Table of Contents

Containerize an Application & run in a Kubernetes Cluster

Introduction

This document provides the process to create or use an existing application, containerize the application using Docker and then move the application to run in an IBM Kubernetes Cluster.

The assumption is the user has an IBM Cloud Account and have downloaded the following Command Line Interfaces:

- IBM Cloud CLI
- IBM Cloud Container Registry CLI
- IBM Cloud Kubernetes Service CLI

The following URL will provide the steps to install the IBM Cloud commands and the plug-ins for IBM Cloud Kubernetes & IBM Cloud Container Registory

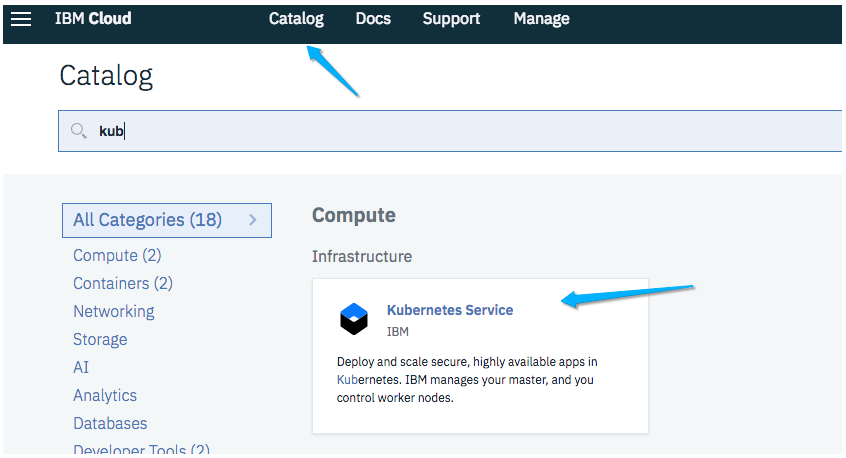
https://cloud.ibm.com/docs/cli/index.html#overview

Kubernetes Cluster

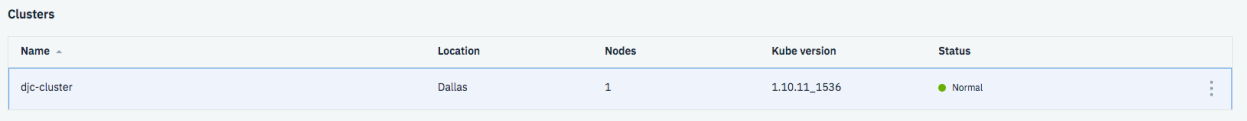
If you don't already have a Kubernetes Cluster, you'll need to create one either using the IBM Cloud Dashboard or from the Command Line Interface.

Dashboard

Using the Catalog, search on kubernetes and follow the steps to create a cluster. This takes approximately 15-20 minutes to deploy a cluster with a single worker node if using the "Standard" plan.



Once the cluster is creates, it will appear in the Dashboard under "Clusters"



Command Line Interface

Set or confirm the region you want to create the cluster in
ic cs region
ic cs region-set us-south

create cluster
ic cs cluster-create <cluster-name>

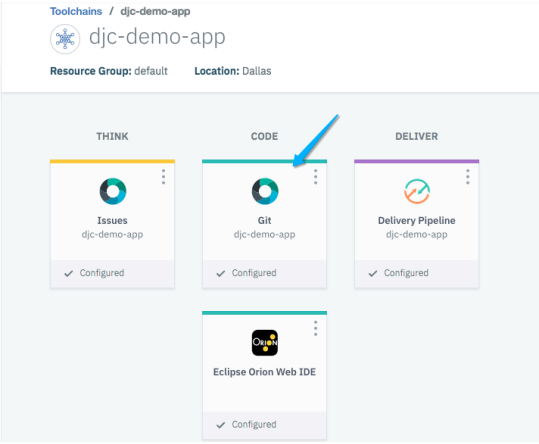
Create Application

If you do not have an existing application to containerize, you can create one for demonstration purposes. The following examples include the steps for containerizing node.js and python.

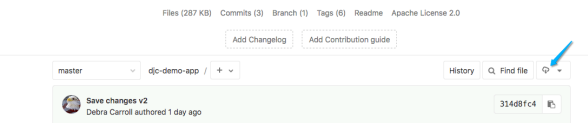
Node.js

If you need to create an application for testing, go to the IBM Catalog and select SDK for Node.js. Follow steps to setup application and complete the steps to set-up a toolchain.

