Zero to Cloud-Native with IBM Cloud

Part 8: Tutorial Application Code Base

Kevin Collins
Technical Sales Leader
IBM Cloud Enterprise Containers – Americas

Kunal Malhotra
Cloud Platform Engineer
IBM Cloud MEA

1 - Codebase Overview

All the code for the zero-to-cloud-native tutorial is stored in GitHub. As with good microservice design, each microservice has its own git repository to keep microservice independence and as you will see will also assist in deploying the microservices with a Continuous Integration / Continuous Delivery (CICD) pipeline.

With the zero-to-cloud-native application, you will be developing an API and Web based application that will provide a framework to create APIs to perform common tasks on IBM Cloud and even automate common tasks. With this tutorial, we will be implementing three API calls:

1) Retrieve an OpenShift login token and login URL.
2) Retrieve a list of supported OpenShift versions on IBM Cloud.
3) Enable worker nodes so you can SSH into the worker nodes.

There are six microservices and seven GitHub code repositories that make up the zero-to-cloud-native application.

1) **API Frontend** – Python flask based microservice that serves as the frontend for calling the three APIs.
   https://github.com/kmcolli/api-frontend-02cn

2) **Web Frontend** – Web based frontend for invoking the APIs.
   https://github.com/kmcolli/web-frontend-02cn

3) **Utility** – a utility microservice that performs common tasks that many of the other microservices use.
   https://github.com/kmcolli/utility-02cn

4) **Enable SSH** – enables SSH on IBM Cloud Managed OpenShift Worker Nodes
   https://github.com/kmcolli/enable-node-SSH-02cn

5) **OCP Realtime** – 'realtime' APIs that will return an OpenShift login token and a list of currently supported OpenShift versions.
https://github.com/kmcolli/ocp-realtime-02cn

6) **Load OCP Versions** – cronjob implementation that will retrieve a list of supported versions of IBM Cloud Managed OpenShift versions and store the results in Redis for quick access.
https://github.com/kmcolli/load-OCP-versions-02cn

7) **Secrets** – secrets template yaml file. The zero-to-cloud-native application will require that a number of Kubernetes secrets are set. This template file will need to updated with parameters for your environment.
https://github.com/kmcolli/secrets-02cn

2 - Setting up GitHub

Before you can build and deploy the code, you will need to clone the above repositories and with the exception of the Secrets repo, add them to your GitHub account. The first thing you will want to do is clone each code repository listed above.

2 -1 Clone Zero to Cloud Native Repository

Using your command line, navigate to where you want to store your code. Create a directory such as zero-to-cloud-native and then switch to the directory.

```
mkdir zero-to-cloud-native
cd zero-to-cloud-native
```

While under the directory you just created, you will need to clone each code repository.

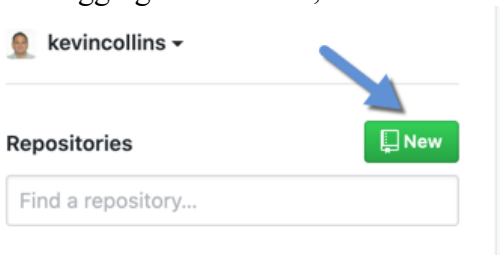
Example using the command line:

```
git clone https://github.com/kmcolli/api-frontend-02cn
```

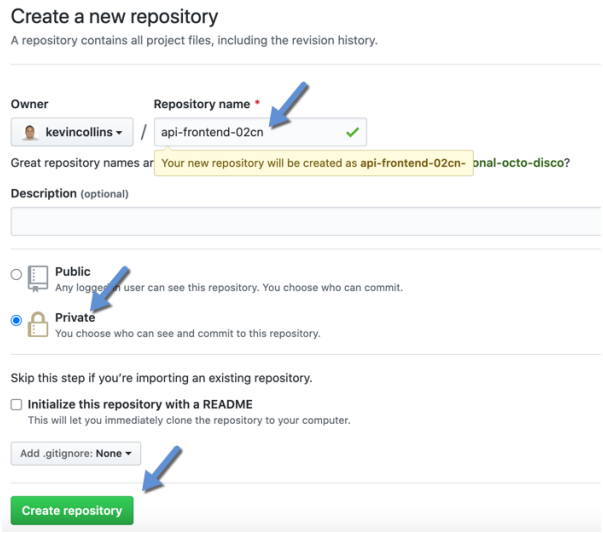
2 – 2 Add the cloned repositories into your own GitHub account.

