

Get started with Docker and Kubernetes

Learn how to use Docker containers and Kubernetes clusters, the building blocks of the next generation of DevOps

Doug Tidwell

January 16, 2018

Docker and Kubernetes are the building blocks of the next generation of DevOps. In this tutorial, you'll see how to build Docker images, run them locally, and then push those images to your IBM Cloud account so you can deploy them to a Kubernetes cluster running in the IBM Cloud.

Docker and Kubernetes are two of the hottest technologies in the world of IT. This tutorial will get you up and running with both technologies in the IBM Cloud. Best of all, you can do everything here in your free IBM Cloud Lite account (see sidebar).

Try IBM Cloud for free

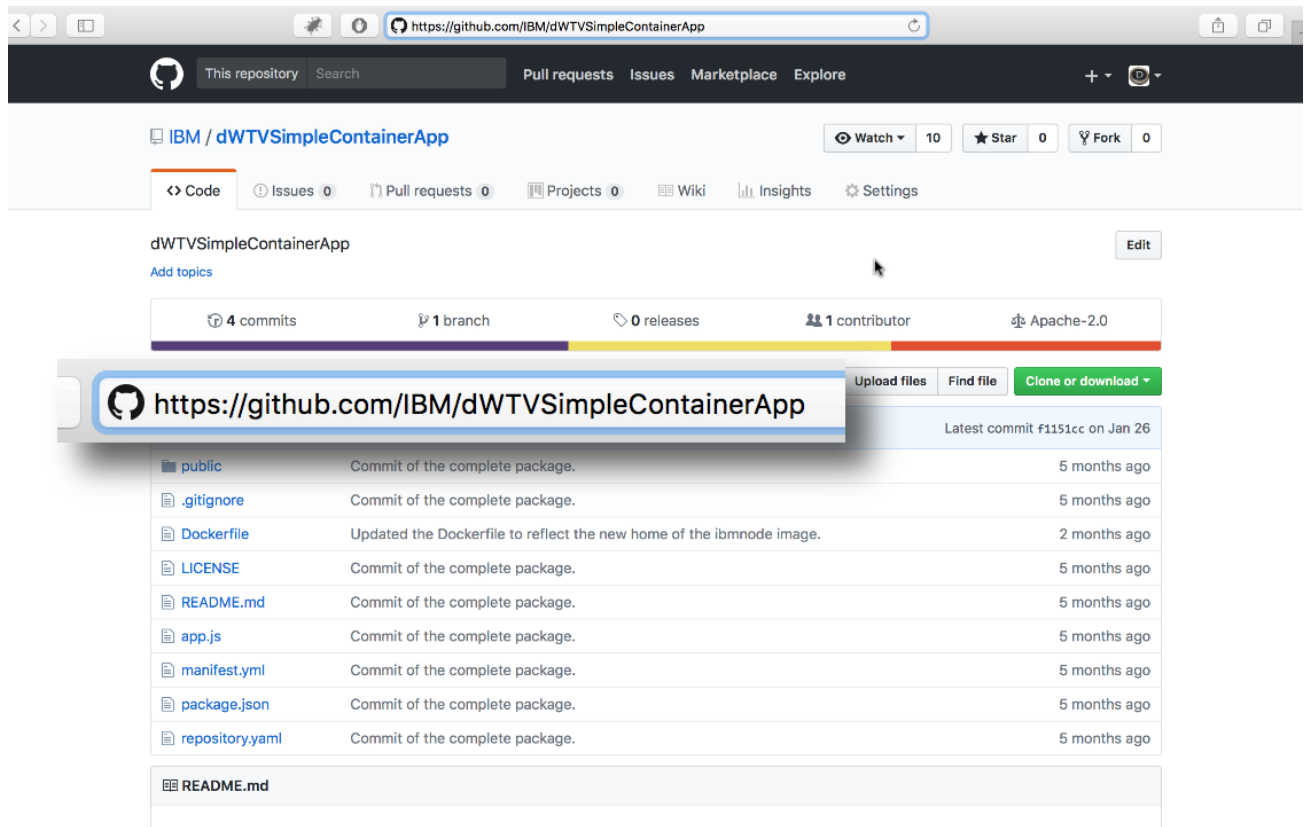
Build your next app quickly and easily with [IBM Cloud Lite](#). Your free account never expires, and you get 256 MB of Cloud Foundry runtime memory, plus 2 GB with Kubernetes Clusters. [Get all the details](#) and find out how to get started. And if you're new to IBM Cloud, check out the [IBM Cloud Essentials course on developerWorks](#).

This tutorial will show you how to:

1. Get the sample code and install the tools you'll need
2. Build a Docker image and run it on your local machine
3. Create a Kubernetes cluster in the IBM Cloud
4. Create a Docker image in the IBM Cloud
5. Deploy the Docker image to your Kubernetes cluster

Getting the sample code

To get started, clone the Github repo that contains the sample code. The app you'll deploy is nothing more than a "Hello World" page combined with a `Dockerfile` that builds the app into a Docker image. The sample code is at github.com/IBM/dWTVSimpleContainerApp.



At the command line, type `git clone https://github.com/IBM/dWTVSimpleContainerApp.git` to clone the repo, then change to the directory of the cloned repo (`cd dWTVSimpleContainerApp`):

```
doug@dougs-mbp-2:~/Documents $ git clone https://github.com/IBM/dWTVSimpleContainerApp.git
Cloning into 'dWTVSimpleContainerApp'...
remote: Counting objects: 25, done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 25 (delta 4), reused 25 (delta 4), pack-reused 0
Unpacking objects: 100% (25/25), done.
doug@dougs-mbp-2:~/Documents $ cd dWTVSimpleContainerApp/
doug@dougs-mbp-2:~/Documents/dWTVSimpleContainerApp $
```

Now that you have the sample code installed on your machine, you're ready to install the tools you'll need as you take over the world of containers and clusters.

Before we continue...