Download Source Code:
From the toolchain, select “Git”



Download code by selecting the download symbol



Create a directory locally & unzip the downloaded source code. Once you’ve done that cd into the main directory where the source code resides. Example:

```
Debras-MacBook-Pro:djc-demo-app-master dcinmca$ pwd
/Users/dcinmca/djcdemo/djc-demo-app-master
Debras-MacBook-Pro:djc-demo-app-master dcinmca$ ls
Dockerfile              README.md               deployment.yaml         node_modules            package.json
LICENSE                 app.js                  manifest.yml            package-lock.json       public
Debras-MacBook-Pro:djc-demo-app-master dcinmca$
```

Run the following command to download dependencies
npm install

You can also test that the source code runs locally using "npm start" which will provide you with a URL you can use to test the application

```
Debras-MacBook-Pro:djc-demo-app-master dcinmca$ npm start

> NodejsStarterApp@0.0.1 start /Users/dcinmca/djcdemo/djc-demo-app-master
> node app.js

server starting on http://localhost:6002
```

Create a Docker Container

To create a docker container you will first need to have downloaded DOCKER and run that locally. You can get the binaries from https://www.docker.com

Next you need to create a "Dockerfile" in the main directory where your source code resides. Example code for the Dockerfile is below. Note: You will need edit the following
WORKDIR: This will need to be a directory you've created locally
EXPOSE 6001 ← this port is the default set with SDK you can change this

```
FROM node:carbon
WORKDIR /users/dcinmca/app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 6001
CMD ["npm", "start" ]
```

Login to IBM Cloud

In IBM Cloud using the ibmcloud CLI, validate that you have a docker registry namespace, if not you will need to create one. You will first need to login to your IBM Cloud Account from the command line.

Example Login:
bx login –a https://api.ng.bluemix.net –u <username> –o <org name> –s <space name> ––apikey <apikey>

Create Docker Namespace

```
ic cr namespace-list

Listing namespaces...

Namespace
djcarro-ibm

OK
```

If you don't have a namespace created, the following command can be used to create it.

ic cr namespace-add <name>

Create Docker Container Locally

In the directory containing the source code and Dockerfile, run the following command (NOTE: the space & then a "**.**" At the end of the line which tells the command to look for the Dockerfile in the current directory.

docker build -t <container registry API> /<registry name>/<containerName>:<version #> .

example:
docker build -t registry.ng.bluemix.net/djcarrol-ibm/demotest:v1 .

Note: Region you are in IBM Cloud will dictate the Container Registry API Endpoint (i.e. registry.ng.bluemix.net). You determine which region using the following command:

ic cr info

```
Container Registry                registry.ng.bluemix.net
Container Registry API endpoint   https://registry.ng.bluemix.net/api
```

Push Docker Container into IBM Cloud

docker push <container registry API>/<registry name>/<containerName>:<version #>

example:
docker push registry.ng.bluemix.net/djcarrol-ibm/demotest:v1

Move Containerized Application to Kubernetes Cluster

This next steps assumes that you have downloaded the kubernetes cli and a cluster has been created w/ a minimum of one worker node

Validate communication to Cluster and Worker Node(s)

This command will provide the name of the cluster running in IBM Cloud
ic cs clusters

This command will validate the worker node(s) state & IP Addresses

ic cs workers <cluster name>

Example**:** **ic cs workers djc-cluster**

Set Cluster Configuration Variable

ic cs cluster-config <cluster name>

The output of this command will provide the location of the Kubernetes configuration file. Copy & paste the entire line into your terminal window as this is needed to communicate within the cluster & the worker nodes.

Example:

```
dcinmca$ ic cs cluster-config djc-cluster
OK
The configuration for djc-cluster was downloaded successfully.

export KUBECONFIG=/Users/dcinmca/.bluemix/plugins/container-service/clusters/djc-cluster/kube-config-hou02-djc-cluster.yml
```

Create Deployment file for Application

The deployment file should be created in the same directory as the source code. It is used to move the docker container image that you created earlier into the Kubernetes cluster. Kubernetes will assign the application to a worker node and a pod within the worker node.

Example: deployment.yaml

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: <application Name>
  namespace: default
spec:
  replicas: 1    ← you can change the # of replicas
```

```
  template:
    metadata:
      labels: # labels to select/identify the deployment
        app: <application Name>
    spec:     # pod spec
      containers:
      - name: <application Name>
        image: <container registry API>/<registry name>/<containerName>:<version #>
        ports:
        - containerPort: 6001  ← this port should be the same as in
the dockerfile
```

To push this up use the following command:

kubectl create –f deployment.yaml

Validate deployment

kubectl get deployments

Create external port for application

Running the following command will create an external port for the application that users can connect to. You can assign a port number or allow kubernetes to assign a random port within the port range of 30000-32767 (Note if assigning a port number, you must stay within this range and not assign a port already in use)

kubectl expose deployment <application Name> --type NodePort

Once the port is created, you can validate the port number assigned with the following command

kubectl get services

Access the application running in Kubernetes from URL

The URL provided for external access is a combination of the worker node public IP Address and the port number assigned to the application. To obtain the Worker Node's Public IP, enter the following command

ic cs workers <cluster Name>

In URL enter:

<Worker Node Public IP>:<Port Number>