To enable automatic deployment using CI/CD toolchains, you will need to add the repositories you just cloned to your GitHub account. Using **api-frontend-02cn** as example, you will first need to create a new repository on GitHub.

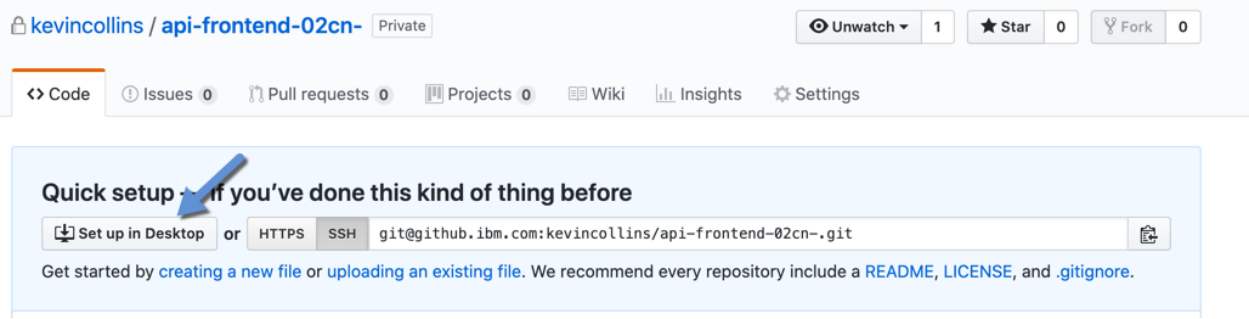
After logging into GitHub, click on New under repositories.



On the next string, enter a name for your repository, you can keep the name the same **api-frontend-02cn**, by default I keep the repository Private, and then click Create repository.



As I previously mentioned, I have a personal battle with the GitHub CLI so I have resulted in using GitHub Desktop. After creating the repository on the GitHub website, the next screen will allow you to click Set up in Desktop.



This will add the new repository to your GitHub Desktop application.

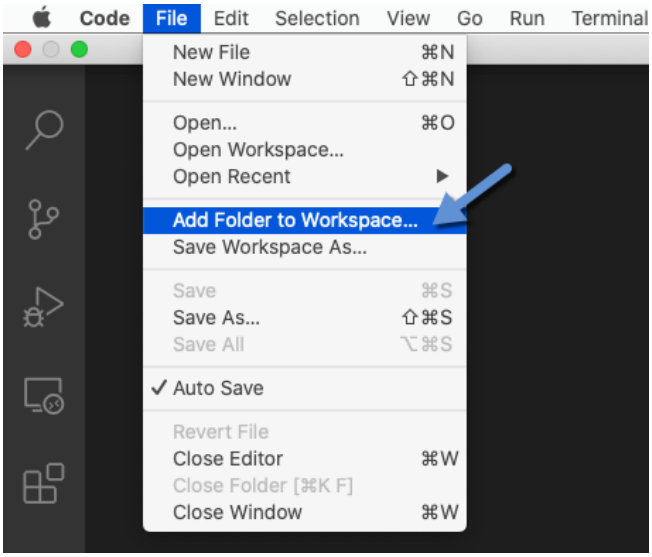
Repeat the same process for all other repositories.

3 - Setting up Visual Studio Code

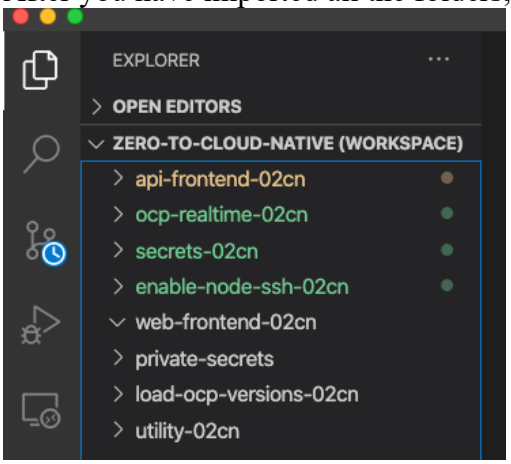
Now that you have cloned each Git repository and created your own repository, the next step is to import the code into your IDE. For this tutorial, I will be using Visual Studio Code.

One of the nice features of Visual Studio code is the concept of workspaces. Workspaces allow you to group different folders together that make up an application. In this tutorial, we will be creating a zero-to-cloud-native workspace.

