From Docker Compose to Kubernetes with Podman

Use Podman 3.0 to convert Docker Compose YAML to a format Podman recognizes.

Posted: January 26, 2021 | Brent Baude (Red Hat), Urvashi Mohnani (Red Hat)

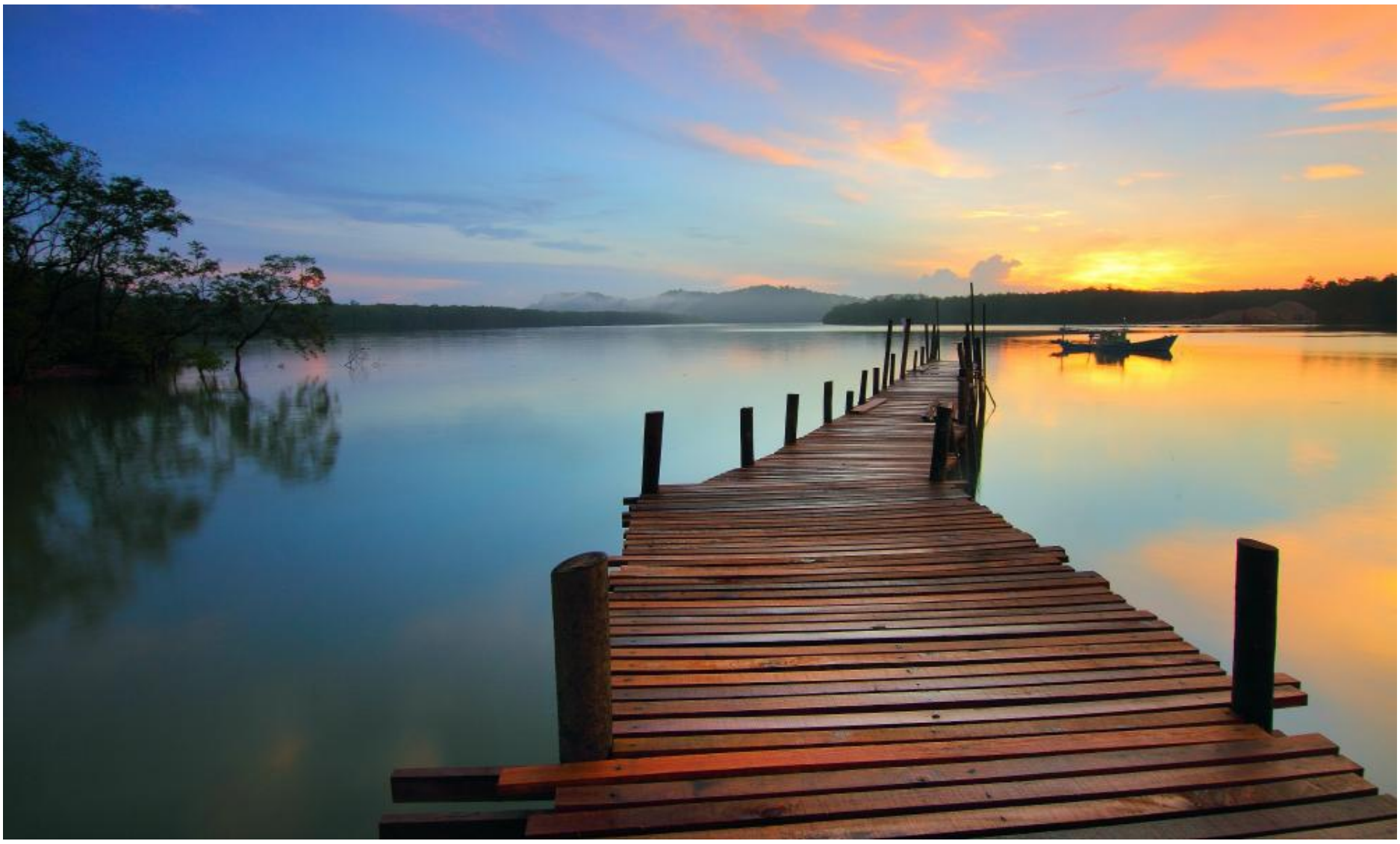

*Photo by **Pok Rie** from **Pexels***

The Docker Compose tool has been valuable for many people who have been working with containers. According to the documentation, Docker Compose describes itself as:

> ... a tool for defining and running multi-container applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

One challenge with Docker Compose is that the YAML file format only works with the Docker engine. While you can give it to other Docker users for local replication, they cannot use it with other container runtimes. That is, until now.