By law, any article that discusses containers must include a picture of a container ship loaded down with ... well, containers. So here you go:



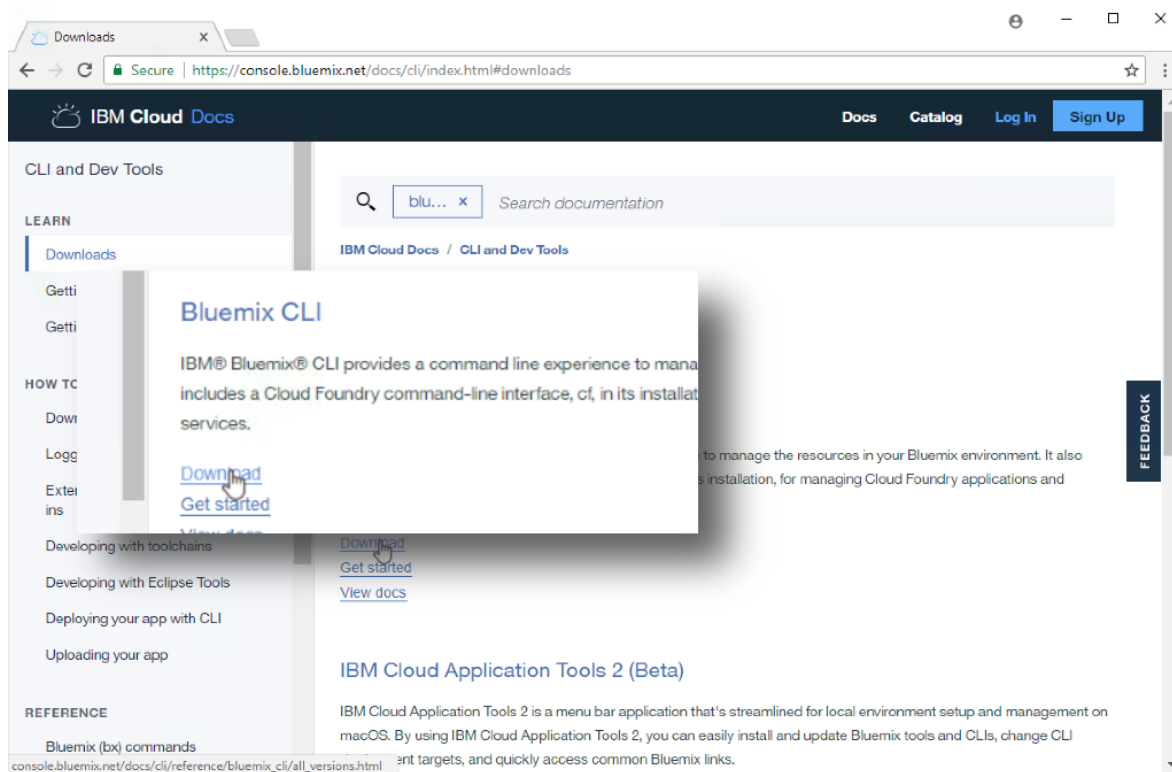
We hope you enjoyed this compulsory diversion.

Installing the required tools

Before you can run `docker build` or do anything else with Docker and Kubernetes, you'll need to install the tools for IBM Cloud, Docker, and Kubernetes.

Installing the IBM Cloud tools

Go to console.bluemix.net/docs/cli/index.html#downloads and click the **Download** link to get the latest version of the IBM Cloud tools. Download the tools for your platform and run the installer.

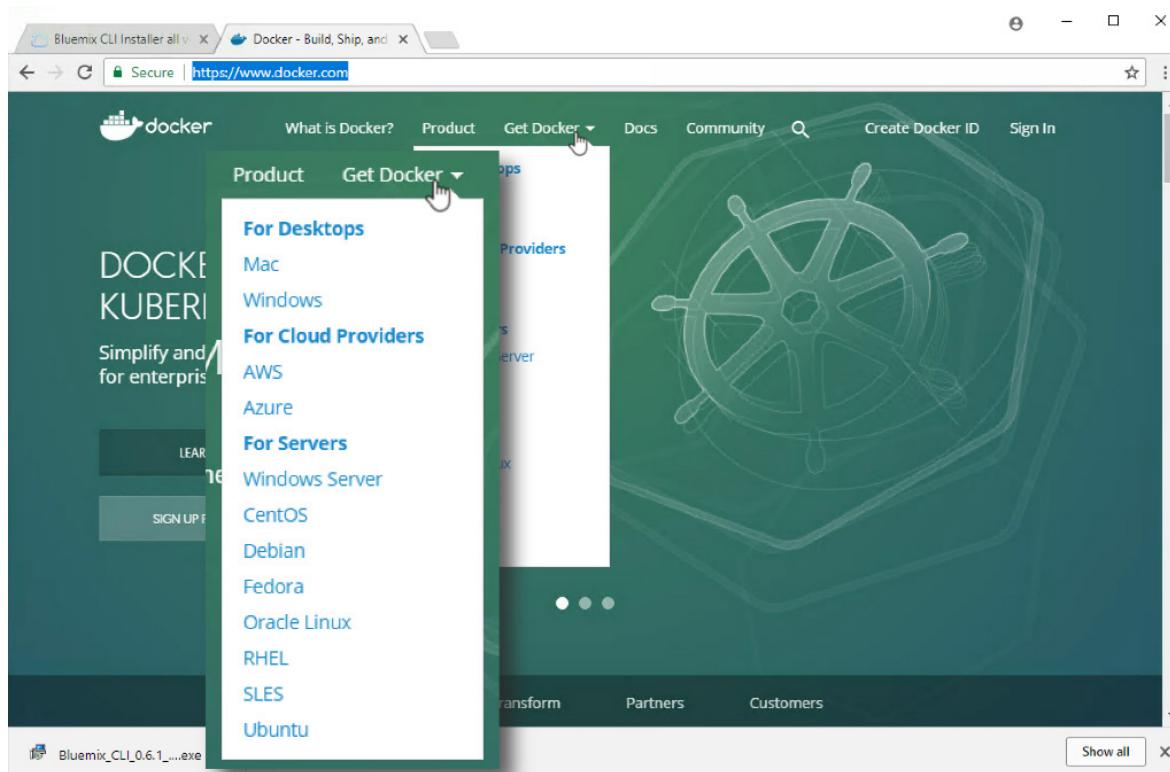


Note: For historical reasons, the command to work with IBM Cloud resources is `bluemix`. You may be concerned that typing *seven whole characters* for every command will be exhausting. Worry not, friends: We've abbreviated the command to `bx`. You'll find this shortcut will save you hours of typing each day, freeing you to spend more time with family and friends, re-devote yourself to long-neglected hobbies, or perhaps take a second job to make ends meet.

