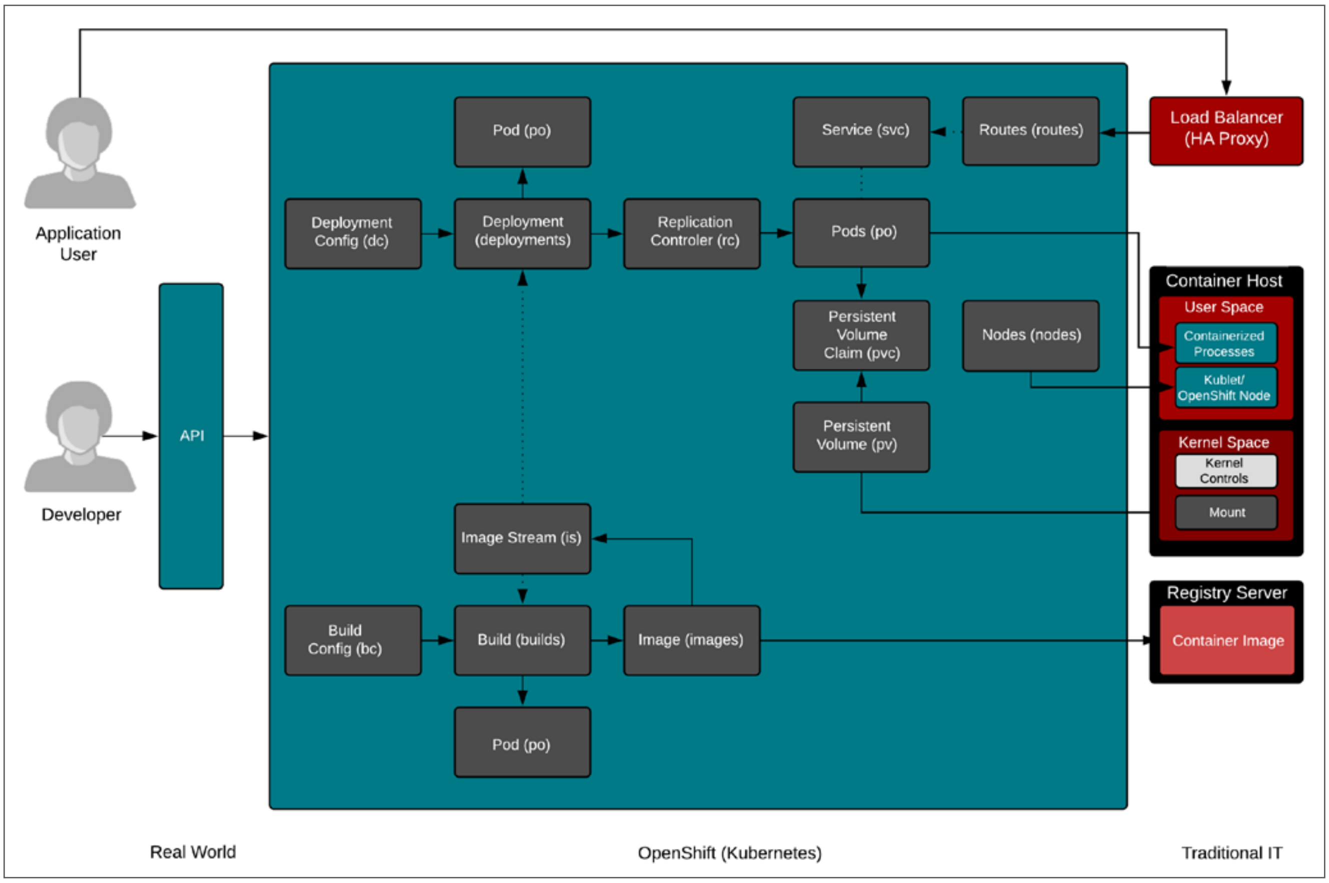
Kubernetes is like a dump truck. It's elegant for solving the problems it's designed for, but you have to master the learning curve first.

THE JOURNEY TO KUBERNETES [1] often starts with running one container on one host. You quickly discover how easy it is to run new versions of software, how easy it is to share that software with others, and how easy it is for those users to run it the way you intended.

But then you need

- Two containers
- Two hosts

It's easy to fire up one web server on port 80 with a container, but what happens when you need to fire up a second container on port 80? What happens when you are building a production environment and you need the containerized web server to fail over to a second host?



Kubernetes business model

The short answer, in either case, is you have to move into container orchestration.

Inevitably, when you start to handle the two containers or two hosts problem, you'll introduce complexity and, hence, a learning curve. The two services (a more generalized version of a container) / two hosts problem has been around for a long time and has always introduced complexity.

