InterConnect
2017

March 19–23
MGM Grand &
Mandalay Bay
Las Vegas, NV

IBM

Hands-on Lab
Session 2720
Application deployment automation in IBM Bluemix using Docker, IBM Bluemix Container Service and IBM Bluemix Continuous Delivery Service

Jesus Arteche, CASE team, IBM Cloud
Jesus Almaraz, CASE team, IBM Cloud

You must include the first two pages of this template.

This document is current as of the initial date of publication and may be changed by IBM at any time.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

Table of Contents

Objectives

The aim of this lab is to drive the attendees through:

- A very basic microservices architecture.
- The local deployment and test of this microservices architecture.
- The containerization of the architecture.
- The local deployment and test of the containerized architecture.
- The remote deployment and test of the architecture in the IBM Bluemix public cloud by using the IBM Bluemix Container Service.
- The automation of the deployment process by using the IBM Bluemix Continuous Delivery Service.
- The creation of an agile and continuous zero-downtime CI/CD process by integrating Active Deploy in the automated deployment process.
- End to end CI/CD flow demo.

Pre-requisites

In order to carry out the lab we will need the following beforehand:

- Bluemix account - https://console.ng.bluemix.net/registration/
  (Create space in US South region)

- GitHub account - https://github.com/join?source=header-home

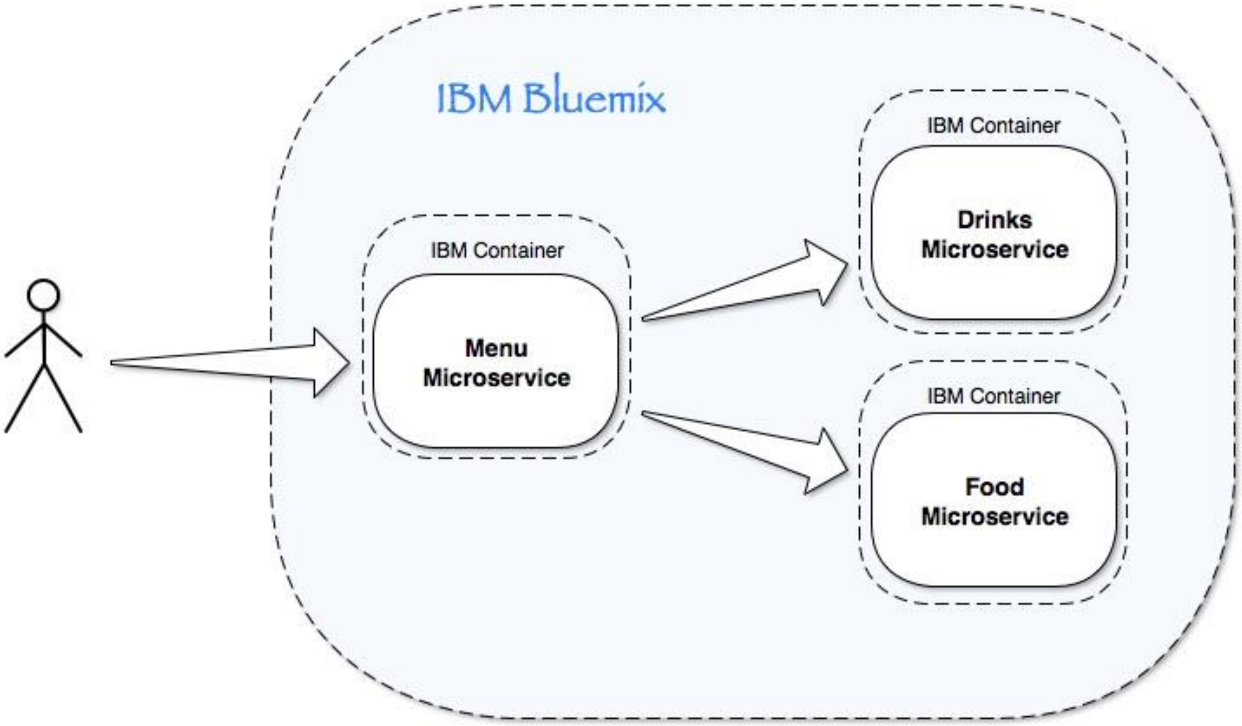
Introduction

During this lab, we will use a simple Java application which will display the 2017 IBM InterConnect menu consisting of drinks and food. This Java application is made up of three different microservices:

- **Menu** microservice: This microservice will request the drinks and food menus and display them as a single menu to the 2017 IBM InterConnect user.

- **Drinks** microservice: This microservice will retrieve the available drinks for the 2017 IBM InterConnect menu.

- **Food** microservice: This microservice will retrieve the available food for the 2017 IBM InterConnect menu.



These microservices are simple Java Spring Boot applications which we will containerize and deploy onto Bluemix using the IBM Bluemix Container Service. We will firstly deploy these microservices on a manual fashion to later use the IBM Bluemix Continuous Delivery Service to automate the same deployment process. We will finally explore Active Deploy capabilities for an agile and zero-downtime CI/CD process.

Java Application

The 2017 IBM InterConnect menu application is a simple Java application implemented using Spring Boot technology. Spring framework helps you build web applications. It takes care of dependency injection, handles transactions, implements an MVC framework and provides foundation for the other Spring frameworks (including Spring Boot).

While you can do everything in Spring without Spring Boot, Spring Boot helps you get things done faster:

- simplifies your Spring dependencies, no more version collisions
- can be run straight from a command line without an application container
- build more with less code - no need for XML, not even web.xml, auto-configuration
- useful tools for running in production, database initialization, environment specific configuration files, collecting metrics

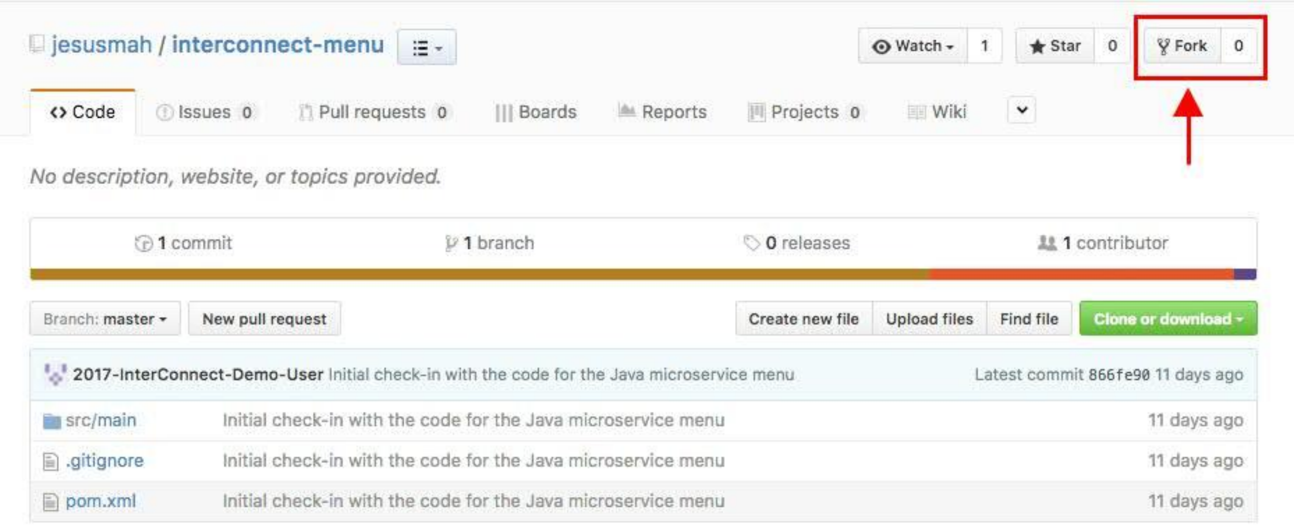
In this section, we will fork the three Java microservices GitHub repos, so that we get our own personal copy of the code to work with, modify, etc. without getting interfered by anyone. We will also deploy the app locally in our computers to see how the app looks like and get familiar with it.

GitHub repositories for the Java microservices:

- **Menu** microservice: https://github.com/jesusmah/interconnect-menu
- **Drinks** microservice: https://github.com/jesusmah/interconnect-drinks
- **Food** microservice: https://github.com/jesusmah/interconnect-food

Execute each of the following steps for each of the three microservices:

1. Fork GitHub repository
   1.1. Open the GitHub repo by clicking the corresponding link above.
   1.2. Click on the Fork button on the top right corner of the browser.

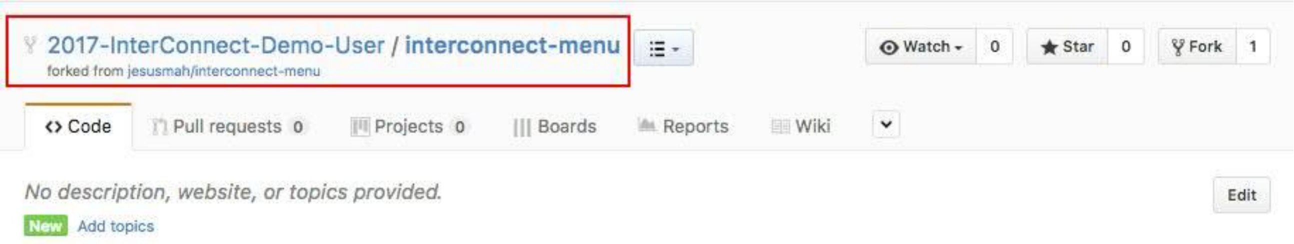


No description, website, or topics provided.

2017-InterConnect-Demo-User Initial check-in with the code for the Java microservice menu — Latest commit 866fe90 11 days ago

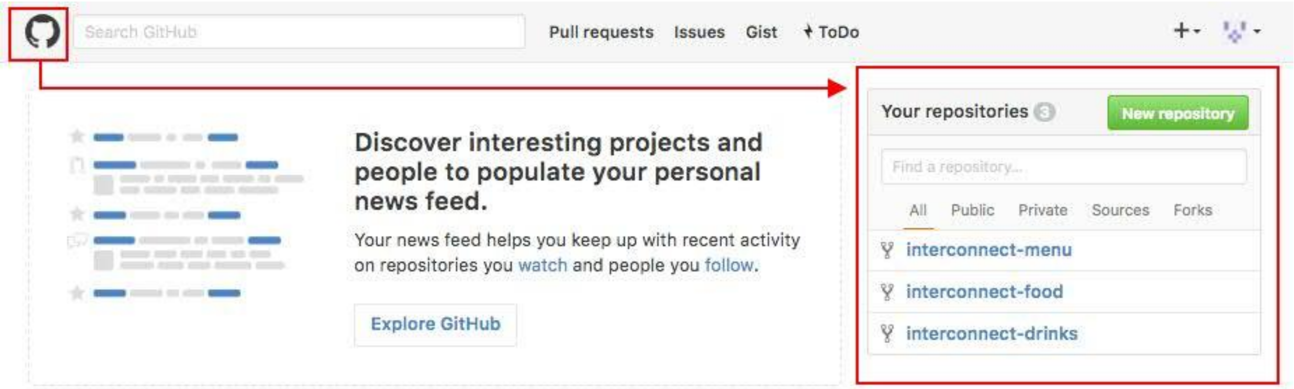
src/main	Initial check-in with the code for the Java microservice menu	11 days ago
.gitignore	Initial check-in with the code for the Java microservice menu	11 days ago
pom.xml	Initial check-in with the code for the Java microservice menu	11 days ago

You should now see the Java project in your GitHub account:


2017-InterConnect-Demo-User / interconnect-menu
forked from jesusmah/interconnect-menu
Watch 0
Star 0
Fork 1
Code
Pull requests 0
Projects 0
Boards
Reports
Wiki
Edit
New
Add topics

No description, website, or topics provided.