[Related tutorial: Using Podman and Docker Compose]

With Podman 3.0, the Docker Compose works with a Podman backend

The real power of Podman shines through by easily converting the containers based on Docker Compose to a Kubernetes YAML file. Like Docker Compose, Podman can use the Kubernetes YAML file to replicate the containers locally. More importantly, this allows Podman to have an orchestrated pod and service that can be run on many platforms, including Kubernetes/OpenShift or minikube.

This article explains the process of starting with a simple Compose file that runs WordPress using two containers. The source for this Compose file is published on GitHub in Docker's awesome-compose repository. Successful use of this file with Podman results in the WordPress initial setup screen appearing in a browser.

Note: At the time of this writing, we only support the `docker-compose` command running rootfully.

Start Podman's system service

To use Compose, the first step is to make sure that all the required packages are installed and then to set up the Podman (3.0 or greater) system service using `systemd`. After installing packages, enable and start the Podman `systemd` socket-activated service using the following command:

```
$ sudo systemctl enable --now podman.socket
```

Verify the service is running by hitting the ping endpoint. This step needs to be successful before proceeding further.

```
$ sudo curl -H "Content-Type: application/json" --unix-socket
/var/run/docker.sock http://localhost/_ping
OK
```

You can now confidently run Compose knowing the RESTful API is working.

Run Compose

As mentioned earlier, the example will run a Compose file consisting of two containers to bring up a WordPress session. One container runs an Apache web service, and the other stores the data in a MySQL database. The two containers communicate via TCP/IP over a network dedicated to this Compose instance. To bring up the containers, run `docker-compose up`.

```
$ sudo docker-compose up -d
Creating network "wordpress-mysql_default" with the default driver
Creating volume "wordpress-mysql_db_data" with default driver
Pulling db (mysql:8.0.19)...
0c27e8e5fcfab7805cfed996b55e5e98f43fd7ee76e1516f20cba139c6a299c5: pulling image
() from docker.io/library/mysql:8.0.19
Pulling wordpress (wordpress:latest)...
0d35c2300ec845fda141ba012f7c6dccde8f0ae106b8f4bb0fcfced69380f851: pulling image
() from docker.io/library/wordpress:latest
Creating wordpress-mysql_db_1        ... done
Creating wordpress-mysql_wordpress_1 ... done
```

Use the `podman ps` command to verify that two containers have been created and are now running. No Docker daemon was necessary.

```
$ sudo podman ps
CONTAINER ID  IMAGE                                COMMAND               CREATED
        STATUS             PORTS                 NAMES
a089a40bb9ae  docker.io/library/mysql:8.0.19       --default-authent...  15
seconds ago  Up 15 seconds ago                       kind_hermann
510c028c273f  docker.io/library/wordpress:latest  apache2-foregroun...  15
seconds ago  Up 15 seconds ago  0.0.0.0:80->80/tcp  competent_kilby
$
```

Verify WordPress is running locally

The instructions for running WordPress indicate that it is working correctly and it can be accessed using the localhost and port 80.

Create the Kubernetes YAML

With a working instance of WordPress on the local machine, begin the process of replicating these containers on a Kubernetes platform. Podman can generate Kubernetes-based YAML from running containers.

[You might also like to read: Start learning Kubernetes from your local machine]

One pod or multiple pods?

There are two approaches for creating the YAML you will use in the Kubernetes environment: Either put two containers in a single pod with a service, or create two pods, with one container in each, and a service to expose the Apache front end. Determining which approach is best may require some trial and error.

One consideration that may dictate which approach to use is how the containers or pods will communicate. When Compose created these containers, it went through a series of steps to ensure that the two containers could communicate with each other using DNS names. In fact,

Compose set up aliases on the containers that are recognized as DNS names when resolving

containers by name. By putting the containers inside the same pod, there is no need for name resolution between them because they share a network namespace. Therefore, they can simply use **localhost** to communicate with each other.