Historically, this would have involved load balancers, clustering software, and even clustered file systems. Configuration logic for every service is embedded in every system (load balancers, cluster software, and file systems). Running 60 or 70 services, clustered, behind load balancers is complex. Adding another new service is also complex. Worse, decommissioning a service is a nightmare. Thinking back on my days of troubleshooting production MySQL and Apache servers with logic embedded in three, four, or five different places, all in different formats, still makes my head hurt.

Kubernetes elegantly solves all these problems with one piece of software:

1. Two services (containers): Check
2. Two servers (high availability): Check
3. Single source of configuration: Check
4. Standard configuration format: Check
5. Networking: Check
6. Storage: Check
7. Dependencies (what services talk to what databases): Check
8. Easy provisioning: Check
9. Easy de-provisioning: Check (perhaps Kubernetes' *most* powerful piece)

Wait, it's starting to look like Kubernetes is pretty elegant and pretty powerful. *It is*. You can model an entire miniature IT universe in Kubernetes.

So yes, there is a learning curve when starting to use a giant dump truck (or any professional equipment). There's also a learning curve to use Kubernetes, but it's worth it because you can solve so many problems with one tool. If you are apprehensive about the learning curve, think through all the underlying networking, storage, and security problems in IT infrastructure and envision their solutions today—they're not easier. Especially when you introduce more and more services, faster and faster. Velocity is the goal nowadays, so give special consideration to the provisioning and de-provisioning problem.

But don't confuse the learning curve for building or equipping Kubernetes (picking the right mud flaps for your dump truck can be hard, LOL) with the learning curve for using it. Learning to build your own Kubernetes with so many different choices at so many different layers (container engine, logging, monitoring, service mesh, storage, networking), and then maintaining updated selections of each component every six months, might not be worth the investment—but learning to use it is absolutely worth it.

I eat, sleep, and breathe Kubernetes and containers every day, and even I struggle to keep track of all the major new projects announced literally almost every day. But there isn't a day that I'm not excited about the operational benefits of having a single tool to model an entire IT mini-verse. Also, remember Kubernetes has matured a ton and will continue to do so. Like Linux and OpenStack before it, the interfaces and de facto projects at each layer will mature and become easier to select.

Links

[1] https://kubernetes.io/

Kubernetes basics: Learn how to drive first

Quit focusing on new projects and get focused on getting your Kubernetes dump truck commercial driver's license.

RECENTLY, I SAW A THREAD ON REDDIT about essential Kubernetes projects [1]. People seem hungry to know the bare minimum they should learn to get started with Kubernetes. The "driving a dump truck analogy" helps frame the problem to stay on track. Someone in the thread mentioned that you shouldn't be running your own registry unless you have to, so people are already nibbling at this idea of driving Kubernetes instead of building it.

The API is Kubernetes' engine and transmission. Like a dump truck's steering wheel, clutch, gas, and brake pedal, the YAML or JSON files you use to build your applications are the primary interface to the machine. When you're first learning Kubernetes, this should be your primary focus. Get to know your controls. Don't get sidetracked by all the latest and greatest projects. Don't try out an experimental dump truck when you are just learning to drive. Instead, focus on the basics.

Defined and actual states

First, Kubernetes works on the principles of defined state and actual state.

Humans (developers/sysadmins/operators) specify the defined state using the YAML/JSON files they submit to the Kubernetes API. Then, Kubernetes uses a controller to analyze the difference between the new state defined in the YAML/JSON and the actual state in the cluster.

In the example "Defined state and actual state," the Replication Controller sees the difference between the three pods specified by the user, with one Pod running, and schedules two more. If you were to log into Kubernetes and manually kill one of the Pods, it would start another one to replace it—over and over and over. Kubernetes does not stop until the actual state matches the defined state. This is super powerful.

Primitives

Next, you need to understand what primitives you can specify in Kubernetes.

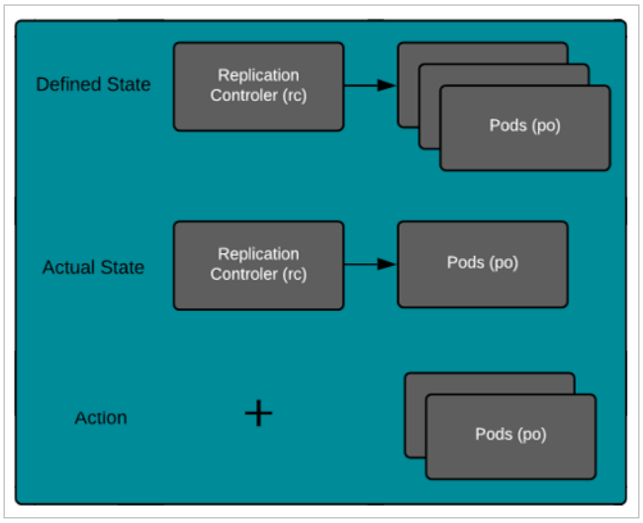
It's more than just Pods; it's Deployments, Persistent Volume Claims, Services, routes, etc. With Kubernetes platform OpenShift [2], you can add builds and BuildConfigs. It will take you a day or so to get decent with each of these primitives. Then you can dive in deeper as your use cases become more complex.