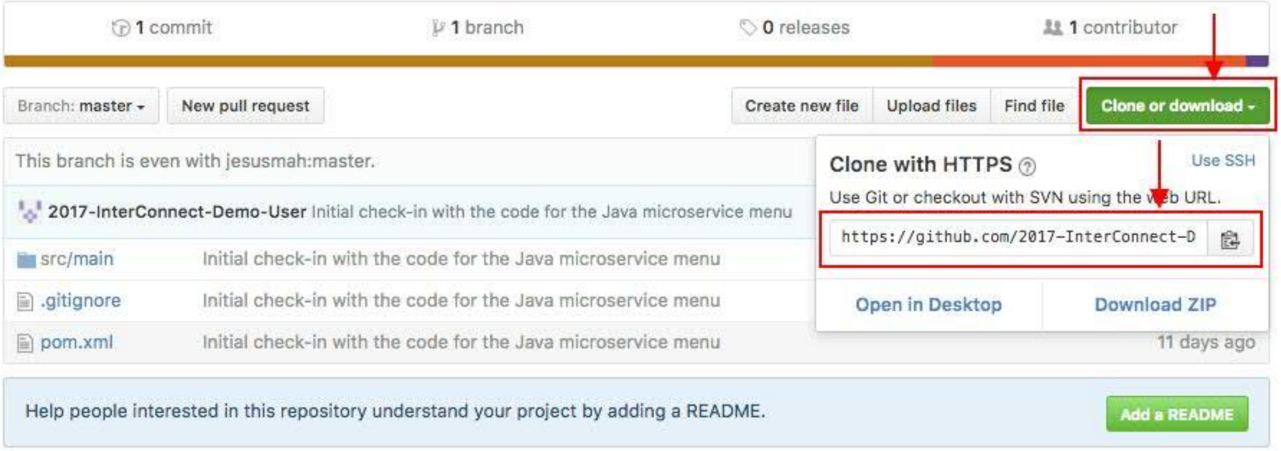
   1.3. **Repeat previous steps for the other two GitHub repositories**.
   After forking the three GitHub repository, your GitHub repository should look like:



Discover interesting projects and people to populate your personal news feed.

Your news feed helps you keep up with recent activity on repositories you watch and people you follow.

2. Clone or download the code to your workstation
   2.1. Click on the Clone or download button and copy the address



This branch is even with jesusmah:master.

2017-InterConnect-Demo-User Initial check-in with the code for the Java microservice menu

src/main	Initial check-in with the code for the Java microservice menu
.gitignore	Initial check-in with the code for the Java microservice menu
pom.xml	Initial check-in with the code for the Java microservice menu

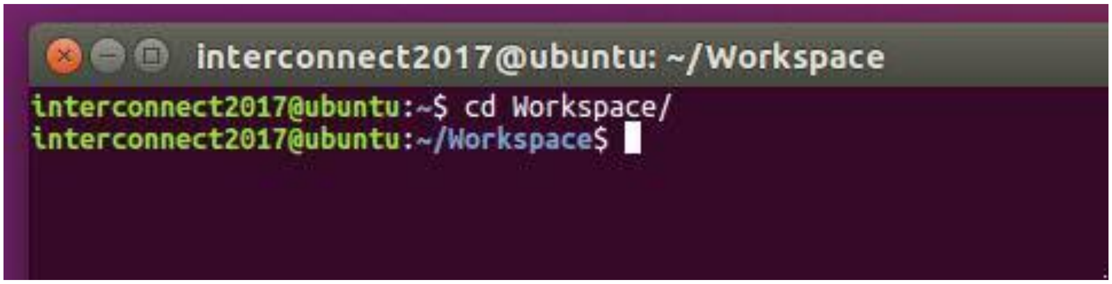
11 days ago

Use Git or checkout with SVN using the web URL.

https://github.com/2017-InterConnect-D

Help people interested in this repository understand your project by adding a README.

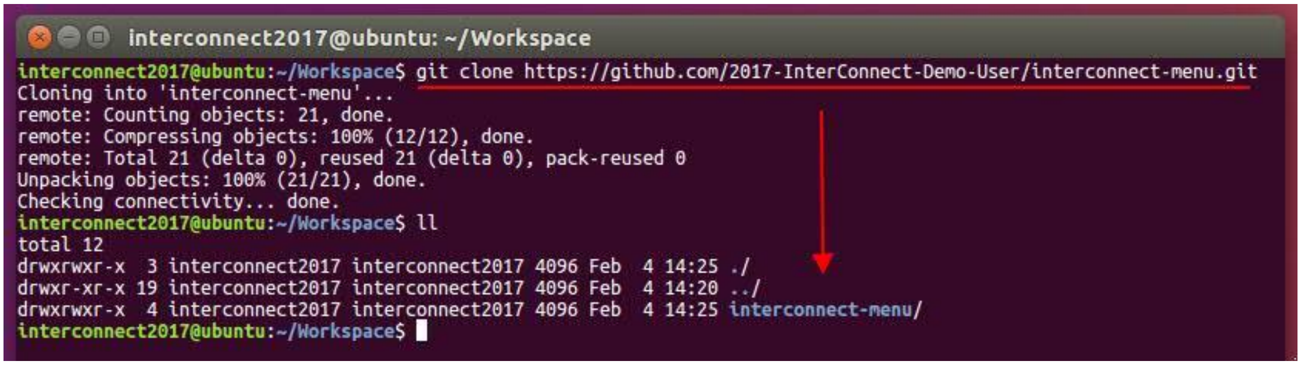
   2.2. Open a terminal and change directory to your workspace



```
interconnect2017@ubuntu: ~/Workspace
interconnect2017@ubuntu:~$ cd Workspace/
interconnect2017@ubuntu:~/Workspace$ 
```

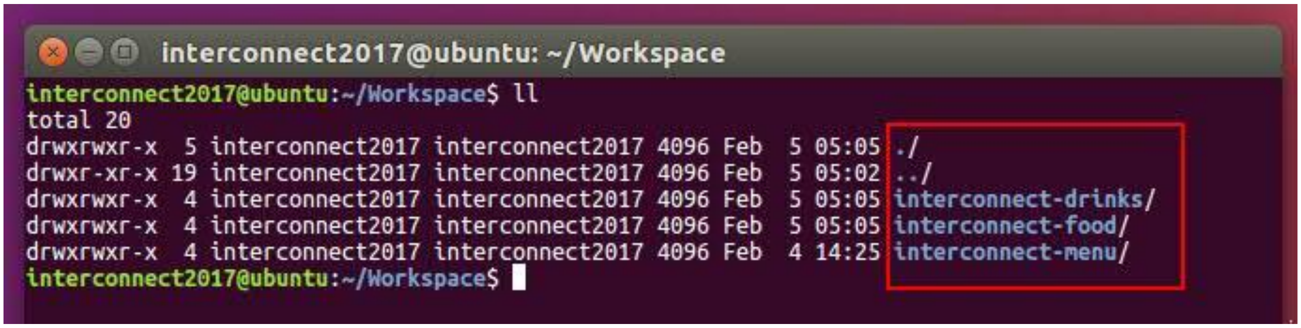
   2.3. Clone or download the code from your GitHub repository by executing

```
$ git clone your-repository-url
```



```
interconnect2017@ubuntu: ~/Workspace
interconnect2017@ubuntu:~/Workspace$ git clone https://github.com/2017-InterConnect-Demo-User/interconnect-menu.git
Cloning into 'interconnect-menu'...
remote: Counting objects: 21, done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 21 (delta 0), reused 21 (delta 0), pack-reused 0
Unpacking objects: 100% (21/21), done.
Checking connectivity... done.
interconnect2017@ubuntu:~/Workspace$ ll
total 12
drwxrwxr-x  3 interconnect2017 interconnect2017 4096 Feb  4 14:25 ./
drwxr-xr-x 19 interconnect2017 interconnect2017 4096 Feb  4 14:20 ../
drwxrwxr-x  4 interconnect2017 interconnect2017 4096 Feb  4 14:25 interconnect-menu/
interconnect2017@ubuntu:~/Workspace$ 
```

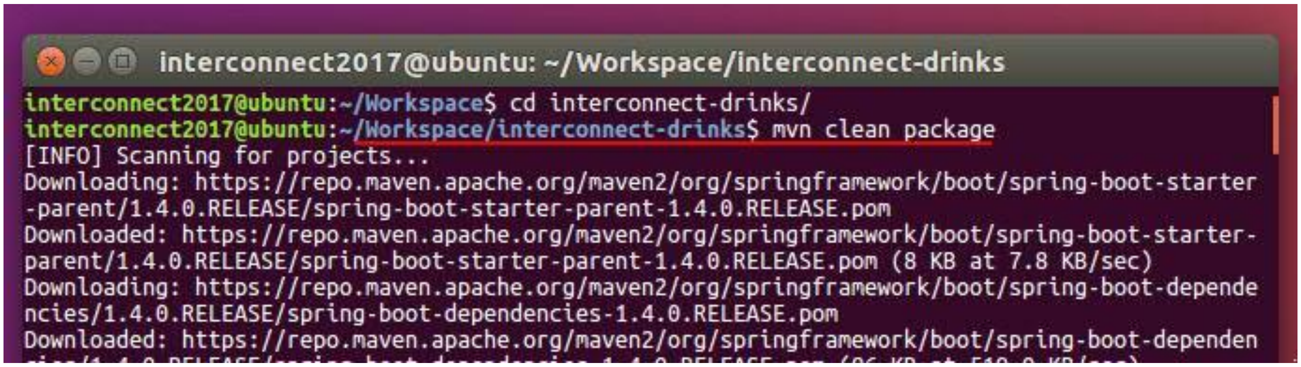
2.4. **Repeat previous steps for the other two GitHub repositories.**
Your workspace should now look like:



```
interconnect2017@ubuntu: ~/Workspace
interconnect2017@ubuntu:~/Workspace$ ll
total 20
drwxrwxr-x  5 interconnect2017 interconnect2017 4096 Feb  5 05:05 ./
drwxr-xr-x 19 interconnect2017 interconnect2017 4096 Feb  5 05:02 ../
drwxrwxr-x  4 interconnect2017 interconnect2017 4096 Feb  5 05:05 interconnect-drinks/
drwxrwxr-x  4 interconnect2017 interconnect2017 4096 Feb  5 05:05 interconnect-food/
drwxrwxr-x  4 interconnect2017 interconnect2017 4096 Feb  4 14:25 interconnect-menu/
interconnect2017@ubuntu:~/Workspace$
```

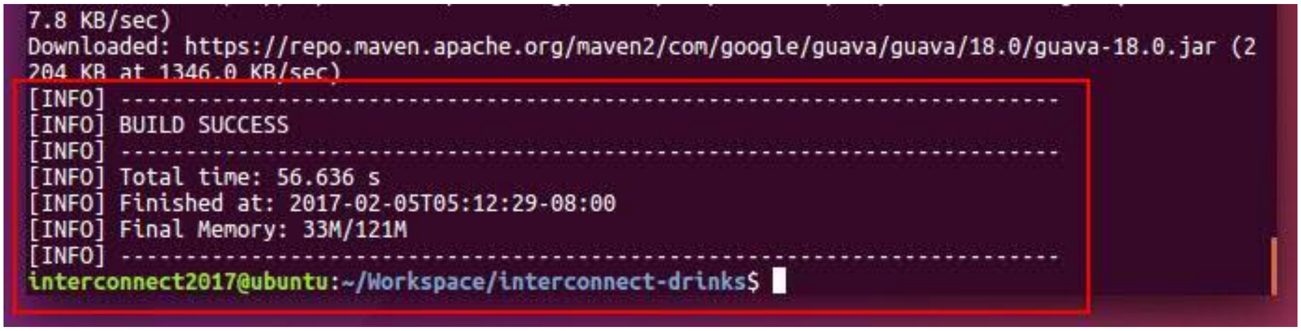
3. Build Java applications
   3.1. Change directory into one of the three projects
   3.2. Build the Java application by executing

```
$ mvn clean package
```



```
interconnect2017@ubuntu: ~/Workspace/interconnect-drinks
interconnect2017@ubuntu:~/Workspace$ cd interconnect-drinks/
interconnect2017@ubuntu:~/Workspace/interconnect-drinks$ mvn clean package
[INFO] Scanning for projects...
Downloading: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter
-parent/1.4.0.RELEASE/spring-boot-starter-parent-1.4.0.RELEASE.pom
Downloaded: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-starter-
parent/1.4.0.RELEASE/spring-boot-starter-parent-1.4.0.RELEASE.pom (8 KB at 7.8 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-depende
ncies/1.4.0.RELEASE/spring-boot-dependencies-1.4.0.RELEASE.pom
Downloaded: https://repo.maven.apache.org/maven2/org/springframework/boot/spring-boot-dependen
```

After the build process has finished, you should see the following:

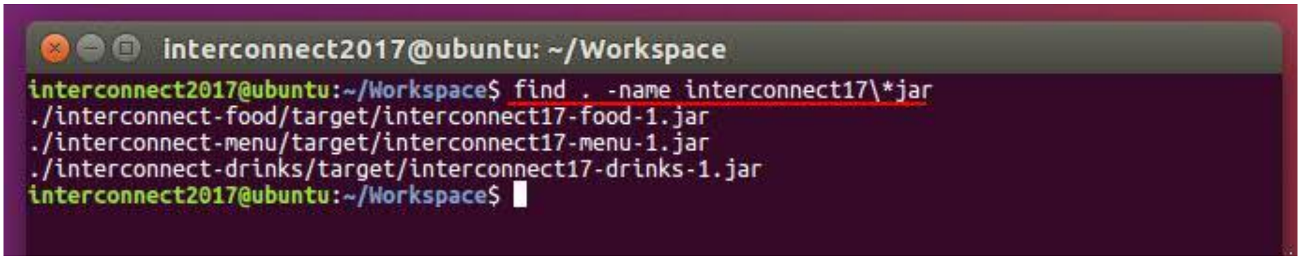


```
7.8 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/com/google/guava/guava/18.0/guava-18.0.jar (2
204 KB at 1346.0 KB/sec)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 56.636 s
[INFO] Finished at: 2017-02-05T05:12:29-08:00
[INFO] Final Memory: 33M/121M
[INFO] ------------------------------------------------------------------------
interconnect2017@ubuntu:~/Workspace/interconnect-drinks$
```

3.3. **Repeat previous steps for the other two GitHub repositories.**
If you execute the following in your workspace directory,

```
$ find . -name interconnect17\*jar
```

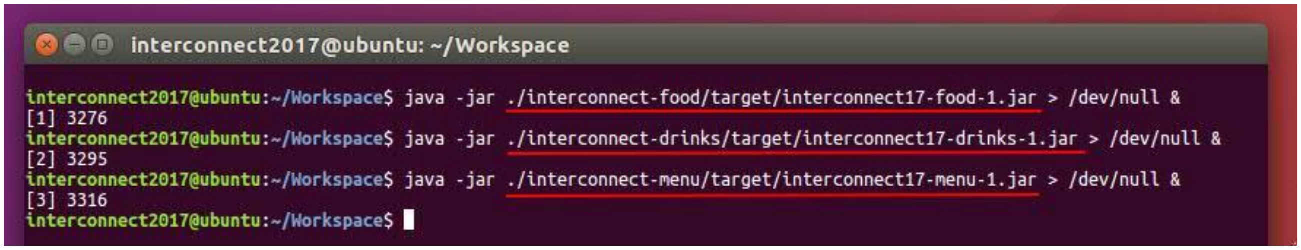
you should see the three Java application built jar files:



```
interconnect2017@ubuntu:~/Workspace$ find . -name interconnect17\*jar
./interconnect-food/target/interconnect17-food-1.jar
./interconnect-menu/target/interconnect17-menu-1.jar
./interconnect-drinks/target/interconnect17-drinks-1.jar
interconnect2017@ubuntu:~/Workspace$
```

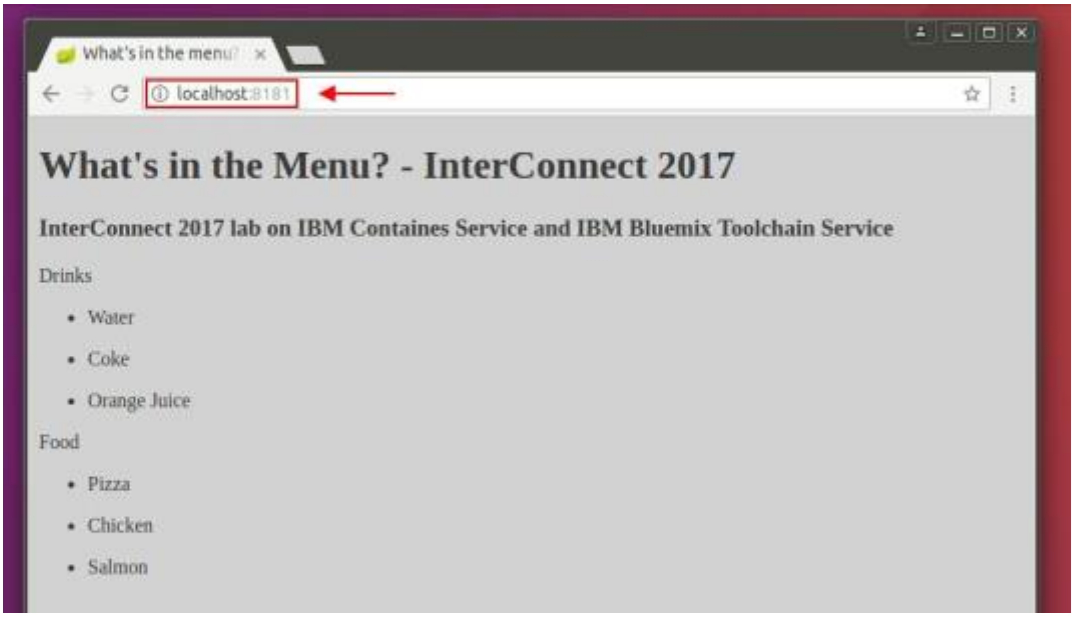
4. Execute the 2017 IBM InterConnect menu Java app
   4.1. Execute each of the three jar files in the background with their standard output redirected to */dev/null* by issuing the following command:

```
$ java -jar jar-file > /dev/null &
```



```
interconnect2017@ubuntu:~/Workspace$ java -jar ./interconnect-food/target/interconnect17-food-1.jar > /dev/null &
[1] 3276
interconnect2017@ubuntu:~/Workspace$ java -jar ./interconnect-drinks/target/interconnect17-drinks-1.jar > /dev/null &
[2] 3295
interconnect2017@ubuntu:~/Workspace$ java -jar ./interconnect-menu/target/interconnect17-menu-1.jar > /dev/null &
[3] 3316
interconnect2017@ubuntu:~/Workspace$
```

4.2. Check the 2017 IBM InterConnect menu application is working by pointing your browser to *localhost:8181*



What's in the Menu? - InterConnect 2017

InterConnect 2017 lab on IBM Containes Service and IBM Bluemix Toolchain Service

Drinks

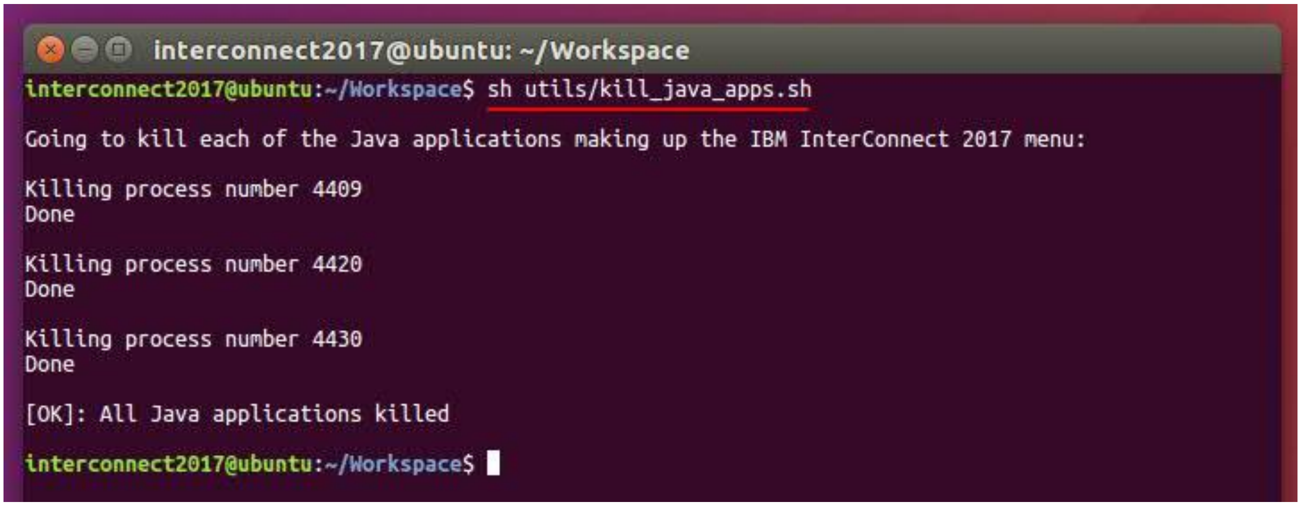
- Water
- Coke
- Orange Juice

Food

- Pizza
- Chicken
- Salmon

5. Kill the local 2017 IBM InterConnect menu Java application
    5.1. Kill each of the previously spawn Java processes by executing the following script within the utils folder in your workspace

```
$ sh utils/kill_java_apps.sh
```



```
interconnect2017@ubuntu:~/Workspace$ sh utils/kill_java_apps.sh

Going to kill each of the Java applications making up the IBM InterConnect 2017 menu:

Killing process number 4409
Done

Killing process number 4420
Done

Killing process number 4430
Done

[OK]: All Java applications killed

interconnect2017@ubuntu:~/Workspace$
```

Docker and Containers

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment. Key Docker containers properties:

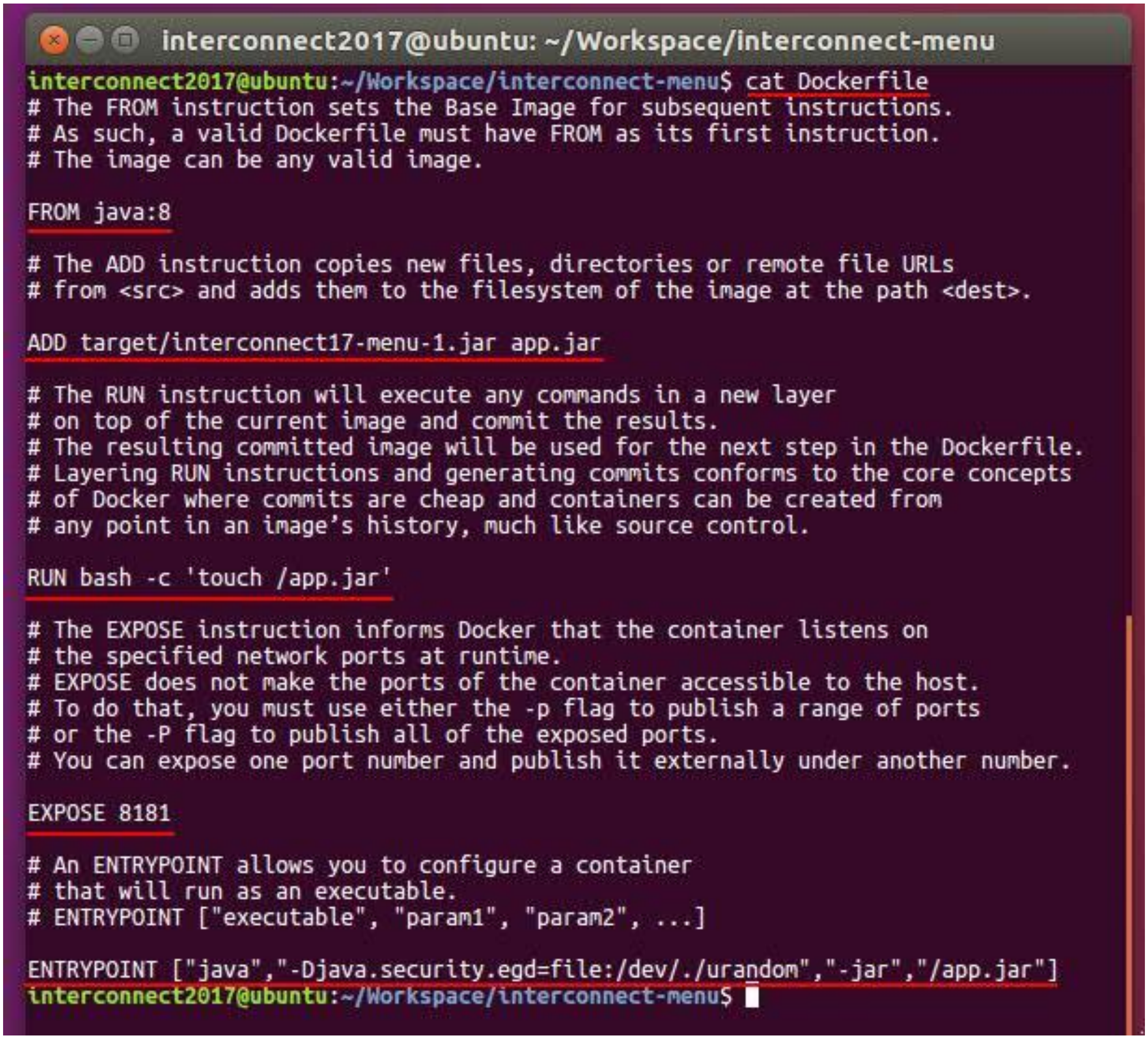
- **Lightweight** - Containers running on a single machine share the same operating system kernel; they start instantly and use less RAM. Images are constructed from layered filesystems and share common files, making disk usage and image downloads much more efficient.