Placing the containers in different Kubernetes pods gives better flexibility, but the containers will need to communicate with each other using some other mechanism.

Generate the YAML

You must know the container names or IDs to begin creating the Kubernetes YAML. Decide whether Podman should generate a service description for Kubernetes. In this case, expose the Apache front end so that it can interact with WordPress using the browser. Use the `podman generate kube` command to create YAML files.

```
$ sudo podman generate kube -s -f wordpress.yaml a089a40bb9ae 510c028c273f
```

The `-s` in the previous command signifies that Podman will generate service for this pod. The `-f` option allows us to save the generated YAML into a file. Otherwise, the output is sent to **stdout**, where it can be redirected to a file.

```
$ cat wordpress.yaml
# Save the output of this file and use kubectl create -f to import
# it into Kubernetes.
#
# Created with podman-3.0.0-dev
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2020-12-03T22:30:07Z"
  labels:
    app: kindhermann
  name: kindhermann
spec:
  containers:
  - command:
    - docker-entrypoint.sh
    - --default-authentication-plugin=mysql_native_password
    env:
    - name: PATH
      value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
 ...
    workingDir: /
  - command:
    - docker-entrypoint.sh
    - apache2-foreground
    env:
    - name: PATH
      value: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
...
    - name: WORDPRESS_DB_HOST
      value: kindhermann
    - name: WORDPRESS_DB_PASSWORD
      value: db
    - name: APACHE_ENVVARS
      value: /etc/apache2/envvars
 ...
    image: docker.io/library/wordpress:latest
    name: competentkilby
    ports:
    - containerPort: 80
      hostPort: 80
      protocol: TCP
    resources: {}
    securityContext:
      allowPrivilegeEscalation: true
      capabilities:
        drop:
        - CAP_MKNOD
        - CAP_NET_RAW
      privileged: false
      readOnlyRootFilesystem: false
      seLinuxOptions: {}
    workingDir: /var/www/html
status: {}
```

```
---
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2020-12-03T22:30:07Z"
  labels:
    app: kindhermann
  name: kindhermann
spec:
  ports:
  - name: "80"
    nodePort: 30579
    port: 80
    protocol: TCP
    targetPort: 0
  selector:
    app: kindhermann
  type: NodePort
status:
  loadBalancer: {}
```

In order for the Apache container to communicate with the MySQL container, the author of the Compose file has opted to use an environment variable named **WORDPRESS_DB_HOST** to signify the hostname of the MySQL container. Before running this in a Kubernetes environment, change the value of **WORDPRESS_DB_HOST** to the MySQL container name (**kindhermann** in this example) or 127.0.0.1 (containers within the same pod can communicate with each other over localhost).

```
...
    - name: WORDPRESS_DB_HOST
      value: kindhermann OR 127.0.0.1
---
```

SideBar:

When Compose performs a build

In many Compose examples, the author chooses to build their container image. This is usually because they require additional packages or want to perform some level of customization in the image. When this occurs, there will be an additional new image in Podman's image store. Choosing to run the outputted Kubernetes YAML might fail because it references a container image that is only present in the local store.

To remedy this, use `podman push` to move these new images to either a global registry like quay.io or a Kubernetes-specific registry so that Kubernetes can pull these images. Ensure the image name in the resulting YAML file is the same as the image that was pushed.

Kubernetes

The next step in carrying this example forward and applying it to a Kubernetes environment will show how to run this example on both minikube and OpenShift. There is nothing specific in the YAML that prevents the pods from running in another Kubernetes environment, so it should theoretically work with other Kubernetes flavors.

This article assumes the existence of a minikube and/or OpenShift environment. It is out of scope for this article to document the setup of a minikube or OpenShift Kubernetes environment.

minikube

The first step to deploying on minikube is simply to create the pod.

```
$ minikube kubectl -- create -f wordpress.yaml
pod/kindhermann created
service/kindhermann created
```

After waiting a few seconds, check the status of the pod and containers. Depending on the speed and network bandwidth, the pod may already be available. Check the status of the pod using `kubectl get pods`.

```
$ minikube kubectl -- get pods
NAME          READY   STATUS    RESTARTS   AGE
kindhermann   2/2     Running   0          28
```

Now that both containers are ready, test the availability of the WordPress session. First, get the IP address of the pod in Kubernetes using `kubectl`.

```
$ minikube kubectl -- describe  pods | grep Node:
Node:         minikube/192.168.39.7
```

Point your browser of choice to the pod's IP address and see the WordPress setup screen.