Mapping developer-native to traditional IT environments

Finally, start thinking about how this maps to what you do in a traditional IT environment.

The user has always been trying to solve a business problem, albeit a technical one. Historically, we have used things like playbooks to tie business logic to sets of IT systems with a single language. This has always been great for operations staff, but it gets hairier when you try to extend this to developers.

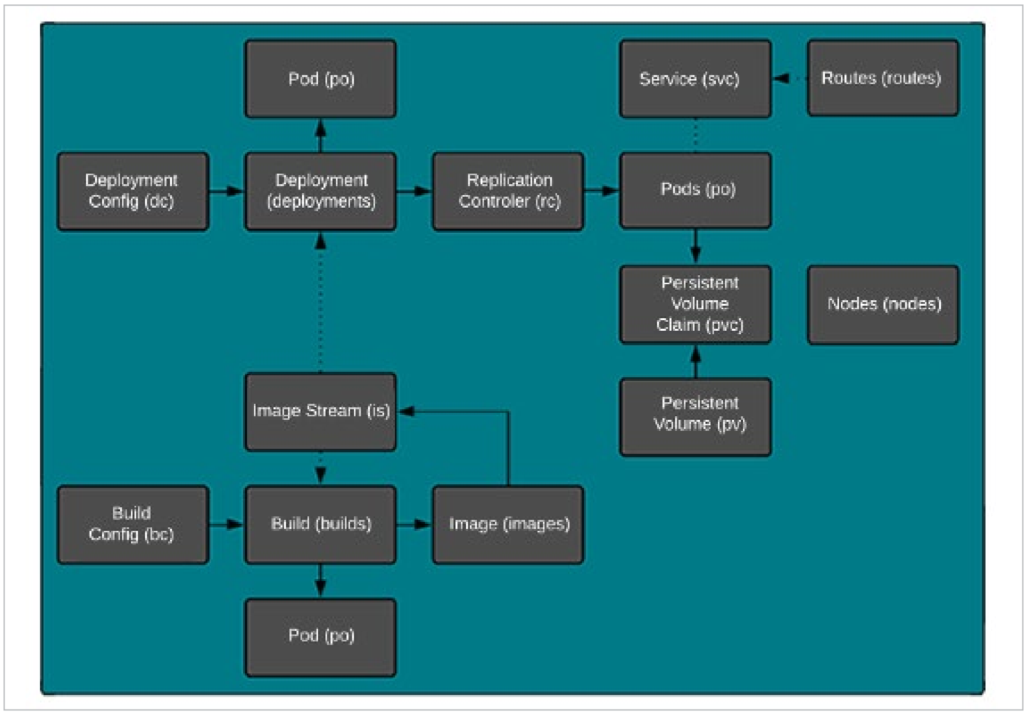
We have never been able to truly specify how a set of IT systems should behave and interact together, in a de-



Defined state and actual state

veloper-native way, until Kubernetes. If you think about it, we are extending the ability to manage storage, network, and compute resources in a very portable and declarative way with the YAML/JSON files we write in Kubernetes, but they are always mapped back to "real" resources somewhere. We just don't have to worry about it in developer mode.

So, quit focusing on new projects in the Kubernetes ecosystem and get focused on driving it.