- **Open** - Docker containers are based on open standards, enabling containers to run on all major Linux distributions and on Microsoft Windows -- and on top of any infrastructure.

- **Secure** - Containers isolate applications from one another and the underlying infrastructure, while providing an added layer of protection for the application.

In this section, we will go through the process of containerizing our three Java applications by using a Dockerfile. Later, we will create the Docker images for the three Java apps from their Dockerfiles to finally run the 2017 IBM InterConnect dockerized menu app locally in our workstations.

1. Containerize the Java applications
    1.1. Verify and understand Dockerfiles in each of the projects.

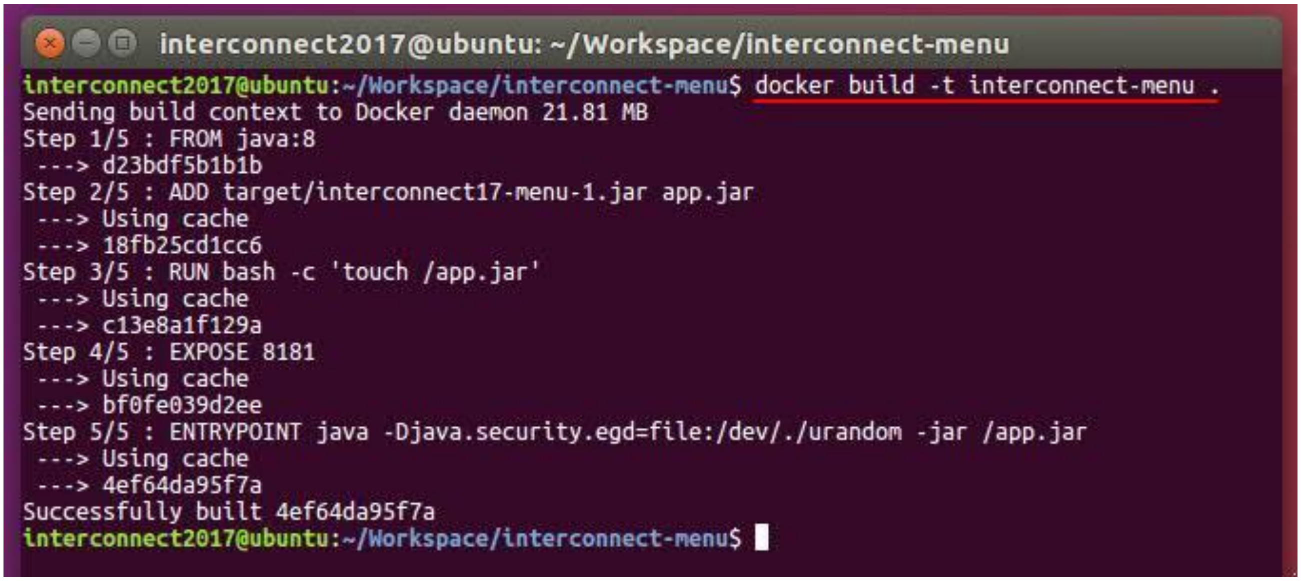


```
interconnect2017@ubuntu:~/Workspace/interconnect-menu$ cat Dockerfile
# The FROM instruction sets the Base Image for subsequent instructions.
# As such, a valid Dockerfile must have FROM as its first instruction.
# The image can be any valid image.

FROM java:8

# The ADD instruction copies new files, directories or remote file URLs
# from <src> and adds them to the filesystem of the image at the path <dest>.

ADD target/interconnect17-menu-1.jar app.jar

# The RUN instruction will execute any commands in a new layer
# on top of the current image and commit the results.
# The resulting committed image will be used for the next step in the Dockerfile.
# Layering RUN instructions and generating commits conforms to the core concepts
# of Docker where commits are cheap and containers can be created from
# any point in an image's history, much like source control.

RUN bash -c 'touch /app.jar'

# The EXPOSE instruction informs Docker that the container listens on
# the specified network ports at runtime.
# EXPOSE does not make the ports of the container accessible to the host.
# To do that, you must use either the -p flag to publish a range of ports
# or the -P flag to publish all of the exposed ports.
# You can expose one port number and publish it externally under another number.

EXPOSE 8181

# An ENTRYPOINT allows you to configure a container
# that will run as an executable.
# ENTRYPOINT ["executable", "param1", "param2", ...]

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
interconnect2017@ubuntu:~/Workspace/interconnect-menu$
```

    1.2. Build Docker image by executing within the folder containing the Dockerfile

```
$ docker build -t <docker_image_name> .
```

Important: mind the dot at the end of the instruction. It indicates the path to the Dockerfile

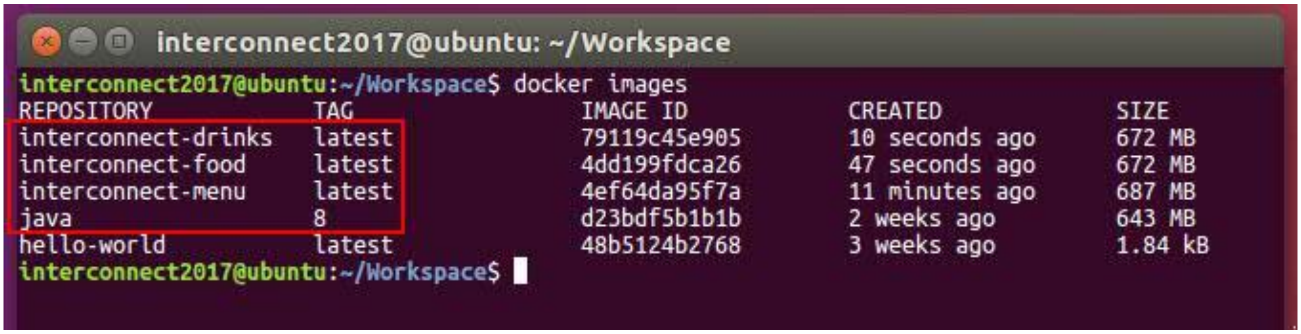


interconnect2017@ubuntu: ~/Workspace/interconnect-menu

```
interconnect2017@ubuntu:~/Workspace/interconnect-menu$ docker build -t interconnect-menu .
Sending build context to Docker daemon 21.81 MB
Step 1/5 : FROM java:8
 ---> d23bdf5b1b1b
Step 2/5 : ADD target/interconnect17-menu-1.jar app.jar
 ---> Using cache
 ---> 18fb25cd1cc6
Step 3/5 : RUN bash -c 'touch /app.jar'
 ---> Using cache
 ---> c13e8a1f129a
Step 4/5 : EXPOSE 8181
 ---> Using cache
 ---> bf0fe039d2ee
Step 5/5 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
 ---> Using cache
 ---> 4ef64da95f7a
Successfully built 4ef64da95f7a
interconnect2017@ubuntu:~/Workspace/interconnect-menu$
```

1.3. **Repeat previous steps for the other two Java project**
After building the three images, your local Docker image registry should look like

```
$ docker images
```

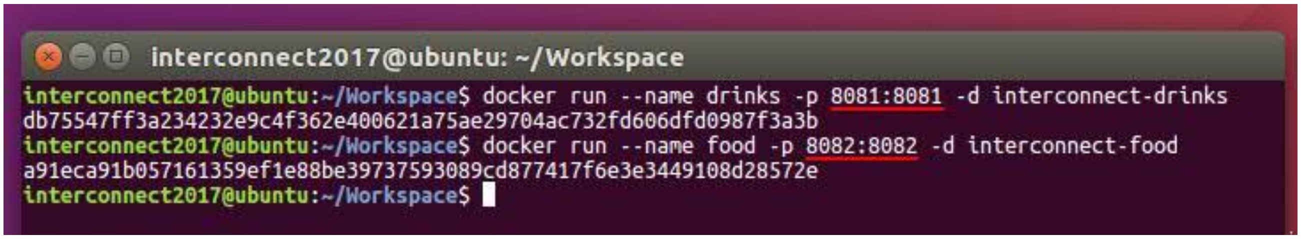


interconnect2017@ubuntu: ~/Workspace

```
interconnect2017@ubuntu:~/Workspace$ docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
interconnect-drinks  latest              79119c45e905        10 seconds ago      672 MB
interconnect-food    latest              4dd199fdca26        47 seconds ago      672 MB
interconnect-menu    latest              4ef64da95f7a        11 minutes ago      687 MB
java                 8                   d23bdf5b1b1b        2 weeks ago         643 MB
hello-world          latest              48b5124b2768        3 weeks ago         1.84 kB
interconnect2017@ubuntu:~/Workspace$
```

2. Run and test the dockerized/containerized 2017 IBM InterConnect menu Java app
   2.1. Run the following command to startup a container with the drinks app and another with the food app

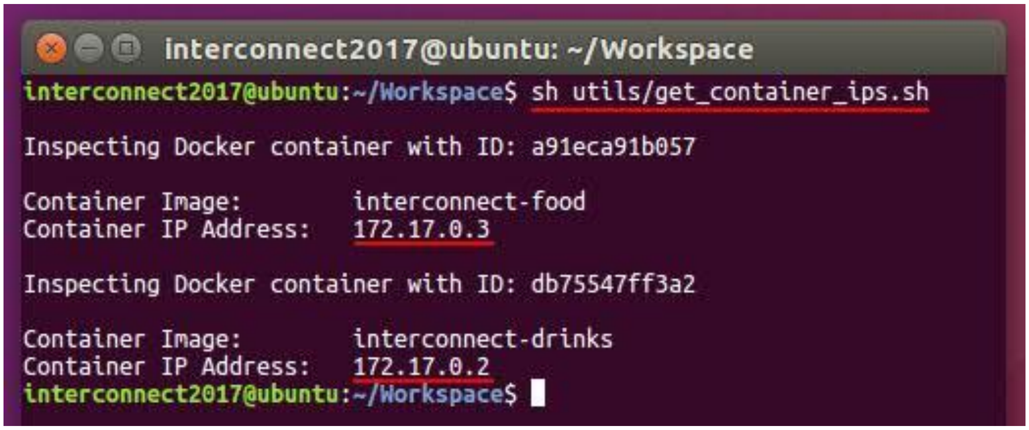
```
$ docker run --name <container_name> -p
<host_port>:<container_port> -d <docker_image>
```



interconnect2017@ubuntu: ~/Workspace

```
interconnect2017@ubuntu:~/Workspace$ docker run --name drinks -p 8081:8081 -d interconnect-drinks
db75547ff3a234232e9c4f362e400621a75ae29704ac732fd606dfd0987f3a3b
interconnect2017@ubuntu:~/Workspace$ docker run --name food -p 8082:8082 -d interconnect-food
a91eca91b057161359ef1e88be39737593089cd877417f6e3e3449108d28572e
interconnect2017@ubuntu:~/Workspace$
```

2.2. Get drinks and food containers IPs by executing the *get_container_ips.sh* located within the *utils* folder



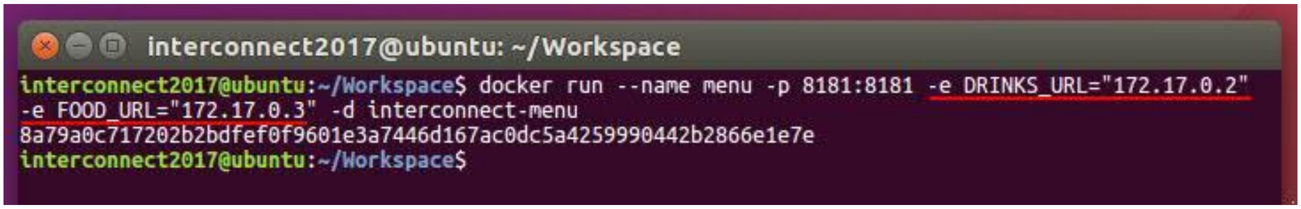
interconnect2017@ubuntu: ~/Workspace

```
interconnect2017@ubuntu:~/Workspace$ sh utils/get_container_ips.sh

Inspecting Docker container with ID: a91eca91b057

Container Image:        interconnect-food
Container IP Address:   172.17.0.3

Inspecting Docker container with ID: db75547ff3a2

Container Image:        interconnect-drinks
Container IP Address:   172.17.0.2
interconnect2017@ubuntu:~/Workspace$ 
```

2.3. Startup a container with the menu app by executing

$ docker run --name *<container_name>* -p *<host_port>*:*<container_port>* -e DRINKS_URL=<*drinks_container_IP*> -e FOOD_URL=<*food_container_IP*> -d *<docker_image>*