After starting Visual Studio Code, click on File and then select Add Folder to Workspace. Select all seven code repositories.



After you have imported all the folders, you should see the following:



Note that you will see that I have two folders for secrets. One is secrets-02cn which is the secrets template that I am syncing with GitHub that you cloned. I also have a private secrets folder, which stores private secrets such as my IBM Cloud API Key that I am keeping local or storing in a private GitHub Repository. Navigate to your filesystem and directory where you are

storing all source code. Create a new directory called private-secrets. Copy the file zero-to-cloud-native-secrets file that is under the secrets-02cn directly to the newly created private-secrets directory. You will be editing this file in the next section.

4 - Code Layout

Now that we have Visual Studio Code setup for the zero to cloud native tutorial, you can start exploring the code.

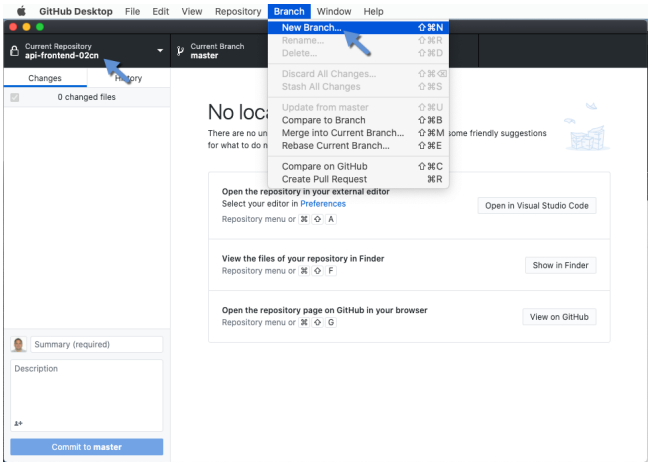
Each repository has the same folder layout. You will see the following folder structure:

- **app** – contains the application source code.
- **deployments** – contains Kubernetes deployment files
- **pipeline** – contains 'Classic' pipeline configuration scripts – use for our classic CI/CD toolchain
- **.tekton** – contains 'Tekton' pipeline configuration scripts – used for our Tekton CI/CD toolchains.
- **Dockerfile** – Dockerfile used to build the image for the microservice configuration – Python requirements file
- **.dockerignore** – file that tells docker which files to ignore while building an image

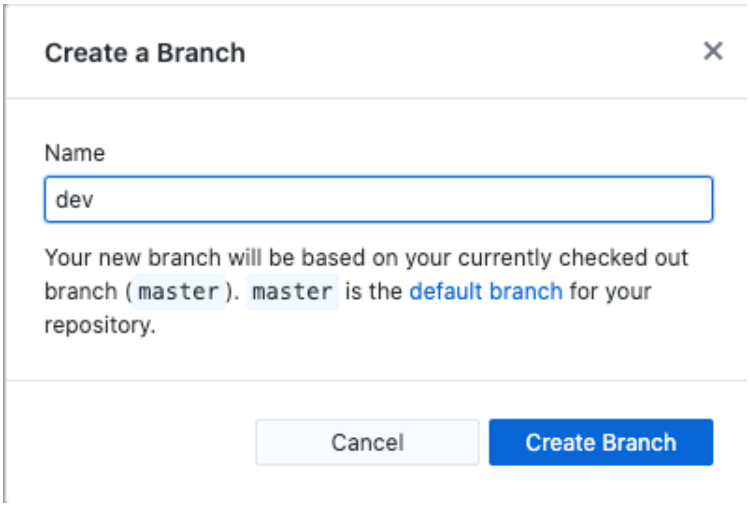
5 - Git Branches

A best practice that I like to follow is having a unique git branch for each Git repository for each environment. For example, if you have three environments, Dev, Staging and Production, you would have three branches in GitHub – Dev, Staging, and Production. In this tutorial, we will be simulating the dev environment for our CI/CD pipelines which means we need a dev branch for our Git Repository.

To create a new branch using GitHub Desktop, change the repository to the one you want to create a new branch.



On the next screen, simply type in the name of the branch you want to create, in our case dev. Note that when you create a new branch, all the files from the master branch will automatically be copied over to the new branch.



Now that we have setup and configured all the services for the tutorial application and imported the source code into our development environment, we can finally deploy the application! The next section will go through deploying the application.