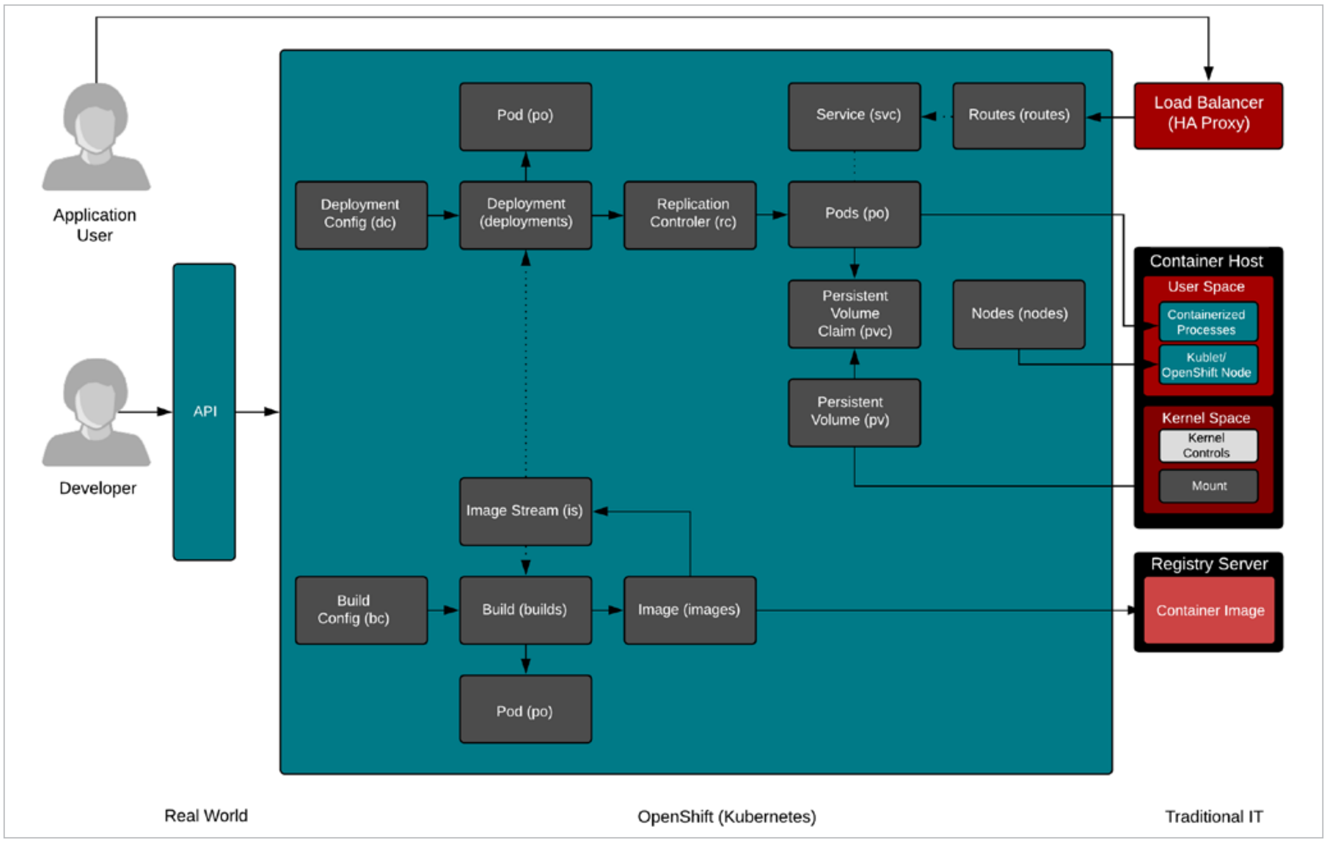


Kubernetes primitives

Links

[1] https://www.reddit.com/r/kubernetes/comments/bsoixc/what_are_the_essential_kubernetes_related/

[2] https://www.openshift.com/



Mapping developer-native to traditional IT environments

4 tools to help you drive Kubernetes

Learning to drive Kubernetes is more important that knowing how to build it, and these tools will get you on the road fast.

I WANT TO EMPHASIZE this point: the set of primitives that you need to learn are the easiest set of primitives that you can learn to achieve production-quality application deployments (i.e., high-availability [HA], multiple containers, multiple applications). Stated another way, learning the set of primitives built into Kubernetes is easier than learning clustering software, clustered file systems, load balancers, crazy Apache configs, crazy Nginx configs, routers, switches, firewalls, and storage backends—all the things you would need to model a simple HA application in a traditional IT environment (for virtual machines or bare metal).

In this article, I'll share some tools that will help you learn to drive Kubernetes quickly.

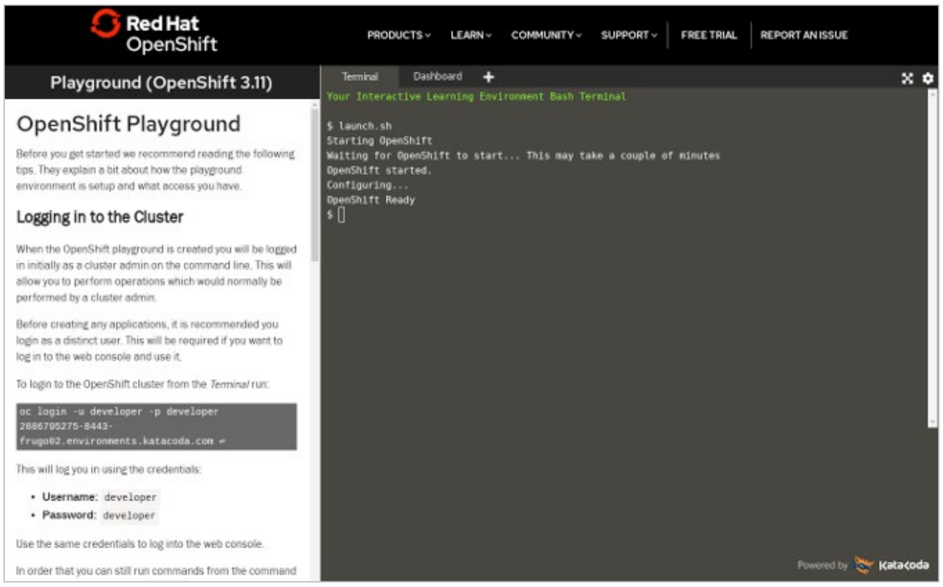
1. Katacoda

Katacoda [1] is the easiest way to test-drive a Kubernetes cluster, hands-down. With one click and five seconds of time, you have a web-based terminal plumbed straight into a running Kubernetes cluster. It's magnificent for playing and learning. I even use it for demos and testing out new ideas. Katacoda provides a completely ephemeral environment that is recycled when you finish using it.