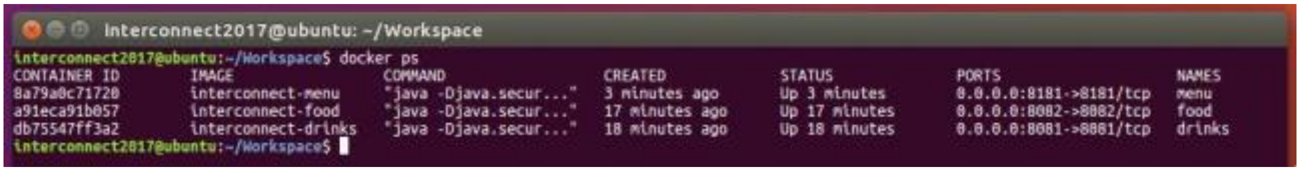
Important: since applications are now encapsulated in containers, localhost do not make reference to your workstation but to the container itself. Thus, the menu app needs to get the drinks and food container IP addresses as environment variables so that it can reach those apps and retrieve their data.



interconnect2017@ubuntu: ~/Workspace

```
interconnect2017@ubuntu:~/Workspace$ docker run --name menu -p 8181:8181 -e DRINKS_URL="172.17.0.2"
 -e FOOD_URL="172.17.0.3" -d interconnect-menu
8a79a0c717202b2bdfef0f9601e3a7446d167ac0dc5a4259990442b2866e1e7e
interconnect2017@ubuntu:~/Workspace$
```

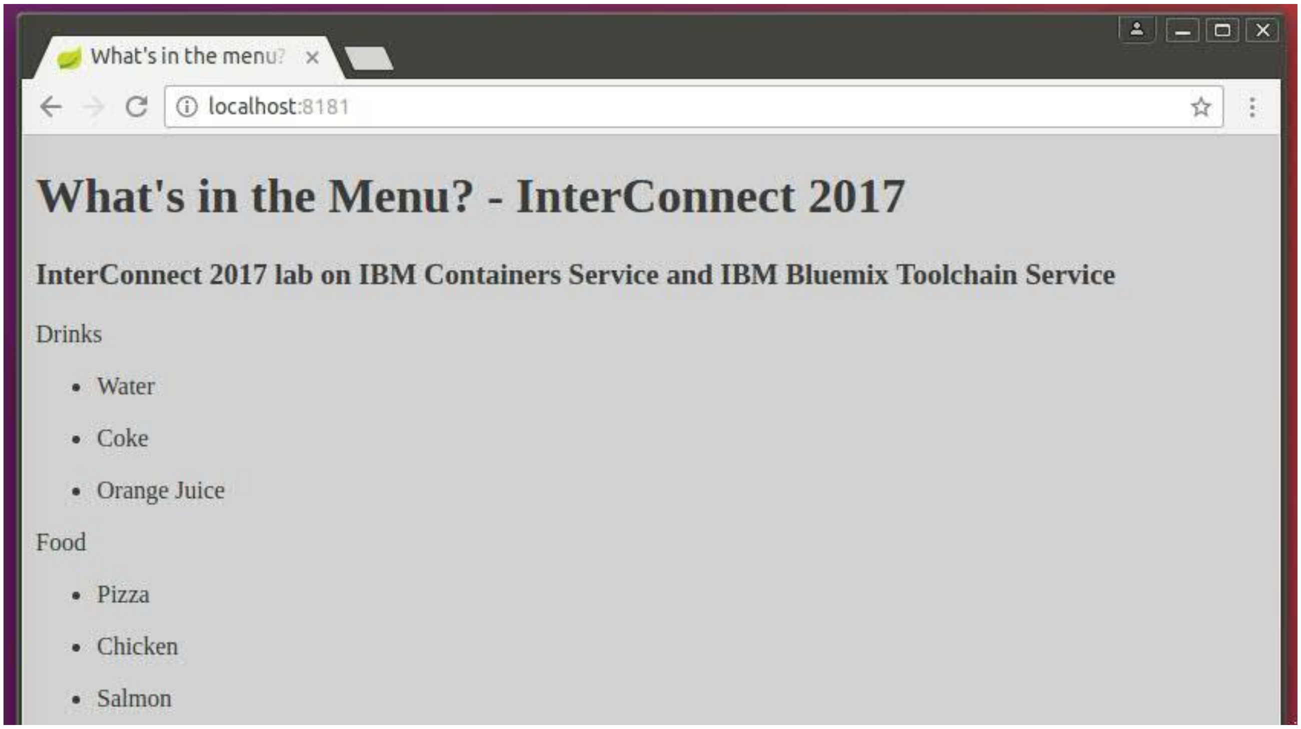
2.4. At this point, if you execute *docker ps* you should see the following



interconnect2017@ubuntu: ~/Workspace

```
interconnect2017@ubuntu:~/Workspace$ docker ps
CONTAINER ID        IMAGE                 COMMAND                  CREATED              STATUS              PORTS                      NAMES
8a79a0c71720        interconnect-menu     "java -Djava.secur..."   3 minutes ago        Up 3 minutes        0.0.0.0:8181->8181/tcp     menu
a91eca91b057        interconnect-food     "java -Djava.secur..."   17 minutes ago       Up 17 minutes       0.0.0.0:8082->8082/tcp     food
db75547ff3a2        interconnect-drinks   "java -Djava.secur..."   18 minutes ago       Up 18 minutes       0.0.0.0:8081->8081/tcp     drinks
interconnect2017@ubuntu:~/Workspace$ 
```

2.5. Check the dockerized/containerized 2017 IBM InterConnect menu application is working by pointing your browser to *localhost:8181*



What's in the Menu? - InterConnect 2017

InterConnect 2017 lab on IBM Containers Service and IBM Bluemix Toolchain Service

Drinks

- Water
- Coke
- Orange Juice

Food

- Pizza
- Chicken
- Salmon

IBM Bluemix Container Service

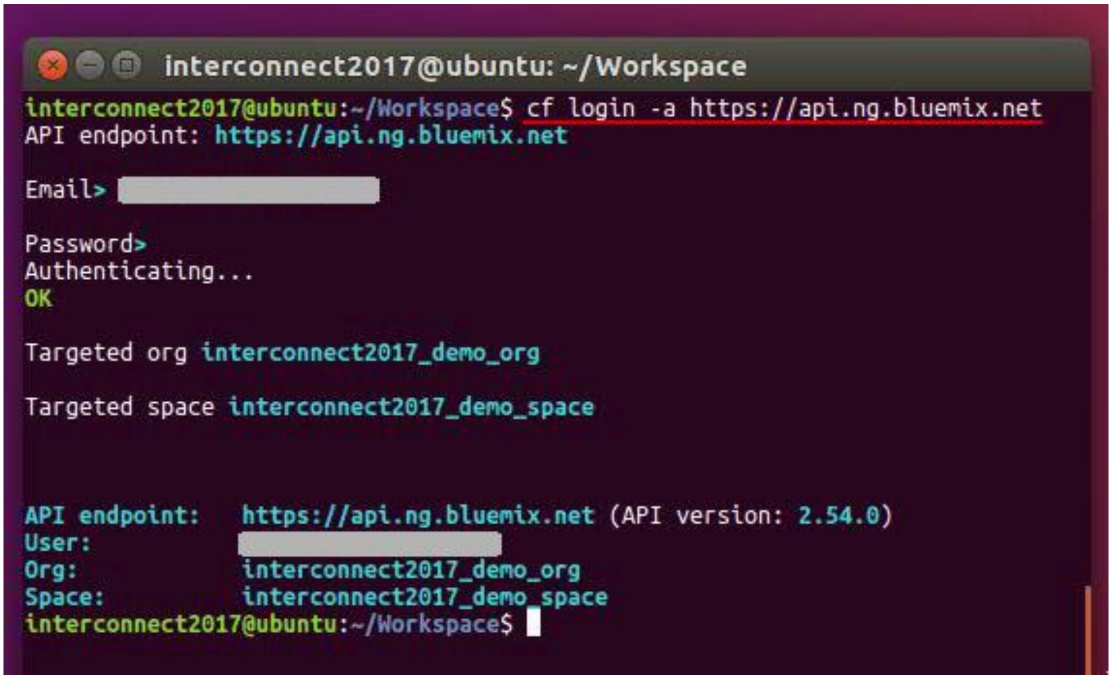
IBM Bluemix Container Service adapts Docker technology to enterprise application development, extending the benefits of resource allocation and isolation across public and private deployments, providing a private registry for your Docker images and the tools necessary to manage the entire life cycle of containerized applications you deploy in the cloud.

In this section, we will go through the same process of containerizing our three Java applications by using a Dockerfile but remotely in IBM Bluemix. We will now have our Docker images remotely in our private Docker Registry in IBM Bluemix. Finally, we will deploy those Docker images that make up the 2017 IBM InterConnect menu app remotely in IBM Bluemix public cloud by using the IBM Bluemix Container Service.

1. Open a terminal and log into IBM Bluemix by using your IBM Bluemix credentials (pre-requisite section) by executing

   ```
   $ cf login –a <API_endpoint>
   ```

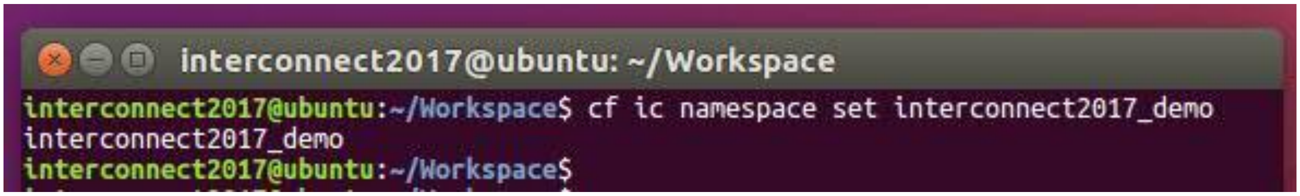
   Important: the API endpoint changes based upon the IBM Bluemix space region you want to work with. In this lab, we assume people will create a space in the US South region and work with it.



```
interconnect2017@ubuntu:~/Workspace$ cf login -a https://api.ng.bluemix.net
API endpoint: https://api.ng.bluemix.net

Email> [redacted]

Password>
Authenticating...
OK

Targeted org interconnect2017_demo_org

Targeted space interconnect2017_demo_space



API endpoint:   https://api.ng.bluemix.net (API version: 2.54.0)
User:           [redacted]
Org:            interconnect2017_demo_org
Space:          interconnect2017_demo_space
interconnect2017@ubuntu:~/Workspace$
```

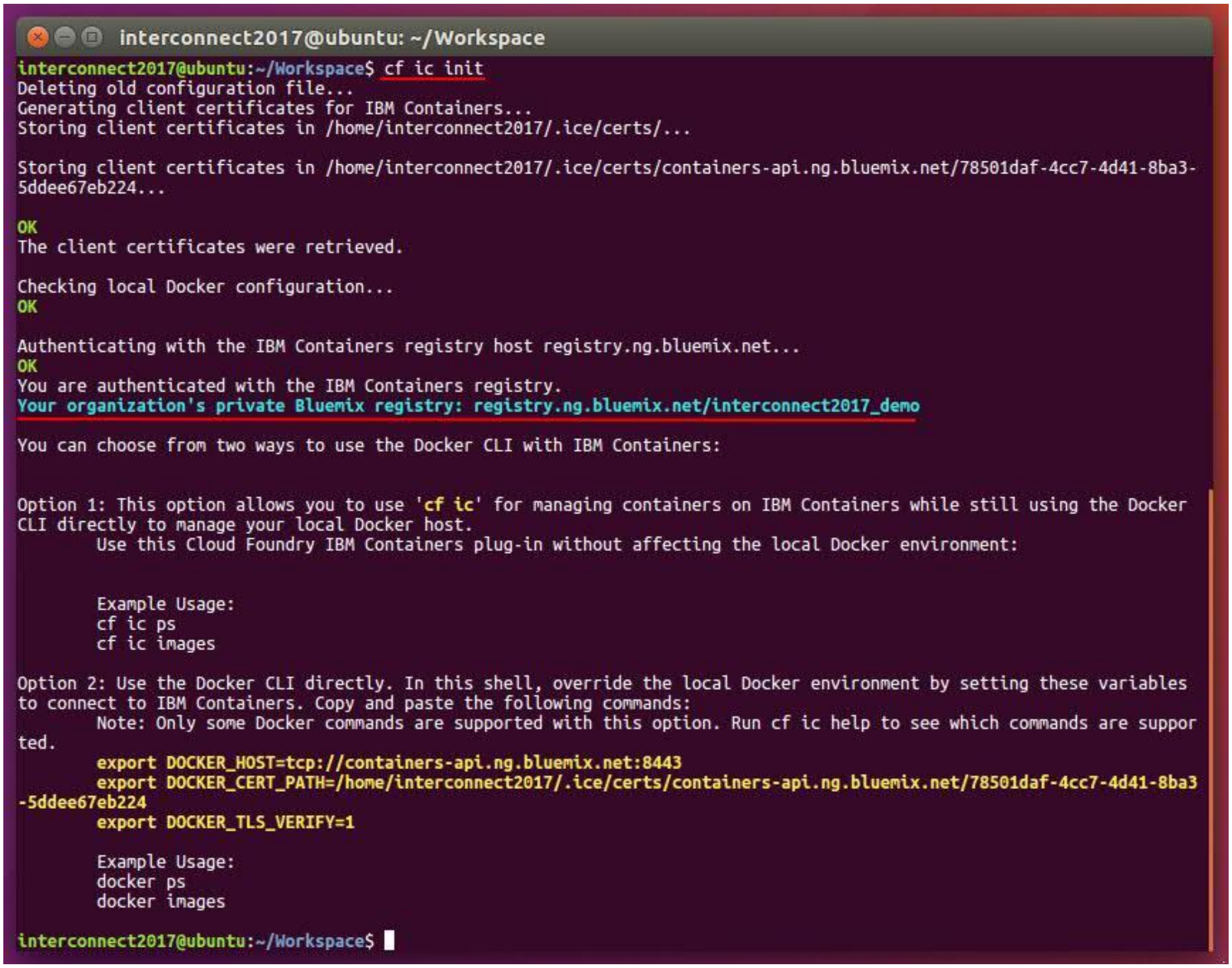
2. Create a namespace for your private Docker registry by executing

