Zero to Cloud-Native with IBM Cloud

Part 6: Setting Up and Configuring a Cloud-Native Development Environment

Kevin Collins
kevincollins@us.ibm.com
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Kunal Malhotra
kunal.malhotra3@ibm.com
Cloud Platform Engineer
IBM Cloud MEA

1 - Development Environment

There are many different tools you will use to develop a cloud-native application from various CLIs, an IDE, and support tools. Setting up a proper environment can make your development experience much more productive. If you are new to cloud-native, it is worth the time to setup a good development environment before starting to code. This section will go through what I have found to be effective but certainly not the only option. I'm always looking for new tools and techniques to make my life easier. The tutorial here will focus on setting up a development environment on a mac, if you are a windows user – switch to a mac ☺.

1 – 1 Command Line Tools

If you don't already use the command line / terminal you should. It is an essential tool and skill that is essential when it comes to cloud-native. For example, with OpenShift the console is really great, you can point and click and get all the information about your cluster and workloads that you need. However, if you know how to use the oc (Openshift) cli or Kubernetes kubectl cli, you will be much more productive. With the exception of GitHub, which I have a personal battle with when it comes to the command line, I use the command line as much as possible and find it to be much more productive.

2 - Terminal

The first thing you will want to setup is a good terminal. While the terminal that comes with the mac is good, there are better alternatives.

2 – 1 iTerm 2

I recommend iTerm2 as a great terminal for the mac.

To install iTerm2, run the command:

```
brew cask install iterm2
```

Note: if you have not installed homebrew yet, run this command in your terminal to install it.

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

2 – 2 ZSH

Next, install zsh which is a shell designed for interactive use that will make you more productive.

```
brew install zsh
```

Next up, we are going to install Oh My Zsh. "Oh My Zsh which is an open source, community-driven framework for managing your zsh configuration. It will not make you a 10x developer…but you might feel like one"
— Robby Russell

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

2 – 3 Add Plugins to Oh My Zsh

You can add productivity plugins in zsh that will really make your life easier. The plugins that I have installed that I found useful are:

zsh-syntax-highlighting
It enables highlighting of commands whilst they are typed at a zsh prompt into an interactive terminal. This helps in reviewing commands before running them, particularly in catching syntax errors.

To install zsh-syntax-highlightin run:

```
git clone https://github.com/zsh-users/zsh-syntax-highlighting.git
${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting
```

zsh-autosuggestions
Suggests commands as you type based on history and completions.

To install zsh-autosuggestions run:

```
git clone https://github.com/zsh-users/zsh-autosuggestions ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
```

git
The git plugin provides many aliases and a few useful functions. It will also graphically show which branch you are on in your terminal

You do not need to install the git plugin, it comes with Oh My Zsh.

history-substring-search
history search feature, where you can type in any part of any command from history and then press chosen keys, such as the UP and DOWN arrows, to cycle through matches

To install history-substring-serach run:

```
git clone https://github.com/zsh-users/zsh-history-substring-search ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-history-substring-search
```

zsh-autocomplete
Auto completes and corrects typos:

To install zsh-autocomplete run:

```
git clone https://github.com/marlonrichert/zsh-autocomplete ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autocomplete
```

2 – 4 kube-ps1: Kubernetes prompt

A script that lets you add the current Kubernetes context and namespace configured on kubectl to your Bash/Zsh prompt strings (i.e. the $PS1).

To install run:

```
brew install kube-ps1
```

2 – 5 Customizing .zshrc

Once you have all the plugins installed, you will need to add them to Oh My Zsh. I've also created a couple of alias for commands I use frequently that I will show you how to add those to your environment.

Edit the file ~/.zshrc

```
vi ~/.zshrc
```

This is what the start of my ~/.zshrc file looks like :

```
# Path to your oh-my-zsh installation.
export ZSH="/Users/kevincollins/.oh-my-zsh"
export K_KEY=<IBM CLOUD API KEY>
alias k=kubectl
alias ic=ibmcloud
alias vi="vim"
alias pm='podman-machine ssh fedbox sudo podman'
source "/usr/local/opt/kube-ps1/share/kube-ps1.sh"


plugins=(

 zsh-syntax-highlighting

 zsh-autosuggestions
```

```
 git

 history-substring-search

 kubectl

)

source $ZSH/oh-my-zsh.sh

PS1='$(kube_ps1)'$PS1
```

```
export K_KEY=<IBM_CLOUD_APIKEY>
```

I added my IBM Cloud apikey to my environment variable as I use it many times a day.

```
The following Alias are for frequent commands that I have shortened.
alias k=kubectl
alias vi="vim"
alias ic="ibmcloud"
alias pm='podman-machine ssh fedbox sudo podman'
```

Configure ZSH to use the kube-ps1 tool

```
source "/usr/local/opt/kube-ps1/share/kube-ps1.sh"
```

Plugins to add to Oh My Zsh

```
plugins=(
zsh-syntax-highlighting
zsh-autosuggestions
git
kubectl
history-substring-search
zsh-autocomplete
)
```

After the plugins section, you will see a line that starts with:

```
source $ZSH/oh-my-zsh.sh
```

After this line, create a new line with the following:

```
PS1='$(kube_ps1)'$PS1
```

Next, we need to make the kube-ps1.sh script executable. Run the following command in your terminal.

```
chmod +x "/usr/local/opt/kube-ps1/share/kube-ps1.sh"
```

ZSH Theme – there are many you can choose from. I like spaceship the best.