You're welcome.

Installing Docker

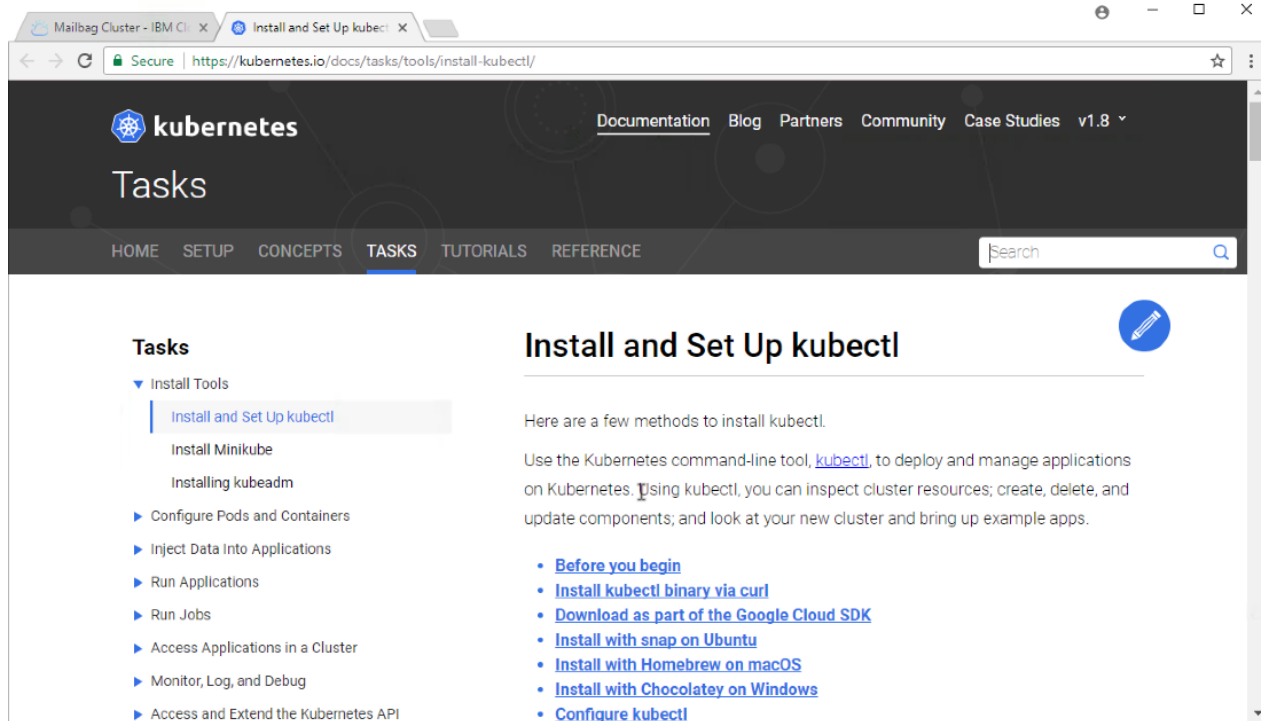
The next step is to install Docker and its associated command-line tools. Go to docker.com and click the **Get Docker** link:



Download and install Docker Community Edition. Depending on your platform, you may get a message that you need to reboot your system before continuing. Wait until you've installed the Kubernetes tools before doing that.

Installing the Kubernetes tools

Although you won't use Kubernetes until later in this article, you might as well install the `kubectl` tool now. Go to kubernetes.io/docs/tasks/tools/install-kubectl/ and install `kubectl` according to the instructions for your platform.



When you create a cluster in the IBM Cloud, you'll configure `kubectl` to work with your cluster ... but there are a few steps you'll need to go through first. For now, just install the command.

Logging in to the IBM Cloud

At this point, you should have all of the IBM Cloud, Docker, and Kubernetes tools installed, and you've rebooted your machine if necessary. The next step is to log in to your IBM Cloud account. At the command line, type `bx login` to log in:

```
Command Prompt - bx login
C:\Users\Doug>bx login
API endpoint: https://api.ng.bluemix.net
Email> [redacted]
Password>
Authenticating...
OK
Select an account (or press enter to skip):
1. Doug Tidwell's Account ([redacted]) <-->
Enter a number> 1
```

Once you've logged in to the IBM Cloud, you need to install the plugins for the IBM Container Service and the IBM Container Registry. Type the following commands:

- `bx plugin install container-service -r Bluemix`
- `bx plugin install container-registry -r Bluemix`
- `bx cr login`

The first two commands install the plugins for the container service and the container registry from the [BlueMix](#) repository. The last command logs you in to the container registry service. This lets you access Docker images in the public registry of the IBM Cloud.

Using Docker images and containers on your machine

To view this video, **Video: Using Docker images and containers on your machine**, please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

Starting Docker

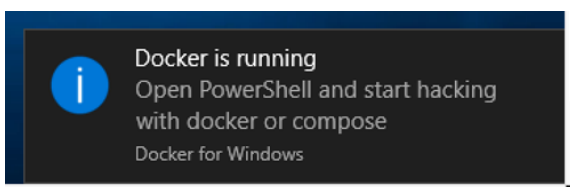
Now it's time to start building things. First, you need to start Docker on your machine. How you do that, of course, depends on your platform.

Windows

Double-click the desktop icon:



When Docker is up and running, you'll see a message in the lower right-hand corner of your desktop:

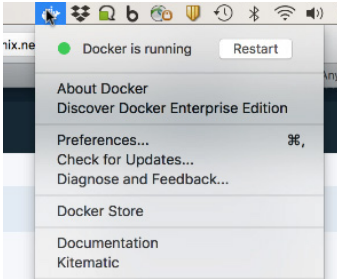


Mac

Click the desktop icon:



When Docker is up and running, clicking the Docker icon in the menu bar shows this comforting status message:



Linux

Starting Docker on Linux can vary from one distribution to another, but you'll typically run one of these two commands:

- `sudo systemctl start docker`
- `sudo service docker start`

If you're having trouble starting Docker on Linux, see the [Docker documentation](#) for more information.