   ```
   $ cf ic namespace set <namespace_id>
   ```



```
interconnect2017@ubuntu:~/Workspace$ cf ic namespace set interconnect2017_demo
interconnect2017_demo
interconnect2017@ubuntu:~/Workspace$
```

3. Initialize the IBM Bluemix Container Service CLI by executing `cf ic init`



```
interconnect2017@ubuntu:~/Workspace$ cf ic init
Deleting old configuration file...
Generating client certificates for IBM Containers...
Storing client certificates in /home/interconnect2017/.ice/certs/...

Storing client certificates in /home/interconnect2017/.ice/certs/containers-api.ng.bluemix.net/78501daf-4cc7-4d41-8ba3-
5ddee67eb224...

OK
The client certificates were retrieved.

Checking local Docker configuration...
OK

Authenticating with the IBM Containers registry host registry.ng.bluemix.net...
OK
You are authenticated with the IBM Containers registry.
Your organization's private Bluemix registry: registry.ng.bluemix.net/interconnect2017_demo

You can choose from two ways to use the Docker CLI with IBM Containers:


Option 1: This option allows you to use 'cf ic' for managing containers on IBM Containers while still using the Docker
CLI directly to manage your local Docker host.
        Use this Cloud Foundry IBM Containers plug-in without affecting the local Docker environment:


        Example Usage:
        cf ic ps
        cf ic images

Option 2: Use the Docker CLI directly. In this shell, override the local Docker environment by setting these variables
to connect to IBM Containers. Copy and paste the following commands:
        Note: Only some Docker commands are supported with this option. Run cf ic help to see which commands are suppor
ted.
        export DOCKER_HOST=tcp://containers-api.ng.bluemix.net:8443
        export DOCKER_CERT_PATH=/home/interconnect2017/.ice/certs/containers-api.ng.bluemix.net/78501daf-4cc7-4d41-8ba3
-5ddee67eb224
        export DOCKER_TLS_VERIFY=1

        Example Usage:
        docker ps
        docker images

interconnect2017@ubuntu:~/Workspace$ 
```

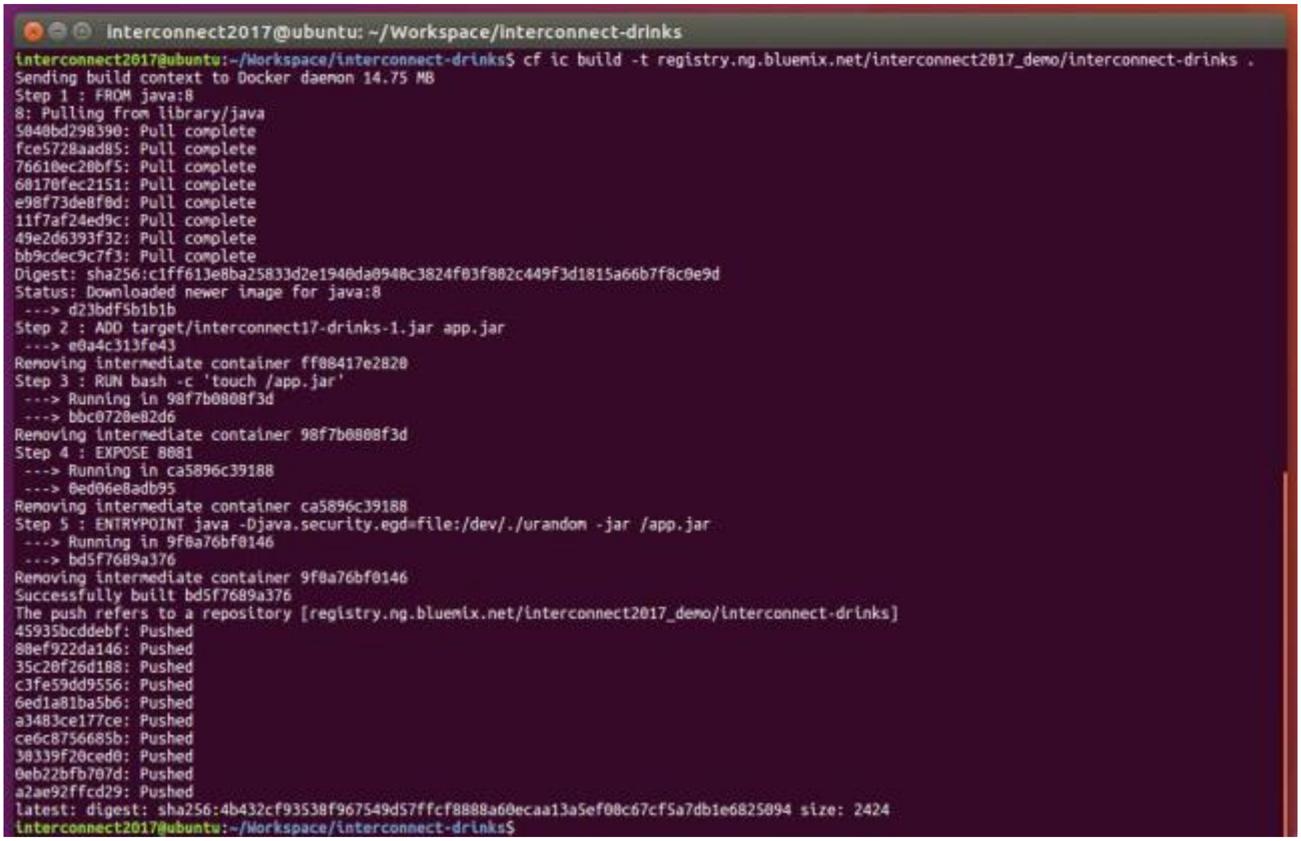
4. Build the three Docker images in your private Bluemix Docker registry by using the IBM Bluemix Container Service build service

    ```
    $ cf ic build -t registry.<Bluemix_region>.<Bluemix_domain>/<Docker_private_namespace>/<image_name> <path_to_Dockerfile>
    ```

    Hence, the above command for one of the apps of this lab would be:

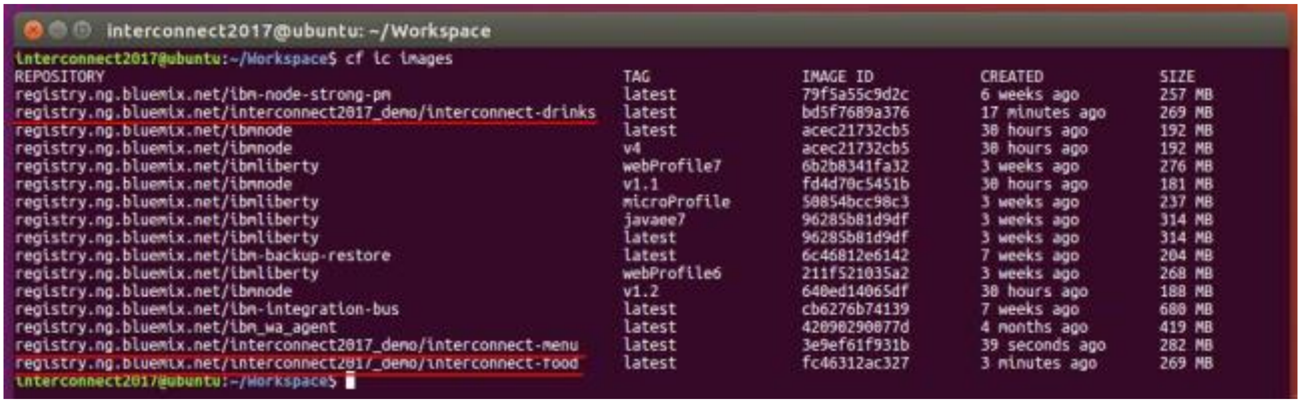
    *cf ic build -t registry.**ng**.bluemix.net/**interconnect2017_demo**/interconnect-drinks* **.**

After executing the build command, you should see something like this



```
interconnect2017@ubuntu:~/Workspace/interconnect-drinks$ cf ic build -t registry.ng.bluemix.net/interconnect2017_demo/interconnect-drinks .
Sending build context to Docker daemon 14.75 MB
Step 1 : FROM java:8
8: Pulling from library/java
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
Digest: sha256:c1ff613e8ba25833d2e1940da0940c3824f03f882c449f3d1815a66b7f8c0e9d
Status: Downloaded newer image for java:8
 ---> d23bdf5b1b1b
Step 2 : ADD target/interconnect17-drinks-1.jar app.jar
 ---> e0a4c313fe43
Removing intermediate container ff08417e2820
Step 3 : RUN bash -c 'touch /app.jar'
 ---> Running in 98f7b0808f3d
 ---> bbc0720e82d6
Removing intermediate container 98f7b0808f3d
Step 4 : EXPOSE 8081
 ---> Running in ca5896c39188
 ---> 0ed06e8adb95
Removing intermediate container ca5896c39188
Step 5 : ENTRYPOINT java -Djava.security.egd=file:/dev/./urandom -jar /app.jar
 ---> Running in 9f0a76bf0146
 ---> bd5f7689a376
Removing intermediate container 9f0a76bf0146
Successfully built bd5f7689a376
The push refers to a repository [registry.ng.bluemix.net/interconnect2017_demo/interconnect-drinks]
45935bcddebf: Pushed
80ef922da146: Pushed
35c20f26d188: Pushed
c3fe59dd9556: Pushed
6ed1a81ba5b6: Pushed
a3483ce177ce: Pushed
ce6c8756685b: Pushed
30339f20ced0: Pushed
0eb22bfb707d: Pushed
a2ae92ffcd29: Pushed
latest: digest: sha256:4b432cf93538f967549d57ffcf8888a60ecaa13a5ef00c67cf5a7db1e6825094 size: 2424
interconnect2017@ubuntu:~/Workspace/interconnect-drinks$
```

Once the three images have been built, execute the command `cf ic images` to see the Docker images in your private Docker registry in Bluemix. You should see the following