To install spaceship, you will need to install it like following the same method we used to install the plugins.

```
git clone https://github.com/denysdovhan/spaceship-prompt.git
"$ZSH_CUSTOM/themes/spaceship-prompt" --depth=1

ln -s "$ZSH_CUSTOM/themes/spaceship-prompt/spaceship.zsh-theme"
"$ZSH_CUSTOM/themes/spaceship.zsh-theme"
```

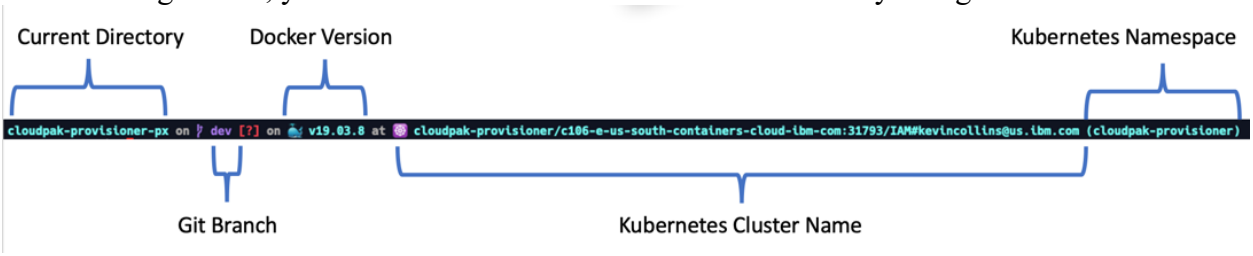
Next, we will need to run the following command to enable the terminal to write to the filesystem.

```
compaudit | xargs chmod g-w,o-w
```

For your changes to the .zshrc file to take place run:

```
source ~/.zshrc
```

After starting iterm2, your command line should look like this once you log into a cluster.



3 - CLIs

3 – 1 IBM Cloud

There are a number of CLIs that you will want to install that you will frequently use.

The first one is the IBM Cloud CLI. Install the IBM Cloud CLI with this command.

```
curl -fsSL https://clis.cloud.ibm.com/install/osx | sh
```

Next, you will need to install two frequently used plugins for IBM's container service and container registry service.

```
ibmcloud plugin install container-service
ibmcloud plugin install container-registry
```

3 -2 Kubernetes

Even though we will be using OpenShift and have the oc cli available, I still find myself frequently using the Kubernetes kubectl cli quite often. To install kubectl, enter this command.

```
brew install kubectl
```

3 -3 *OC*

In this tutorial, we will be using the OpenShift cluster we created in part 4. We will be using the OpenShift CLI extensively. Download the latest OpenShift CLI (oc) for your local operating system and OpenShift version with the link below.

OpenShift Container Platform 4 oc download link

By default, the oc command comes unsigned from RedHat and you will not be able to run the oc command from the terminal. To get around this, navigate to where you downloaded the oc command and right click open. This will ask you if you want to trust the oc command that you downloaded.

Next, we will need to copy the command to our /usr/local/bin directory by using the following command:

```
mv oc /usr/local/bin
```

4 - IDE

Now that we have our terminal and CLIs setup, we can now install an IDE. I use Visual Studio Studio code. If you have a different IDE you like then that is great. I know a lot of Python developers prefer PyCharm as an example. If you don't already have a favorite IDE, then Visual Studio Code is a good first choice. In the tutorial and videos, I will be using Visual Studio Code.

You can install Visual Studio Code by following the installation guide here:
https://code.visualstudio.com/docs/setup/mac

Installation

1. Download Visual Studio Code for macOS.
2. Open the browser's download list and locate the downloaded archive.
3. Select the 'magnifying glass' icon to open the archive in Finder.
4. Drag Visual Studio Code.app to the Applications folder, making it available in the macOS Launchpad.
5. Add VS Code to your Dock by right-clicking on the icon to bring up the context menu and choosing **Options**, **Keep in Dock**.

Visual studio code has a number of extensions to support the programming languages that you use for development. The microservices in this tutorial use Python so you will want to install the Python Extension for Visual Studio Code.

You can find instructions to install the python extension here:
https://code.visualstudio.com/docs/python/python-tutorial

5 Additional Tools

5 – 1 GitHub Desktop

I realize I will get a lot of flak from developer purists for using GitHub Desktop and not the command line. I have a love / hate relationship with GitHub from the terminal which has caused me a lot of pain and suffering with losing and overwriting code. While certainly a user error, I have resulting in using GitHub Desktop which I find very easy to use.

Install GitHub Desktop here:
https://desktop.github.com/

5 – 2 Postman

While developing, using, and sharing APIs, a great tool that I use pretty much every day is Postman. In the part 1 introduction, I included a postman link to all the APIs I created as part of CloudPak Provisioner. It's a great tool for calling and running APIs. We will be using Postman in this tutorial to test out the APIs we deploy.

Download and install postman here:
https://www.postman.com/downloads/

5 -3 Docker

The last tool we will install is Docker. While developing a cloud-native application, you will want to test your application locally before committing your code to GitHub which will start the deployment process.

In this tutorial, we will also go through building your application manually, so you can see all the steps that a CI/CD toolchain does for you. In order to experience this, you will need Docker installed locally on your machine.

You can download and install Docker by following these instructions:
https://docs.docker.com/docker-for-mac/install/