Katacoda provides ephemeral playgrounds and deeper lab environments. For example, the Linux Container Internals Lab [4], which I have run for the last three or four years, is built in Katacoda.

Katacoda maintains a bunch of Kubernetes and cloud tutorials [5] on its main site and collaborates with Red Hat to support a dedicated learning portal for OpenShift [6]. Explore them both—they are excellent learning resources.

When you first learn to drive a dump truck, it's always best to watch how other people drive first.



Playground (OpenShift 3.11)

OpenShift Playground

Before you get started we recommend reading the following tips. They explain a bit about how the playground environment is setup and what access you have.

Logging in to the Cluster

When the OpenShift playground is created you will be logged in initially as a cluster admin on the command line. This will allow you to perform operations which would normally be performed by a cluster admin.

Before creating any applications, it is recommended you login as a distinct user. This will be required if you want to log in to the web console and use it.

To login to the OpenShift cluster from the *Terminal* run:

```
oc login -u developer -p developer
2886795275-8443-
frugo02.environments.katacoda.com ↵
```

This will log you in using the credentials:

- **Username:** `developer`
- **Password:** `developer`

Use the same credentials to log into the web console.

In order that you can still run commands from the command

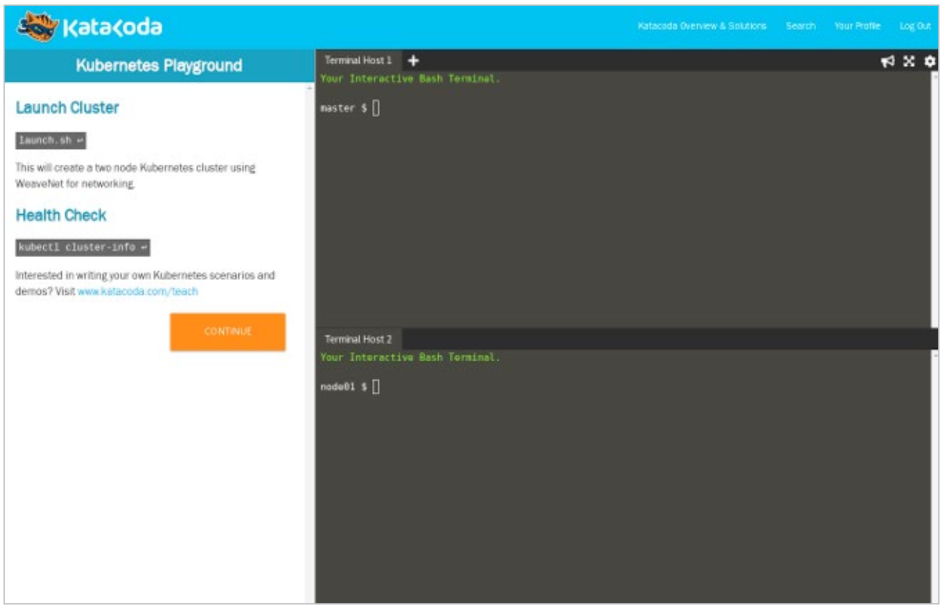
```
Your Interactive Learning Environment Bash Terminal

$ launch.sh
Starting OpenShift
Waiting for OpenShift to start... This may take a couple of minutes
OpenShift started.
Configuring...
OpenShift Ready
$ 
```

OpenShift Playground [2]

2. Podman generate kube

The **podman generate kube** command is a brilliant little subcommand that helps users naturally transition from a simple container engine running simple containers to a cluster use case running many containers (as I described in the last chapter). Podman [7] does this by letting you start a few containers, then exporting the working Kube YAML, and firing them up in Kubernetes. Check this out (pssst, you can run it in this Katacoda lab [8], which already has Podman and OpenShift).

First, notice the syntax to run a container is strikingly similar to Docker:



Launch Cluster

```
launch.sh
```

This will create a two node Kubernetes cluster using WeaveNet for networking.