Building a Docker image

With Docker up and running, it's time to build the image that contains your app. The repo you cloned earlier contains a `Dockerfile` that tells Docker how to build the image. The file looks like this:

```
FROM ibmcom/ibmnode:latest

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app

COPY package.json /usr/src/app
RUN npm install

EXPOSE 6006

COPY . /usr/src/app

CMD ["node", "app.js"]
```

Here is a brief explanation of the commands in the file:

- **Line 1:** This tells Docker that the image will be based on the latest IBM node.js image. (You can access this image because you ran the `bx cr login` command earlier.)
- **Lines 3 and 4:** This creates the directory `/usr/src/app` and tells Docker to use that as the working directory.
- **Lines 6 and 7:** Copies the `package.json` file to the working directory and runs the `npm install` command to install all of your app's dependencies.
- **Line 9:** Tells Docker to open port 6006 to incoming traffic. The app listens for connections on this port.
- **Line 11:** Copies all of the files in the current directory to the working directory.
- **Line 13:** Tells Docker what command to run when the image starts. The first parameter is the command, the second is the list of arguments for that command. When this image starts, Docker runs `node app.js`.

Go to the command line and run the command `docker build -t basicapp:v1 .` (The period there is part of the command; it tells Docker to do its work in the current directory.) You'll see something like this:

```
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ docker build -t basicapp:v1 .
Sending build context to Docker daemon  4.128MB
Step 1/8 : FROM registry.ng.bluemix.net/ibmnode:latest
latest: Pulling from ibmnode
b2375db445eb: Already exists
44591641ad34: Already exists
487d40e8a63d: Already exists
a32789350737: Already exists
cce9c0bf740d: Already exists
57b9ff1a569e: Already exists
4a5433e81f22: Already exists
8e92961bc620: Already exists
a14d0746261c: Already exists
f808329302c0: Already exists
Digest: sha256:0ffd58f899b5fab04fa1d39bfd6aa753dea2148af75f30ebee285975d8a078ef
Status: Downloaded newer image for registry.ng.bluemix.net/ibmnode:latest
--> a2805d9e7ffc
Step 2/8 : RUN mkdir -p /usr/src/app
--> Running in 12185315c607
--> 6c27c23c3c63
Removing intermediate container 12185315c607
Step 3/8 : WORKDIR /usr/src/app
--> 12bba3b06595
Removing intermediate container ba9b6d01e6dc
Step 4/8 : COPY package.json /usr/src/app
--> 73c91eec30f2
Step 5/8 : RUN npm install
--> Running in 1ff8e8a0180b
```

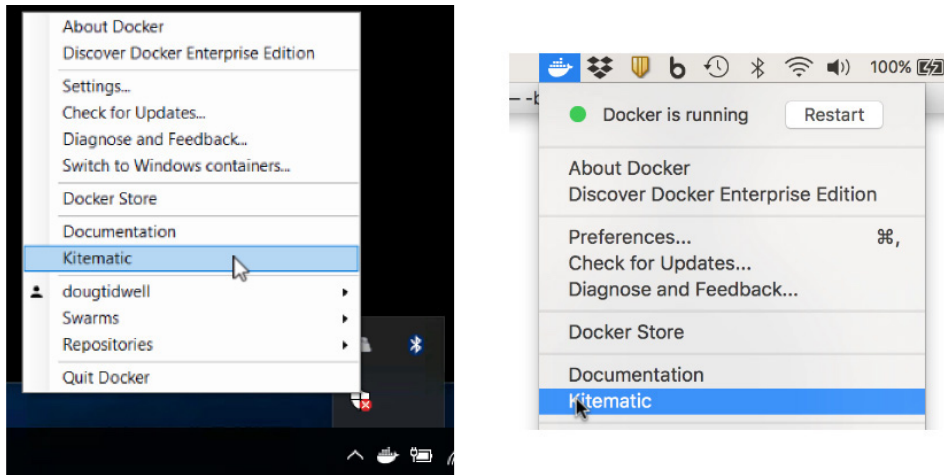
Your results will vary. In this screen capture, the node.js image already exists on the system, so there are lots of things that Docker doesn't have to download. The first time you run `docker build` on your machine, none of the dependencies your image needs will be on your machine, so Docker will have to download them. Subsequent calls to `docker build` will probably work much faster. Also notice that the `npm install` command hasn't finished running in this example.

Running a Docker image on your machine

You're no doubt tingling with excitement already, what with having built your first Docker image and all that, but let's move on and actually *run* that image. At the command line, type `docker create basicapp:v1` to create a container from your image:

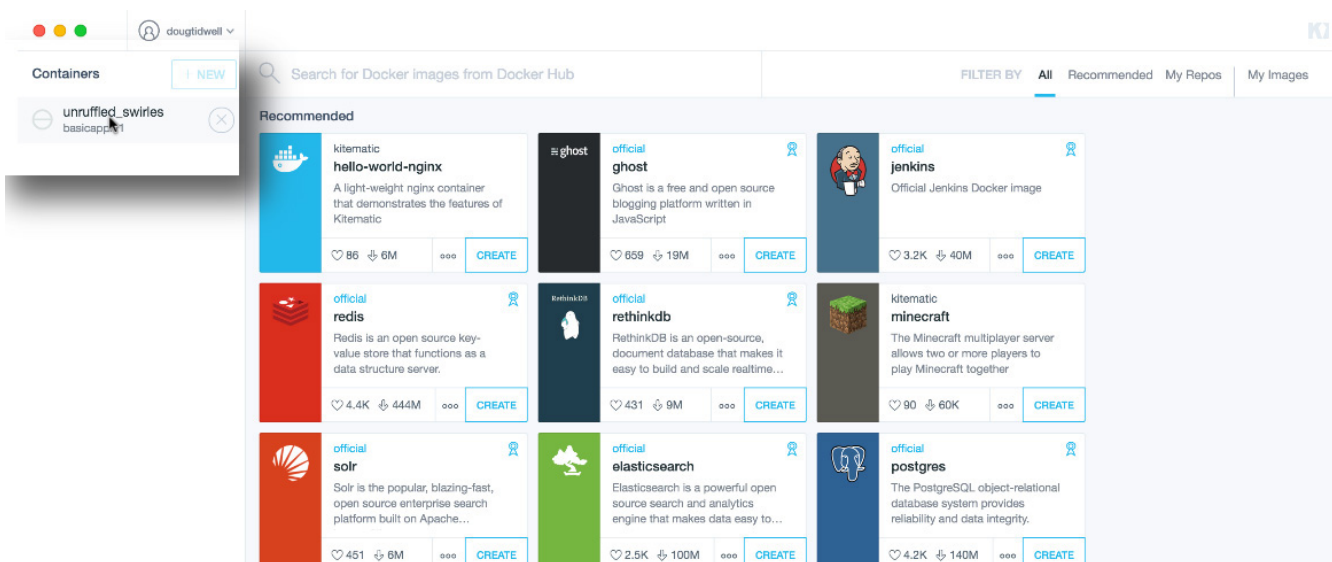
```
doug@dougs-MBP-2:~/Developer/dWTVSimpleContainerApp $ docker create basicapp:v1
fb252d8841f3581606bfe4880eb4a2f9ce5fb3d7775010747d4a81b0dc9763b8
doug@dougs-MBP-2:~/Developer/dWTVSimpleContainerApp $
```

