How to

Build a Serverless Spell Checker with OpenWhisk and Docker

sjfink
Created on June 17, 2016 / Modified on June 20, 2016

0

This post describes a little programming exercise to build a serverless spell checker using IBM Bluemix OpenWhisk. We assume you already have a rough understanding of the OpenWhisk programming model. If not, spend some time exploring our Developer Center to learn the basics.

Motivation

In a number of use cases, I'd like to spell check documents in a serverless, event-driven fashion. For example, I'd like to spell check markdown documents when someone commits changes to the OpenWhisk github repository. OpenWhisk already supports **feeds** to listen for commit messages from a git repository. So I'll need a spellcheck **action**, and then I can set up a **rule** to invoke the spellcheck action on every commit.

So what's the easiest way to implement a spellcheck action?

I'm an old man, and I still rely on good old Unix utilities such as `aspell` and `ispell` to spell check my documents. In this post, I will show how I used OpenWhisk Docker actions to turn GNU aspell into a whisk action, which I can invoke in a serverless flow.

Why Docker actions?

Before diving into code, let's talk big picture for a moment. OpenWhisk and other serverless platforms support a number of *first-class* language environments. For example, in OpenWhisk, you can simply upload Javascript or Swift code to define an **action**, and OpenWhisk will load your code into an appropriate language-specific environment, and run it.

These language environments rely on having a **portable executable format** to describe your code. This is relatively simple for interpreted languages such as Javascript or Python, where the program text itself serves as a portable executable format. For Java, Scala, and other JVM-based languages, JVM bytecode serves as a portable format.

But suppose you want to use code from, say, a C program. If you compile a C program locally, you produce binary code which will probably not run correctly in a different environment than your local host. If you want to upload your binary code and run it as a serverless OpenWhisk **action**, we need some way to package your binary code in a portable executable format.

Enter Docker! A Dockerfile specifies a portable Linux container environment, into which you can install binaries and do practically anything you like. A Docker image represents an environment which should run identically anywhere docker itself runs. That is, the Docker image provides a portable executable format that can describe most code you would run on Linux.

For this reason, OpenWhisk allows you write **actions** which are implemented by Docker images that follow a few conventions.

Next, we'll step through an exercise to build our spellcheck action using OpenWhisk docker actions.

Let's code!

Step 1: Configure the `wsk` command-line tool with your Bluemix account (the account is free!)

Step 2: Download a skeleton template for an OpenWhisk docker action.

```
$ wsk sdk install docker
```

After this step, you will have a directory `./dockerSkeleton`, which contains the source code for a sample OpenWhisk action. We'll use this sample code as our starting point. Change to that directory and you should see the following files:

```
$ cd dockerSkeleton
$ ls
Dockerfile      README.md       buildAndPush.sh client          server
```

> *Implementation details you don't really need to know:*
>
> *Currently the OpenWhisk runtime instantiates each action in a docker container, which the runtime system manages. The runtime communicates with the action container over HTTP. In particular, the action container must implement a simple web server which responds to a POST request on the `/run` endpoint by running the action and returning the action result in the HTTP response.*
>
> *The `server` directory contains the implementation of a small Nodejs web server which provides the HTTP support for this protocol. The server will call a single command called `/blackbox/client/action`. You need not worry anything about the HTTP protocol or the web server — to implement an action you must simply arrange to have an appropriate executable command installed at `/blackbox/client/action`.*
>
> *Currently the first part of the Dockerfile installs the web server for you, and the last command runs it. We will probably refactor the Dockerfile later to move the web server into a separate base image. This doesn't change anything from a user's perspective.*

Step 3: Create a bash script which implements the spell check function.

An OpenWhisk action consumes a JSON object as input and produces a JSON object as output. To create a Docker action, we need a single executable command `action` that consumes a JSON object as input, and produces a JSON object as output.

Design Decision: I've decided that my action will accept a JSON object with a single attribute called b64, which should hold the base64 encoded version of the file to spell check. I like the base64 format because it is easy to pass in a JSON object without worrying about escaping quotes and such.

Design Decision: I've decided that my action will return a JSON object with a single attribute called `result`, which will hold a list of the misspelled words in the input file.