Health Check

```
kubectl cluster-info
```

Interested in writing your own Kubernetes scenarios and demos? Visit www.katacoda.com/teach

```
Your Interactive Bash Terminal.

master $
```

```
Your Interactive Bash Terminal.

node01 $
```

Kubernetes Playground [3]

```
podman run -dtn two-pizza  quay.io/fatherlinux/two-pizza
```

But this is something other container engines don't do:

```
podman generate kube two-pizza
```

The output:

```
# Generation of Kubernetes YAML is still under development!
#
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-1.3.1
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-06-07T08:08:12Z"
  labels:
    app: two-pizza
  name: two-pizza
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - bash -c 'while true; do /usr/bin/nc -l -p 3306
      < /srv/hello.txt; done'
    env:
    - name: PATH
      value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
             sbin:/bin
    - name: TERM
      value: xterm
    - name: HOSTNAME
    - name: container
      value: oci
    image: quay.io/fatherlinux/two-pizza:latest
    name: two-pizza
    resources: {}
    securityContext:
      allowPrivilegeEscalation: true
      capabilities: {}
      privileged: false
      readOnlyRootFilesystem: false
    tty: true
    workingDir: /
status: {}
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2019-06-07T08:08:12Z"
  labels:
    app: two-pizza
  name: two-pizza
spec:
  selector:
    app: two-pizza
  type: NodePort
status:
  loadBalancer: {}
```

You now have some working Kubernetes YAML, which you can use as a starting point for mucking around and learning, tweaking, etc. The **-s** flag created a service for you. Brent Baude [9] is even working on new features like adding Volumes/Persistent Volume Claims [10]. For a deeper dive, check out Brent's amazing work in his blog post "Podman can now ease the transition to Kubernetes and CRI-O." [11]

3. oc new-app

The **oc new-app** command is extremely powerful. It's OpenShift-specific, so it's not available in default Kubernetes, but it's really useful when you're starting to learn Kubernetes. Let's start with a quick command to create a fairly sophisticated application:

```
oc new-project -n example
oc new-app -f https://raw.githubusercontent.com/openshift/
   origin/master/examples/quickstarts/cakephp-mysql.json
```

With **oc new-app**, you can literally steal templates from the OpenShift developers and have a known, good starting point when developing primitives to describe your own applications. After you run the above command, your Kubernetes namespace (in OpenShift) will be populated by a bunch of new, defined resources.

```
oc get all
```

See the "occ new-app output":

The beauty of this is that you can delete Pods, watch the replication controllers recreate them, scale Pods up, and scale them down. You can play with the template and change it for other applications (which is what I did when I first started).

4. Visual Studio Code

I saved one of my favorites for last. I use vi [12] for most of my work, but I have never found a good syntax highlighter and code completion plugin for Kubernetes (if you have one, let me know). Instead, I have found that Microsoft's

occ new-app output

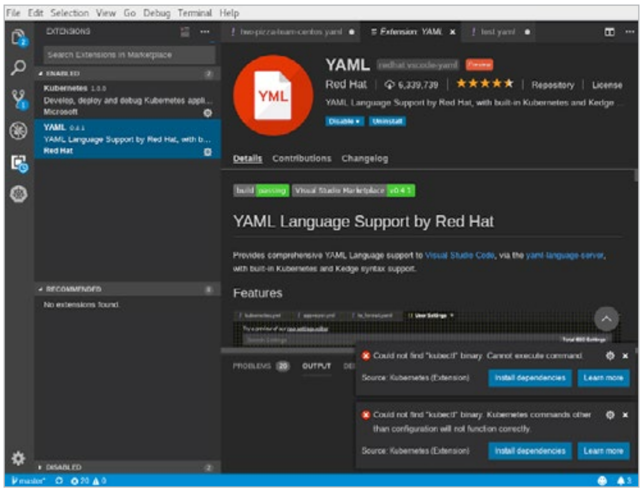
```
NAME                                READY     STATUS      RESTARTS   AGE
pod/cakephp-mysql-example-1-build   0/1       Completed   0          4m
pod/cakephp-mysql-example-1-gz65l   1/1       Running     0          1m
pod/mysql-1-nkhqn                   1/1       Running     0          4m

NAME                                            DESIRED   CURRENT   READY     AGE
replicationcontroller/cakephp-mysql-example-1   1         1         1         1m
replicationcontroller/mysql-1                   1         1         1         4m

NAME                            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/cakephp-mysql-example   ClusterIP   172.30.234.135   <none>        8080/TCP   4m
service/mysql                   ClusterIP   172.30.13.195    <none>        3306/TCP   4m

NAME                                                       REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfig.apps.openshift.io/cakephp-mysql-example  1          1         1         config,image(cakephp-mysql-example:latest)
deploymentconfig.apps.openshift.io/mysql                  1          1         1         config,image(mysql:5.7)

NAME                                                   TYPE      FROM      LATEST
buildconfig.build.openshift.io/cakephp-mysql-example  Source    Git       1

NAME                                               TYPE      FROM          STATUS     STARTED         DURATION
build.build.openshift.io/cakephp-mysql-example-1   Source    Git@47a951e   Complete   4 minutes ago   2m27s

NAME                                                   DOCKER REPO                                                      TAGS
UPDATED
imagestream.image.openshift.io/cakephp-mysql-example   docker-registry.default.svc:5000/example/cakephp-mysql-example   latest
  About aminute ago

NAME                                             HOST/PORT
PATH   SERVICES                PORT      TERMINATION   WILDCARD
route.route.openshift.io/cakephp-mysql-example   cakephp-mysql-example-example.2886795271-80-rhsummit1.environments.katacoda.com
  cakephp-mysql-example   <all>                   None
```