Um, not terribly exciting, is it? You ran a command and got a long hex string that probably means something, although it's not clear what that might be. Fortunately, the Docker community has some great tools that make it easy to work with Docker containers. If you're running on Windows or the Mac, use the awesome Kitematic tool. Start it from the system tray on Windows or the menu bar on the Mac:



Note: As of this writing, Kitematic requires a separate install on Windows. Check the Docker documentation for instructions.

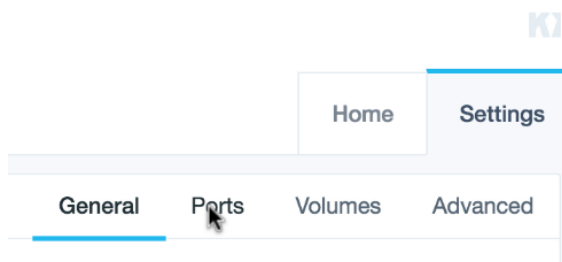
When Kitematic starts, you'll see a display similar to this one:



There are tiles for various Docker images in the dockerhub catalog. If you'd like to set up Jenkins or PostgreSQL or Minecraft or other software, you can easily do that with these images. The

interesting thing here is in the upper left-hand corner: It's a list of all the containers running on your machine. Each one has a randomly assigned name. In this screen capture, the container is named `unruffled_swirles`—much more memorable than the long string of hex characters you saw at the command line a minute ago.

At this point, you have a Docker container running on your machine. It has the basic Hello World app running inside it, but how do you access that app? To do that, you'll have to define a port for the app. Docker will map incoming requests for the port number you define to port 6006 in your container. (Recall the `EXPOSE 6006` command in the `Dockerfile`.) Click the **Settings** menu in the upper right, then click **Ports**:



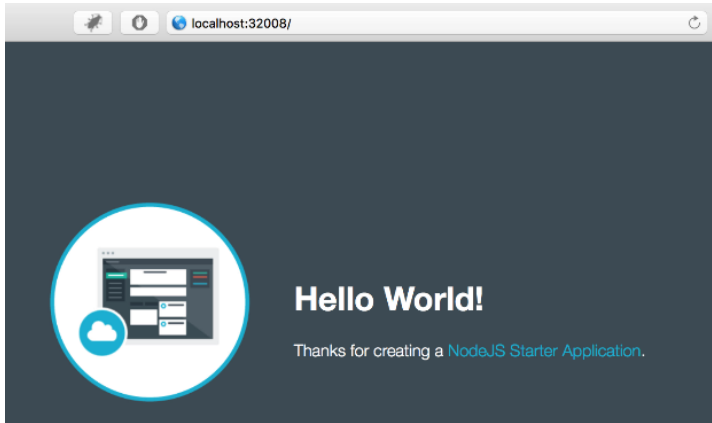
You'll see a display that lets you map the port 6006 in your Docker container to a port on your machine. Here's how to use port 32008:

Configure Ports

DOCKER PORT	MAC IP:PORT	
6006	localhost:32008	TCP
	localhost:	TCP

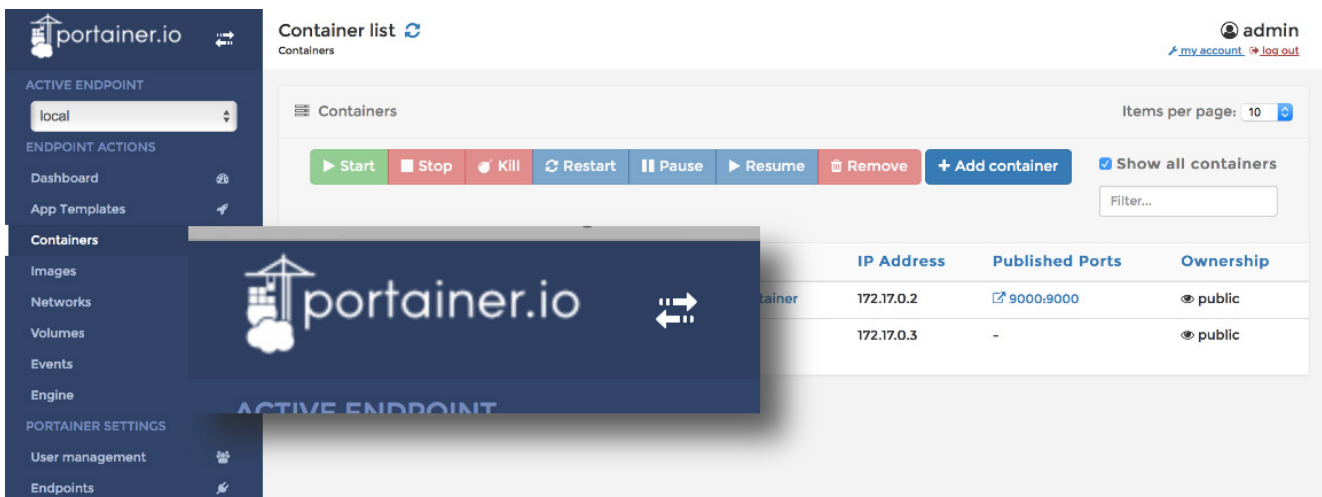
SAVE

Click **Save** to save the mapping, then go to `http://localhost:32008` in your browser. You'll see the Hello World app up and running:



Congratulations! You've successfully built a Docker image, deployed it to your machine, and configured the Docker runtime so that you can access the app running in the Docker container from a browser.