Having made these design decisions, now I'll write a simple bash script which implements the action. Here it is:

```
#!/bin/bash
#
# This script expects one argument, a String representation of a JSON
# object with a single attribute called b64.   The script decodes
# the b64 value to a file, and then pipes it to aspell to check spelling.
# Finally it returns the result in a JSON object with an attribute called
# result
#
FILE=/tmp/output.txt

# Parse the JSON input ($1) to extract the value of the 'b64' attribute,
# then decode it from base64 format, and dump the result to a file.
echo $1 | sed -e 's/b64.//g' \
        | tr -d '"' | tr -d ' ' | tr -d '}' | tr -d '{' \
        | base64 --decode >& $FILE

# Pipe the input file to aspell, and then format the result on one line with
# spaces as delimiters
RESULT=`cat $FILE | aspell list | tr '\n' ' ' `

# Return a JSON object with a single attribute 'result'
echo "{ \"result\": \"$RESULT\" }"
```

Let's test it locally. First let's create an input file called input:

```
Cat dog
elephent lion
mooose bug
```

And let's test the script, feeding it a JSON object with the correct format:

```
$ ./action.sh "{\"b64\":\"`base64 input`\" }"
{ "result": "elephent mooose " }
```

So far so good.

Step 4: Wrap the script in a Docker container.

The sample SDK we downloaded earlier contains an example of an action wrapped in a Docker container. In particular, the sample SDK includes a `Dockerfile` that builds the C program in `client/example.c` and installs the binary as `/blackbox/client/action`.

The key line in the sample Dockerfile is:

```
RUN cd /blackbox/client; gcc -o action example.c
```

Instead of compiling `example.c` and installing the binary as an action, we'll change the `Dockerfile` to install `aspell` into the Linux environment, and then install our `action.sh` script as the executable action command.

To do so, we delete the `RUN` command above, and insert the following commands into the Dockerfile:

```
RUN apt-get install -y aspell
RUN rm -f /blackbox/client/action
ADD action.sh /blackbox/client/action
```

At this point, we have created a `Dockerfile` that will build a Docker image which OpenWhisk can manage as an **action**. Before we upload the action to OpenWhisk, let's test the docker container locally.

> *Tip for Mac users using `docker-machine`:*
> *You currently have to futz with network settings on your Mac in order to communicate via HTTP from the host to the `docker` daemon running in a `docker-machine`. To make a long story short, run this script on your Mac before trying to talk to a docker container over HTTP.*

First we'll build the docker image (naming it `test/test`), and then run it locally (giving the container the name `testing`):

```
$ docker build -t test/test .
$ docker run -d --name testing test/test
```

As described earlier, the container implements a simple web server which is listening to the POST endpoint `/run`. The /run endpoint expects to receive a JSON object with a parameter called "`value`", which holds the JSON object which is the input to the **action**. So we'll do a POST to the endpoint in the docker container, and see if it works.

First we determine the IP address of the running container, and then we POST to port 8080 on the container to test running the action.

```
$ IP=`docker inspect --format '{{ .NetworkSettings.IPAddress }}' testing`
$ echo $IP
172.17.0.2
$ ARG="{\"b64\":\"`base64 input`\" }"
$ echo $ARG
{"b64":"Q3F0RGRvZyAKZWxlcGhlbnQgbGlvbgptb29vc2UgYnVnCg==" }
$ curl -X POST -H "Content-Type: application/json" -d "{\"value\":$ARG }" http://$IP:8080/run
{"value":"elephent mooose "}
```

Brilliant! We've done all the hard stuff. The rest is easy.

Step 5: Upload the docker image to DockerHub.

My DockerHub account is `sjfink`, and I'm going to name the image `sjfink/spellaction`:

```
$ docker login
Username: sjfink
Password: ....
$ ./buildAndPush.sh sjfink/spellaction
```

Step 6: Create an OpenWhisk **action** from the docker container, and test it.

I'm going to call my action *spell*:

```
$ wsk action create --docker spell sjfink/spellaction
$ wsk action invoke --blocking --result spell --param b64 `base64 input`
{
    "value": "elephent mooose "
}
```

Note that the very first invocation of the new action will take a long time (perhaps 30 seconds), since the OpenWhisk runtime needs to fetch the docker image from DockerHub. Subsequent invocations should be much faster, though not as fast as using an OpenWhisk supported language like Javascript.

Epilogue

We're done — now we have an action called **spell** that we can use in any event-driven flows we like, just like any other OpenWhisk action. My next steps will be to build little bots that respond to github commits to check spelling on changed markdown files.

Note that you, the reader, can skip straight to step 6 if you like : since I've published `sjfink/spellaction` on DockerHub, you can create your own actions based on this.

You can also get all the files mentioned in this post from GitHub.

I'd encourage you to build your own actions and publish them yourselves.

Happy whisking!

By sjfink