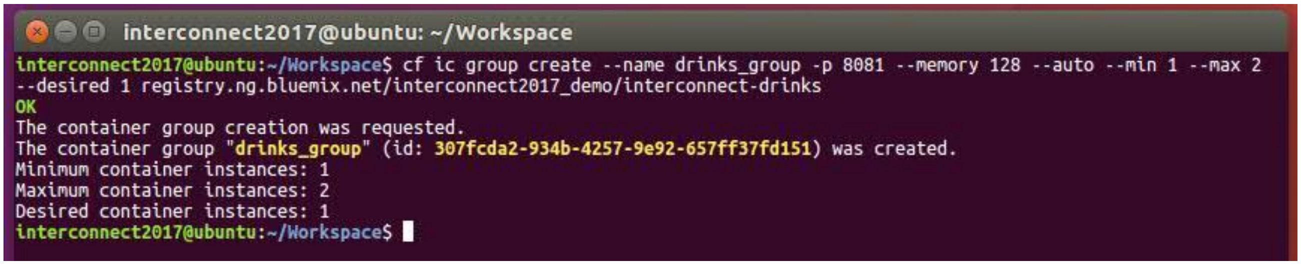
```
interconnect2017@ubuntu:~/Workspace$ cf ic images
REPOSITORY                                                          TAG                 IMAGE ID            CREATED             SIZE
registry.ng.bluemix.net/ibm-node-strong-pm                          latest              79f5a55c9d2c        6 weeks ago         257 MB
registry.ng.bluemix.net/interconnect2017_demo/interconnect-drinks   latest              bd5f7689a376        17 minutes ago      269 MB
registry.ng.bluemix.net/ibmnode                                     latest              acec21732cb5        30 hours ago        192 MB
registry.ng.bluemix.net/ibmnode                                     v4                  acec21732cb5        30 hours ago        192 MB
registry.ng.bluemix.net/ibmliberty                                  webProfile7         6b2b8341fa32        3 weeks ago         276 MB
registry.ng.bluemix.net/ibmnode                                     v1.1                fd4d70c5451b        30 hours ago        181 MB
registry.ng.bluemix.net/ibmliberty                                  microProfile        50854bcc98c3        3 weeks ago         237 MB
registry.ng.bluemix.net/ibmliberty                                  javaee7             96285b81d9df        3 weeks ago         314 MB
registry.ng.bluemix.net/ibmliberty                                  latest              96285b81d9df        3 weeks ago         314 MB
registry.ng.bluemix.net/ibm-backup-restore                          latest              6c46812e6142        7 weeks ago         204 MB
registry.ng.bluemix.net/ibmliberty                                  webProfile6         211f521035a2        3 weeks ago         268 MB
registry.ng.bluemix.net/ibmnode                                     v1.2                640ed14065df        30 hours ago        188 MB
registry.ng.bluemix.net/ibm-integration-bus                         latest              cb6276b74139        7 weeks ago         680 MB
registry.ng.bluemix.net/ibm_wa_agent                                latest              42090290077d        4 months ago        419 MB
registry.ng.bluemix.net/interconnect2017_demo/interconnect-menu     latest              3e9ef61f931b        39 seconds ago      282 MB
registry.ng.bluemix.net/interconnect2017_demo/interconnect-food     latest              fc46312ac327        3 minutes ago       269 MB
interconnect2017@ubuntu:~/Workspace$
```

5. Run the drinks and food Docker images in IBM Bluemix Container groups.

   FYI: A single container in the IBM Bluemix Container Service is similar to a container that you create in your local Docker environment. Single containers are a good way to start with IBM Bluemix Container Service and to learn about how containers work in the IBM cloud and the features that IBM Bluemix Container Service provides. You can also use single containers to run simple app tests or during the development process of an app. Instead of a single container, you can also use a container group, which is used in this task. A container group includes two or more containers that run the same image. Use container groups for running long-term services with workloads that require scalability and reliability or for testing at the required scale.

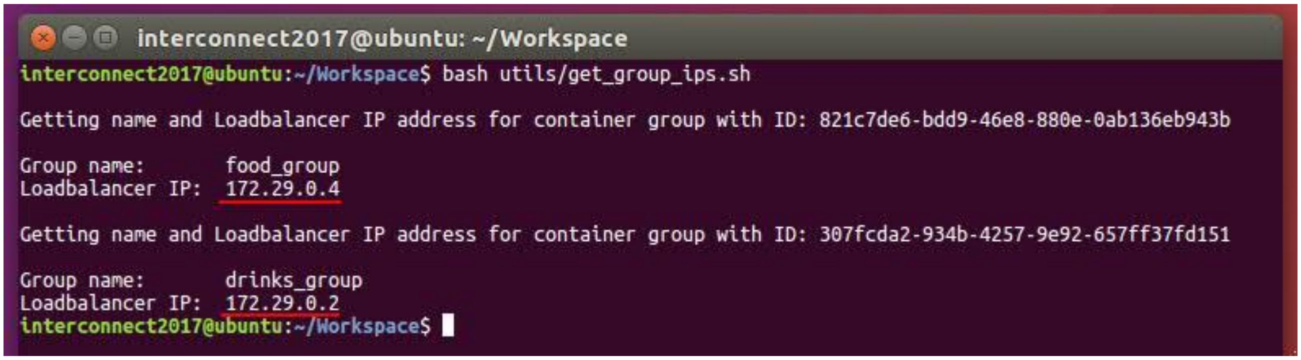
   ```
   $ cf ic group create --name <group_name> -p <port> --memory
   <amount_of_memory> --auto --min <min_number_instances> --max
   <max_number_instances> --desired <desired_number_instances>
   <docker_image_from_your_private_registry>
   ```

   You should see the following output



```
interconnect2017@ubuntu:~/Workspace$ cf ic group create --name drinks_group -p 8081 --memory 128 --auto --min 1 --max 2
--desired 1 registry.ng.bluemix.net/interconnect2017_demo/interconnect-drinks
OK
The container group creation was requested.
The container group "drinks_group" (id: 307fcda2-934b-4257-9e92-657ff37fd151) was created.
Minimum container instances: 1
Maximum container instances: 2
Desired container instances: 1
interconnect2017@ubuntu:~/Workspace$
```

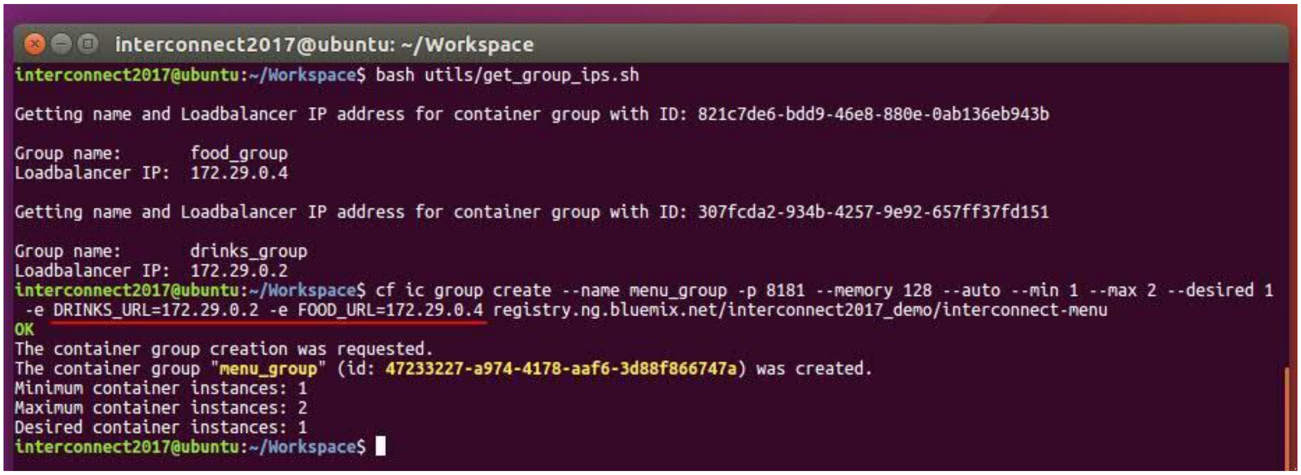
6. As we did when running drinks and app in containers locally, we need to get the IP addresses of those containers so that we can let them know to the menu app so that the menu app can reach the others. For doing so, execute `bash get_group_ips.sh` script within the *utils* folder



```
interconnect2017@ubuntu:~/Workspace$ bash utils/get_group_ips.sh

Getting name and Loadbalancer IP address for container group with ID: 821c7de6-bdd9-46e8-880e-0ab136eb943b

Group name:       food_group
Loadbalancer IP:  172.29.0.4

Getting name and Loadbalancer IP address for container group with ID: 307fcda2-934b-4257-9e92-657ff37fd151

Group name:       drinks_group
Loadbalancer IP:  172.29.0.2
interconnect2017@ubuntu:~/Workspace$
```

7. Run the menu Docker image in an IBM Bluemix Container group passing the other apps' IP addresses by running

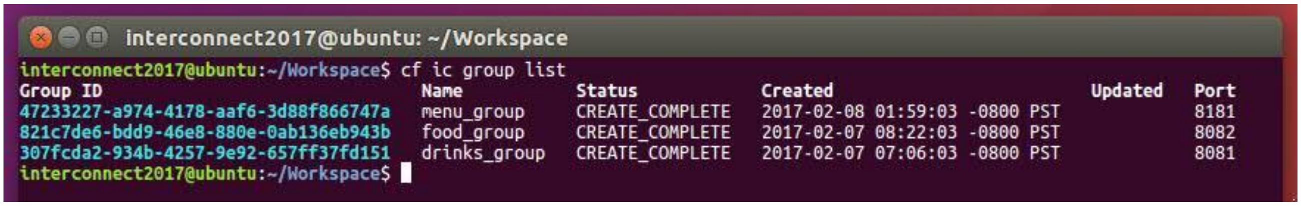
   ```
   $ cf ic group create --name <group_name> -p <port> --memory
   <amount_of_memory> --auto --min <min_number_instances> --max
   <max_number_instances> --desired <desired_number_instances> -e
   DRINKS_URL=<drinks_container_IP> -e FOOD_URL=<food_container_IP>
   <docker_image_from_your_private_registry>
   ```



```
interconnect2017@ubuntu:~/Workspace$ bash utils/get_group_ips.sh

Getting name and Loadbalancer IP address for container group with ID: 821c7de6-bdd9-46e8-880e-0ab136eb943b

Group name:       food_group
Loadbalancer IP:  172.29.0.4

Getting name and Loadbalancer IP address for container group with ID: 307fcda2-934b-4257-9e92-657ff37fd151

Group name:       drinks_group
Loadbalancer IP:  172.29.0.2
interconnect2017@ubuntu:~/Workspace$ cf ic group create --name menu_group -p 8181 --memory 128 --auto --min 1 --max 2 --desired 1
 -e DRINKS_URL=172.29.0.2 -e FOOD_URL=172.29.0.4 registry.ng.bluemix.net/interconnect2017_demo/interconnect-menu
OK
The container group creation was requested.
The container group "menu_group" (id: 47233227-a974-4178-aaf6-3d88f866747a) was created.
Minimum container instances: 1
Maximum container instances: 2
Desired container instances: 1
interconnect2017@ubuntu:~/Workspace$
```

8. Check the three container groups have been created by executing

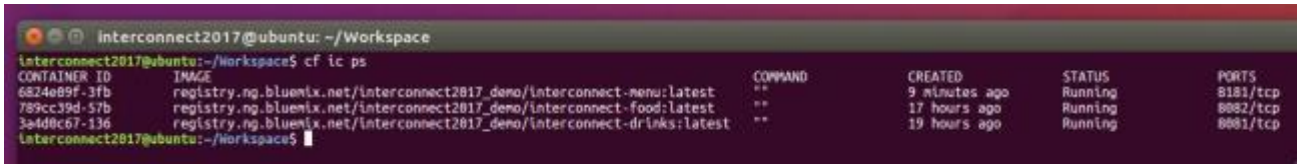
   ```
   $ cf ic group list
   ```



```
interconnect2017@ubuntu:~/Workspace$ cf ic group list
Group ID                               Name          Status            Created                          Updated    Port
47233227-a974-4178-aaf6-3d88f866747a   menu_group    CREATE_COMPLETE   2017-02-08 01:59:03 -0800 PST               8181
821c7de6-bdd9-46e8-880e-0ab136eb943b   food_group    CREATE_COMPLETE   2017-02-07 08:22:03 -0800 PST               8082
307fcda2-934b-4257-9e92-657ff37fd151   drinks_group  CREATE_COMPLETE   2017-02-07 07:06:03 -0800 PST               8081
interconnect2017@ubuntu:~/Workspace$
```

9. Check three containers have been created by executing

   ```
   $ cf ic ps
   ```

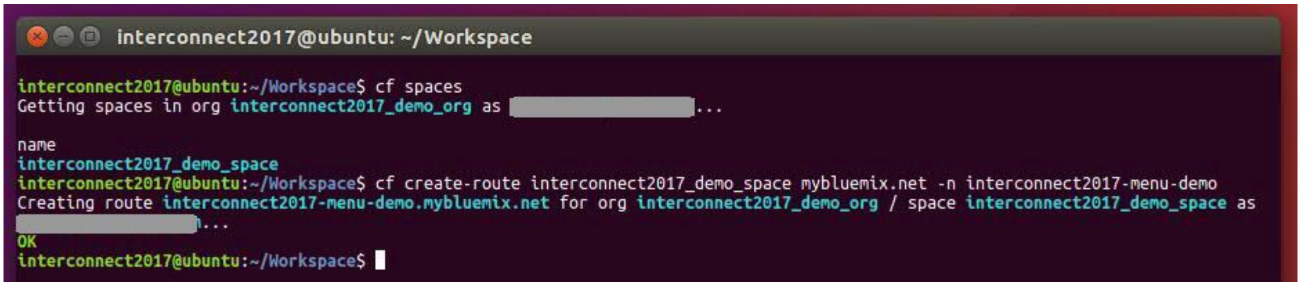


```
interconnect2017@ubuntu:~/Workspace$ cf ic ps
CONTAINER ID    IMAGE                                                                  COMMAND     CREATED          STATUS     PORTS
6824e89f-3fb    registry.ng.bluemix.net/interconnect2017_demo/interconnect-menu:latest     ""          9 minutes ago    Running    8181/tcp
789cc39d-57b    registry.ng.bluemix.net/interconnect2017_demo/interconnect-food:latest     ""          17 hours ago     Running    8082/tcp
3a4d0c67-136    registry.ng.bluemix.net/interconnect2017_demo/interconnect-drinks:latest   ""          19 hours ago     Running    8081/tcp
interconnect2017@ubuntu:~/Workspace$
```