For Linux, as of this writing Kitematic for Linux does not exist. For similar functionality, try Portainer:

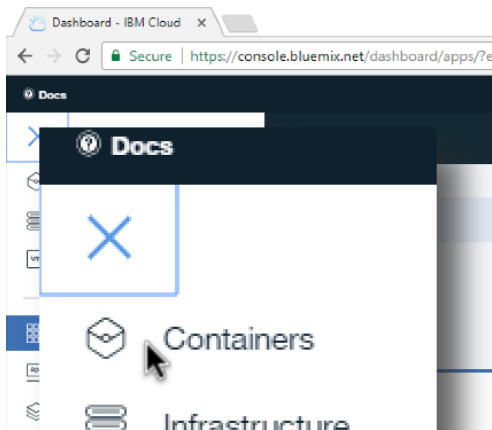


Portainer is packaged as a Docker image, so the installation process is nothing more than asking Docker to download and run that image. See portainer.io for all the details.

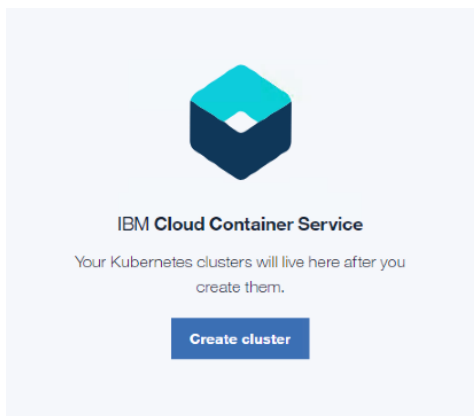
Creating a Kubernetes cluster in the IBM Cloud

To view this video, **Video: Building a Kubernetes cluster in the IBM Cloud**, please access the online version of the article. If this article is in the developerWorks archives, the video is no longer accessible.

The ultimate goal here is to deploy an app running in a Docker container to a Kubernetes cluster. It takes a few minutes for a cluster to get up and running, so go ahead and create the cluster now. You'll do the Docker deployment later. From the IBM Cloud console, go to the menu in the upper left-hand corner and click **Containers**:



Next, click the **Create cluster** button in the middle of the page:



Finally, give your cluster a name (in the example below, the cluster is named "Mailbag") and make sure you've checked the **Lite plan** (aka the free plan), then click the **Create Cluster** button on the right-hand side of the panel:

Dashboard / Clusters /

Create Cluster

Provision a cluster of hosts, called worker nodes, to deploy and manage highly-available apps.

[Terms](#) [Docs](#)

Cluster type

Lite plan ✓

New to Kubernetes? Create a cluster with 1 worker node to explore the capabilities.

Free

Pay-As-You-Go plan ✓

Create a fully-customizable, production-ready cluster with your choice of hardware

Starting from \$0.19 hourly

Cluster details

Cluster Name

Location ⓘ Kubernetes version ⓘ

Order Summary

Lite - 2 CPUs, 4 GB RAM

1 worker node **Free**

Total due now: Free
estimated

Create Cluster

Cancel

It may take several minutes for the cluster to be provisioned and started. While that's happening, go ahead and build the Docker image that you'll deploy to the cluster.

Creating a Docker image in the IBM Cloud

You've created a Docker image on your local machine, which is great, but you need to get that image into the IBM Cloud before you can deploy it to a Kubernetes cluster. (Make sure you've logged in to the IBM Cloud image repository with the `bx cr login` command before you go forward.) First, define a namespace for your images. As an example, to create the `mailbag` namespace, enter this command:

```
bx cr namespace-add mailbag
```

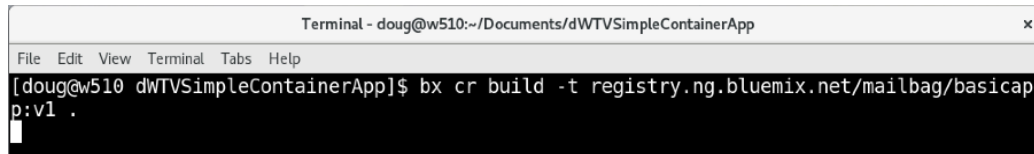
Namespaces allow you to create groups of Docker images in your IBM Cloud account. You can't create or upload a Docker image until you have a namespace. Your results should look like this:

```
Command Prompt
C:\Users\Doug\Documents\dwTVSimpleContainerApp>bx cr namespace-add mailbag
Adding namespace 'mailbag'...
Successfully added namespace 'mailbag'
OK
C:\Users\Doug\Documents\dwTVSimpleContainerApp>
```

Now you need to get your Docker images into your container repository in the IBM Cloud. You can do that in one of two ways: build the image directly in the cloud or push an existing image into the cloud. Here are the instructions for each approach:

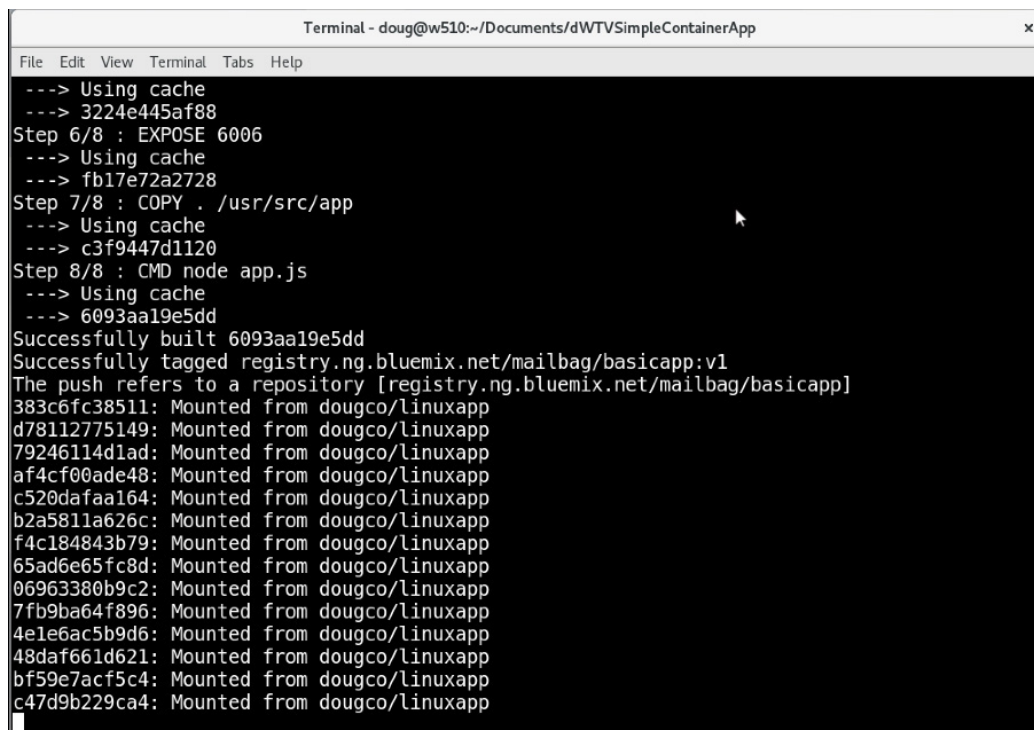
Building a Docker image in the IBM Cloud

You can use the `bluemix` command to create an image in the cloud. Type `bx cr build -t registry.ng.bluemix.net/mailbag/basicapp:v1 .` (Be sure to include the period at the end of the command.)



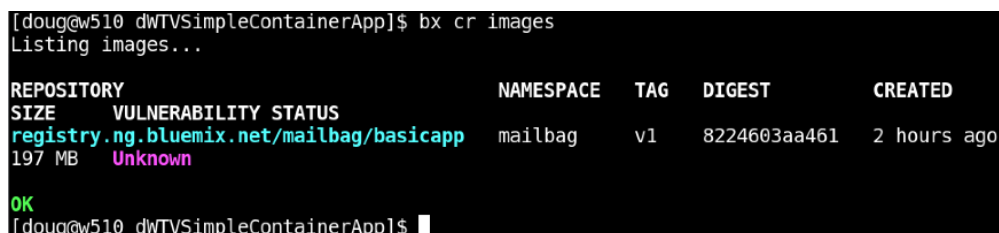
```
Terminal - doug@w510:~/Documents/dWTVSimpleContainerApp
File Edit View Terminal Tabs Help
[doug@w510 dWTVSimpleContainerApp]$ bx cr build -t registry.ng.bluemix.net/mailbag/basicapp:v1 .
```

The output of this command is similar to the `docker build` command:



```
Terminal - doug@w510:~/Documents/dWTVSimpleContainerApp
File Edit View Terminal Tabs Help
---> Using cache
---> 3224e445af88
Step 6/8 : EXPOSE 6006
---> Using cache
---> fb17e72a2728
Step 7/8 : COPY . /usr/src/app
---> Using cache
---> c3f9447d1120
Step 8/8 : CMD node app.js
---> Using cache
---> 6093aa19e5dd
Successfully built 6093aa19e5dd
Successfully tagged registry.ng.bluemix.net/mailbag/basicapp:v1
The push refers to a repository [registry.ng.bluemix.net/mailbag/basicapp]
383c6fc38511: Mounted from dougco/linuxapp
d78112775149: Mounted from dougco/linuxapp
79246114d1ad: Mounted from dougco/linuxapp
af4cf00ade48: Mounted from dougco/linuxapp
c520dafaal64: Mounted from dougco/linuxapp
b2a5811a626c: Mounted from dougco/linuxapp
f4c184843b79: Mounted from dougco/linuxapp
65ad6e65fc8d: Mounted from dougco/linuxapp
06963380b9c2: Mounted from dougco/linuxapp
7fb9ba64f896: Mounted from dougco/linuxapp
4e1e6ac5b9d6: Mounted from dougco/linuxapp
48daf661d621: Mounted from dougco/linuxapp
bf59e7acf5c4: Mounted from dougco/linuxapp
c47d9b229ca4: Mounted from dougco/linuxapp
```

The `bx cr build` command automatically pushes the new image into your image repository. You can type `bx cr images` to ensure that your image was created and is in the IBM Cloud:



```
[doug@w510 dWTVSimpleContainerApp]$ bx cr images
Listing images...

REPOSITORY                                NAMESPACE  TAG  DIGEST              CREATED
SIZE  VULNERABILITY STATUS
registry.ng.bluemix.net/mailbag/basicapp  mailbag     v1   8224603aa461        2 hours ago
197 MB  Unknown

OK
[doug@w510 dWTVSimpleContainerApp]$
```

Pushing an existing Docker image into the IBM Cloud

If you want to take a Docker image that you've already built and push it into the cloud, use these two Docker commands:

1. `docker tag basicapp:v1 registry.ng.bluemix.net/mailbag/basicapp:v2`
2. `docker push registry.ng.bluemix.net/mailbag/basicapp:v2`

Your results should look like this:

```
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ docker tag basicapp:v1 registry.ng.bluemix.net/mailbag/basicapp:v2
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ docker push registry.ng.bluemix.net/mailbag/basicapp:v2
The push refers to a repository [registry.ng.bluemix.net/mailbag/basicapp]
4e024295eab8: Layer already exists
4f1aa0faaeae: Layer already exists
e0812c098653: Layer already exists
955aa9a91d83: Layer already exists
c520dafaa164: Layer already exists
b2a5811a626c: Layer already exists
f4c184843b79: Layer already exists
65ad6e65fc8d: Layer already exists
06963380b9c2: Layer already exists
7fb9ba64f896: Layer already exists
4e1e6ac5b9d6: Layer already exists
48daf661d621: Layer already exists
bf59e7acf5c4: Layer already exists
c47d9b229ca4: Layer already exists
v2: digest: sha256:8e20d36d5ebacfea3812566dbfedf479165bf8f907ff100b42ce95a25849044a size: 3245
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $
```

Note: This example uses the tag `v2` on the assumption that you already have a `v1` image in your repository, created by the command in the previous section. Creating two tags (`v1` and `v2`) gives you two versions of the same image. As before, typing `bx cr images` lists the Docker images in your repository.