VS Code [13] has a killer set of plugins that complete the creation of Kubernetes resources and provide boilerplate.

First, install Kubernetes and YAML plugins shown in the "VS Code plugins UI" image.

Then, you can create a new YAML file from scratch and get auto-completion of Kubernetes resources. The example "Autocomplete in VS Code" shows a Service.

When you use autocomplete and select the Service resource, it fills in some boilerplate for the object. This is magnificent when you are first learning to drive Kubernetes. You can build Pods, Services, Replication Controllers, Deployments, etc. This is a really nice feature when you are building these files from scratch or even modifying the files you create with **Podman generate kube**.



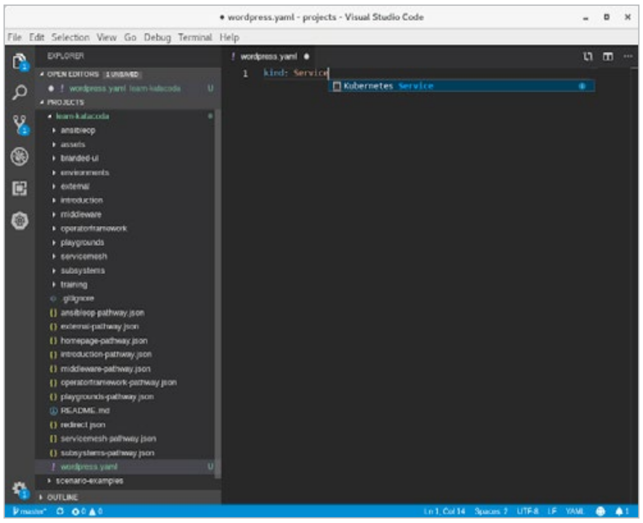
YAML Language Support by Red Hat

Provides comprehensive YAML Language support to Visual Studio Code, via the yaml language-server, with built-in Kubernetes and Kedge syntax support.

Could not find "kubectl" binary. Cannot execute command.
Source: Kubernetes (Extension)

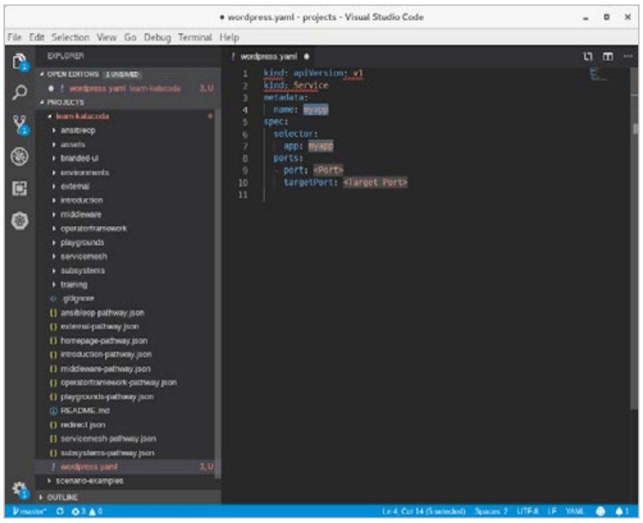
Could not find "kubectl" binary. Kubernetes commands other than configuration will not function correctly.
Source: Kubernetes (Extension)

VS Code plugins UI



```
kind: Service
```

Autocomplete in VS Code



```
kind: apiVersion: v1
kind: Service
metadata:
  name: myapp
spec:
  selector:
    app: myapp
  ports:
  - port: <Port>
    targetPort: <Target Port>
```

VS Code autocomplete filling in boilerplate for an object

Conclusion

These four tools (six if you count the two plugins) will help you learn to drive Kubernetes, instead of building or equipping it.

Links

[1] https://learn.openshift.com/subsystems/container-internals-lab-2-0-part-1
[2] https://learn.openshift.com/playgrounds/openshift311/
[3] https://katacoda.com/courses/kubernetes/playground
[4] https://learn.openshift.com/subsystems/container-internals-lab-2-0-part-1
[5] https://katacoda.com/learn
[6] http://learn.openshift.com/
[7] https://podman.io/
[8] https://learn.openshift.com/subsystems/container-internals-lab-2-0-part-1
[9] https://developers.redhat.com/blog/author/bbaude/
[10] https://github.com/containers/libpod/issues/2303
[11] https://developers.redhat.com/blog/2019/01/29/podman-kubernetes-yaml/
[12] https://en.wikipedia.org/wiki/Vi
[13] https://code.visualstudio.com/

Why containers and Kubernetes have the potential to run almost anything

Go beyond deployment of simple applications and tackle day two operations with Kubernetes Operators.