OpenShift

For this article, an OpenShift cluster is running on GCP.

Use the generated `wordpress.yaml` to create the pod and service. If using a vanilla Kubernetes environment, replace `oc` with `kubectl` in the following commands.

```
$ oc create -f wordpress.yaml
pod/kindhermann created
service/kindhermann created
```

Wait a few seconds for the pod and service to come up. The **kindhermann** pod is in *Running* status with both containers up and running. The **kindhermann** service is also available with a cluster IP assigned to it.

```
$ oc get pods
NAME          READY   STATUS    RESTARTS   AGE
kindhermann   2/2     Running   0          39s
$ oc get services
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP
     PORT(S)        AGE
kindhermann   NodePort       172.30.103.100   <none>
      80:30579/TCP   45s
Kubernetes    ClusterIP      172.30.0.1       <none>
      443/TCP        44m
openshift     ExternalName   <none>
 Kubernetes.default.svc.cluster.local   <none>         36m
```

View the pod and service in the console.

To access the service from outside of the cluster, expose it, which will create a route.

```
$ oc expose svc/kindhermann
route.route.openshift.io/kindhermann exposed
$ oc/kubectl get routes
NAME          HOST/PORT
                 PATH   SERVICES      PORT   TERMINATION   WILDCARD
kindhermann   kindhermann-default.apps.ci-ln-d3gw292-f76d1.origin-ci-int-
gce.dev.openshift.com          kindhermann   80                  None
```

Exposing the service created the host/port route shown above and access that endpoint. View the setup page of the WordPress application running in the OpenShift or Kubernetes cluster.

Examine the route in the console and directly access the endpoint from there as well.

[Get this free ebook: Managing your Kubernetes clusters for dummies.]

Wrap up

As you can see, moving workload configurations from Docker Compose environments to Kubernetes is straightforward with Podman 3.0. Podman not only gives the flexibility of Docker Compose while developing applications, but it also makes the move to Kubernetes easier when applications are ready for the big leagues. It does all this by using the `podman generate kube` command. Try it out yourself in three simple steps.

Check out these related articles on Enable Sysadmin

Using Podman and Docker Compose

Podman 3.0 now supports Docker Compose to orchestrate containers.

Posted: January 7, 2021
Author: Brent Baude (Red Hat)

Leasing routable IP addresses with Podman containers

Container networking doesn't have to be overly complicated. Learn how to let your container lease an IP from DHCP here.

Posted: November 12, 2019
Author: Brent Baude (Red Hat)

A sysadmin's guide to basic Kubernetes components

Kubernetes control plane nodes and worker nodes, their features, and how they interact.

Posted: December 1, 2020
Author: Shashank Nandishwar Hegde (Red Hat, Sudoer)

Topics: Linux Containers Kubernetes

Brent Baude

Brent is a Principle Software Engineer at Red Hat and leads the Container Runtimes team which includes things like Podman and Buildah. He is a maintainer of Podman upstream and a major contributor as well. More about me

Urvashi Mohnani

Urvashi Mohnani is a senior software engineer at Red Hat on the Container Runtimes team. She has spent the past few years developing emerging open source container technologies such as CRI-O, Buildah, and Podman and presenting on the latest developments in the space. More about me

Related Content

Enable Sysadmin's May 2021 top 10 Linux article round-up

Check out our ten most-read articles from the month of May

Posted: June 2, 2021
Author: Tyler Carrigan (Red Hat)

New container feature: Volatile overlay mounts

With containers, we don't always care about data being retained after a crash. See how volatile overlay mounts can help increase performance in these situations.

Posted: June 3, 2021
Author: Dan Walsh (Red Hat)

A beginner's guide to creating

redirects in an .htaccess file

Use the .htaccess file to manage web sites on shared web hosting platforms.
Posted: June 9, 2021
Author: Abdul Rehman