Configuring the `kubect1` command

You'll use a combination of `bx` and `kubect1` commands to deploy the image as a container running in the cluster and to expose the container to the world. Before you can run these commands, you need to configure the `kubect1` command to work with your cluster running in the IBM Cloud. Here are the commands you'll run, along with an explanation of each:

1. `bx cs cluster-config Mailbag`
This uses the Bluemix container service plugin to get the configuration details of the Mailbag cluster you created earlier. It returns a command that you can copy and paste to set the `KUBECONFIG` environment variable.
2. [Set the `KUBECONFIG` variable]
This command is in the output from the previous command. It is an `export` statement on Linux and the Mac, and a `set` command in the Windows command line; it returns the information you need for an `$env:KUBECONFIG` definition in Windows PowerShell.

These two commands look like this on Linux or the Mac:

```
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ bx cs cluster-config Mailbag
OK
The configuration for Mailbag was downloaded successfully. Export environment variables to start using Kubernetes.

export KUBECONFIG=/Users/doug/.bluemix/plugins/container-service/clusters/Mailbag/kube-config-hou02-Mailbag.yml
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ export KUBECONFIG=/Users/doug/.bluemix/plugins/container-service/clusters/Mailbag/kube-config-hou02-Mailbag.yml
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $
```


(Notice that the `export` command is merely a cut and paste of the output of the `bx cs cluster-config` command.)

Once `KUBECONFIG` is set, any `kubectl` commands you run automatically affect your cluster running in the IBM Cloud.

Deploying a Docker image to your Kubernetes cluster

At this point, the Docker image you want to deploy is in the IBM Cloud, your Kubernetes cluster is up and running, and the `kubectl` command is configured to work with your cluster.

It's showtime!

You're only four commands away from victory. Here they are:

1. `kubectl run apptest --image=registry.ng.bluemix.net/mailbag/basicapp:v1`
This starts the Docker image `basicapp:v1` from your IBM Cloud container repository in the cluster. It creates a Kubernetes deployment named `apptest`.

2. `kubectl expose deployment/apptest --type=NodePort --name=apptest-service --port=6006`

This longwinded command exposes the `apptest` deployment as a `NodePort` named `apptest-service` running on port 6006. (We'll talk about `NodePorts` some other time; suffice to say it's your only option with a free cluster.)

At this point, the deployment is created and the service is exposed:

```
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ kubectl run apptest --image=registry.ng.bluemix.net/mailbag/basicapp:v1
deployment "apptest" created
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ kubectl expose deployment/apptest --type=NodePort --name=apptest-service --port=6006
service "apptest-service" exposed
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $
```

3. `kubectl describe service apptest-service`

This displays a variety of useful information about the service you just exposed. Amongst other things, it displays the randomly assigned port number that Kubernetes gave to this

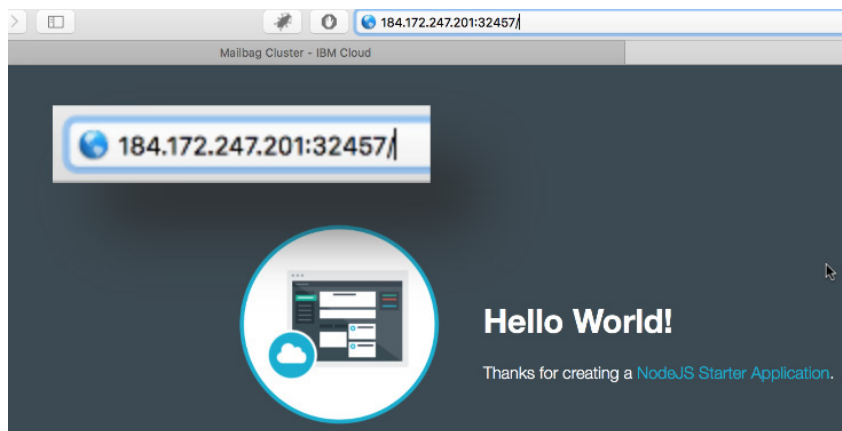
```
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $ kubectl describe service apptest-service
Name:                apptest-service
Namespace:           default
Labels:              run=apptest
Annotations:         <none>
Selector:            run=apptest
Type:               NodePort
IP:                172.21.113.16
Port:              <unset> 6006/TCP
TargetPort:        6006/TCP
NodePort:          <unset> 32457/TCP
Endpoints:         172.30.15.202:6006
Session Affinity:   None
External Traffic Policy: Cluster
Events:            <none>
doug@Doug-MBP-2:~/Developer/dWTVSimpleContainerApp $
```

4. `bx cs workers Mailbag`

Finally, you need to get the IP address of the Kubernetes cluster itself. This command returns that information:

```
doug@dougs-MBP-2:~/Developer/dWTVSimpleContainerApp $ bx cs workers Mailbag
OK
ID                               Public IP      Private IP     Machine Type  State  Status
Version
kube-hou02-paf5a7c83c5fb64c39a239f95d8ccc5cd2-w1 184.172.247.201 10.77.143.40  free         normal Ready
1.7.4_1503
doug@dougs-MBP-2:~/Developer/dWTVSimpleContainerApp $
```

Drum roll please: Now that you know the IP address of the cluster and the port number of the service, you can combine the two in your browser and see Hello World in all its glory:



You've deployed your Docker image to a Kubernetes cluster running on the web. The Hello World application is worldwide; anyone anywhere can access it. Try to remain humble as you bask in the glory of your accomplishments here.

Summary

At this point you're completely ready to take over the world of containers and clusters. You know how to build a Docker image, how to push it to the IBM Cloud, and how to deploy it in a Kubernetes cluster. The app running in the Docker container in your cluster is live on the web, so anyone in the world can behold your handiwork. There are plenty more things to master (handling credentials inside a cluster, for example), but you're off to a solid start.

Related topics

- [Learn more about Docker](#)
- [Learn more about Kubernetes](#)
- [Doug Tidwell's developerWorks Mailbag videos](#)
- [IBM Cloud Developer Center](#)
- [Try IBM Cloud for free](#)

© Copyright IBM Corporation 2018

(www.ibm.com/legal/copytrade.shtml)

[Trademarks](#)

(www.ibm.com/developerworks/ibm/trademarks/)