FROM THE BEGINNING, KUBERNETES has been able to run web-based workloads (containerized) really well. Workloads like web servers, Java, and associated app servers (PHP, Python, etc) just work. The supporting services like DNS, load balancing, and SSH (replaced by kubectl exec) are handled by the platform. For the majority of my career, these are the workloads I ran in production, so I immediately recognized the power of running production workloads with Kubernetes, aside from DevOps, aside from agile. There is incremental efficiency gain even if we barely change our cultural practices. Commissioning and decommissioning become extremely easy, which were terribly difficult with traditional IT. So, since the early days, Kubernetes has given me all of the basic primitives I need to model a production workload, in a single configuration language (Kube YAML/Json).

But, what happened if you needed to run Multi-master MySQL with replication? What about redundant data using Galera? How do you do snapshotting and backups? What about sophisticated workloads like SAP? Day zero (deployment) with simple applications (web servers, etc) has been fairly easy with Kubernetes, but day two operations and workloads were not tackled. That's not to say that day two operations with sophisticated workloads were harder than traditional IT to solve, but they weren't made easier with Kubernetes. Every user was left to devise their own genius ideas for solving these problems, which is basically the status quo today. Over the last 5 years, the number one type of question I get is around day two operations of complex workloads.

Thankfully, that's changing as we speak with the advent of Kubernetes Operators. With the advent of Operators, we now have a framework to codify day two operations knowledge into the platform. We can now apply the same defined state, actual state methodology that I described in Kubernetes basics: Learn how to drive first [1]—we can now define, automate, and maintain a wide range of systems administration tasks.

I often refer to Operators as "Robot Sysadmins" because they essentially codify a bunch of the day two operations knowledge that a subject matter expert (SME, like database administrator or, systems administrator) for that workload type (database, web server, etc) would normally keep in their notes somewhere in a wiki. The problem with these notes being in a wiki is, for the knowledge to be applied to solve a problem, we need to:

1. Generate an event, often a monitoring system finds a fault and we create a ticket
2. Human SME has to investigate the problem, even if it's something we've seen a million times before
3. Human SME has to execute the knowledge (perform the backup/restore, configure the Galera or transaction replication, etc)

With Operators, all of this SME knowledge can be embedded in a separate container image which is deployed before the actual workload. We deploy the Operator container, and then the Operator deploys and manages one or more instances of the workload. We then manage the Operators using something like the Operator Lifecycle Manager (Katacoda tutorial).

So, as we move forward with Kubernetes, we not only simplify the deployment of applications, but also the management over the lifecycle. Operators also give us the tools to manage very complex, stateful applications with deep configuration requirements (clustering, replication, repair, backup/restore. And, the best part is, the people who built the container are probably the subject matter experts for day two operations, so now they can embed that knowledge into the operations environment.

The conclusion to this e-book

The future of Kubernetes is bright, and like virtualization before it, workload expansion is inevitable. Learning how to drive Kubernetes is probably the biggest investment that a developer or sysadmin can make in their own career growth. As the workloads expand, so will the career opportunities. So, here's to driving an amazing dump truck that's very elegant at moving dirt... [2]

Links

[1] https://opensource.com/article/19/6/kubernetes-basics
[2] https://opensource.com/article/19/6/kubernetes-dump-truck

WRITE FOR US

In 2010, Red Hat CEO Jim Whitehurst announced the launch of Opensource.com in a post titled *Welcome to the conversation on Opensource.com*. He explained, “This site is one of the ways in which Red Hat gives something back to the open source community. Our desire is to create a connection point for conversations about the broader impact that open source can have—and is having—even beyond the software world.” he wrote, adding, “All ideas are welcome, and all participants are welcome. This will not be a site for Red Hat, about Red Hat. Instead, this will be a site for open source, about the future.”

By 2013, Opensource.com was publishing an average of 46 articles per month, and in March 2016, Opensource.com surpassed 1-million page views for the first time. In 2019, Opensource.com averages more than 1.5 million page views and 90 articles per month.

More than 60% of our content is contributed by members of open source communities, and additional articles are written by the editorial team and other Red Hat contributors. A small, international team of staff editors and Community Moderators work closely with contributors to curate, polish, publish, and promote open source stories from around the world.

Would you like to write for us? Send pitches and inquiries to open@opensource.com.

To learn more, read 7 big reasons to contribute to Opensource.com.