10. Create a Public route to access the menu from outside of Bluemix

    ```
    $ cf create-route <Bluemix_space> <Bluemix_domain> -n <hostname>
    ```

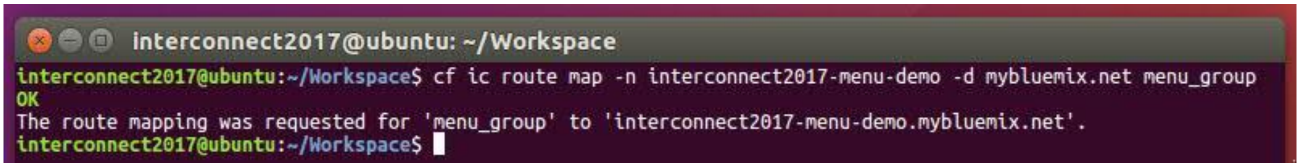
    Important: *hostname must be unique among all hostnames in the Bluemix public domain where your app is deployed. Hence, it is recommended to use some unique word in your hostname such as your surname.*



```
interconnect2017@ubuntu:~/Workspace$ cf spaces
Getting spaces in org interconnect2017_demo_org as [redacted]...

name
interconnect2017_demo_space
interconnect2017@ubuntu:~/Workspace$ cf create-route interconnect2017_demo_space mybluemix.net -n interconnect2017-menu-demo
Creating route interconnect2017-menu-demo.mybluemix.net for org interconnect2017_demo_org / space interconnect2017_demo_space as
[redacted]...
OK
interconnect2017@ubuntu:~/Workspace$ 
```

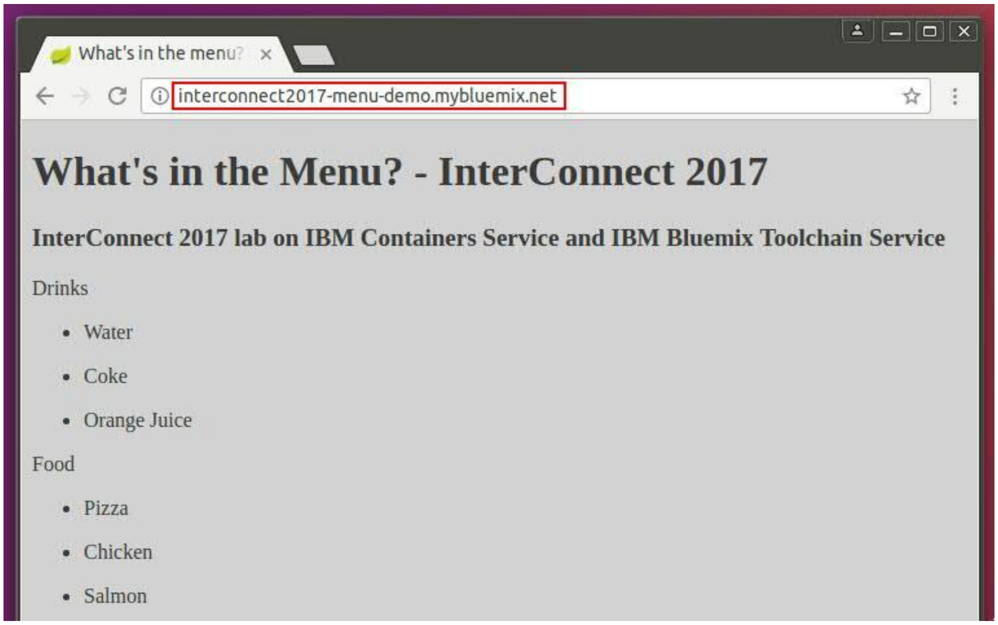
11. Map that public route to the menu container group create in step 7 above

    ```
    $ cf ic route map -n <hostname> -d <Bluemix_domain>
    <container_group_name>
    ```



```
interconnect2017@ubuntu:~/Workspace$ cf ic route map -n interconnect2017-menu-demo -d mybluemix.net menu_group
OK
The route mapping was requested for 'menu_group' to 'interconnect2017-menu-demo.mybluemix.net'.
interconnect2017@ubuntu:~/Workspace$ 
```

12. Check the dockerized/containerized 2017 IBM InterConnect menu application on the Bluemix public cloud is working by pointing your browser to the mapped route specified above



What's in the Menu? - InterConnect 2017

InterConnect 2017 lab on IBM Containers Service and IBM Bluemix Toolchain Service

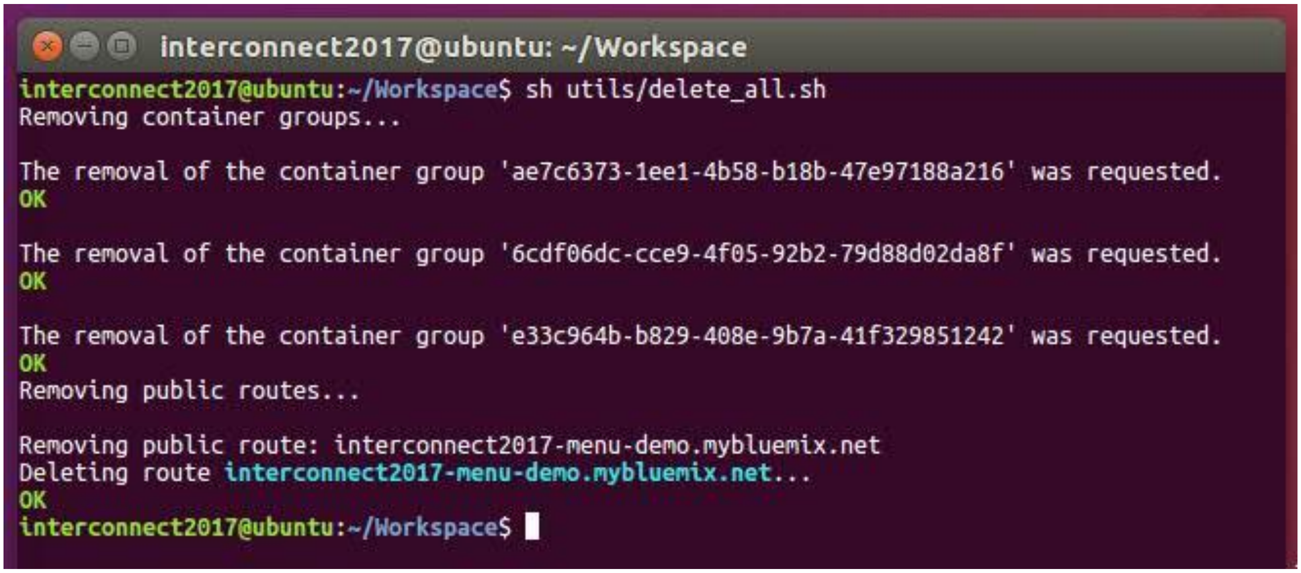
Drinks

- Water
- Coke
- Orange Juice

Food

- Pizza
- Chicken
- Salmon

13. Delete all created container groups and public routes in preparation for next sections of this lab by executing the `delete_all.sh` script in the *utils* folder



```
interconnect2017@ubuntu:~/Workspace$ sh utils/delete_all.sh
Removing container groups...

The removal of the container group 'ae7c6373-1ee1-4b58-b18b-47e97188a216' was requested.
OK

The removal of the container group '6cdf06dc-cce9-4f05-92b2-79d88d02da8f' was requested.
OK

The removal of the container group 'e33c964b-b829-408e-9b7a-41f329851242' was requested.
OK
Removing public routes...

Removing public route: interconnect2017-menu-demo.mybluemix.net
Deleting route interconnect2017-menu-demo.mybluemix.net...
OK
interconnect2017@ubuntu:~/Workspace$
```

IBM Bluemix Continuous Delivery Service

Use Continuous Delivery to automate builds, unit tests, deployments, and more. Edit and push code through the rich web based IDE. Create toolchains to enable tool integrations that support your development, deployment, and operation tasks.

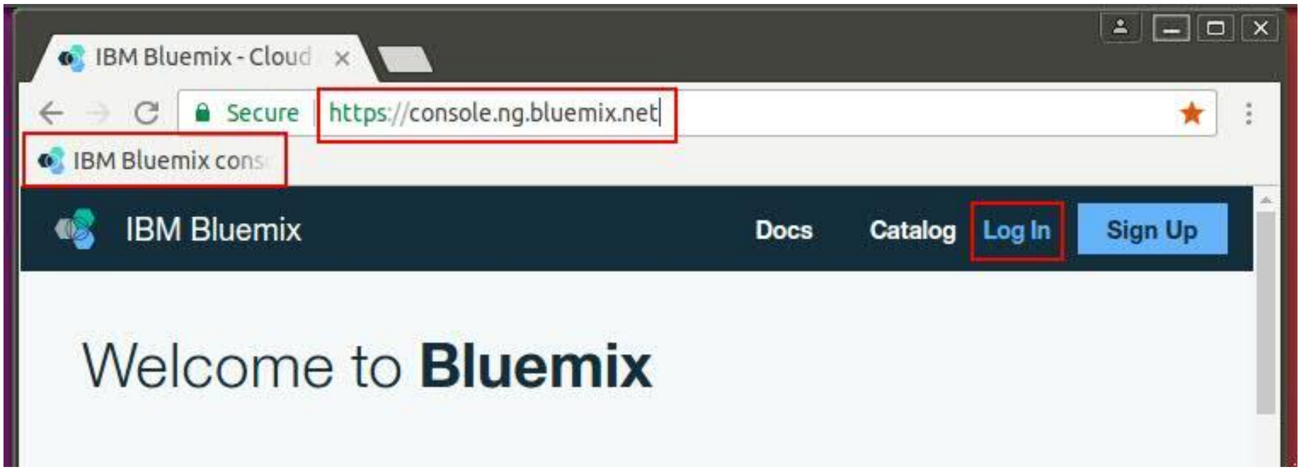
A toolchain is a set of tool integrations that support development, deployment, and operations tasks. The collective power of a toolchain is greater than the sum of its individual tool integrations.

Tools

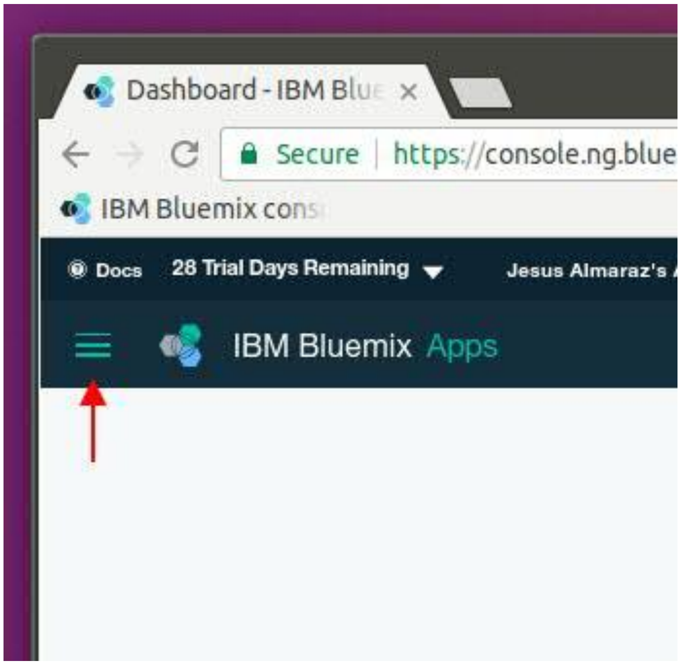
- GitHub – GitHub makes it easy to manage source code and revision history, and to track bugs, feature requests and tasks in hosted Git repositories.
- Delivery Pipeline – Continuously build and deploy. The Delivery Pipeline provides automated, continuous delivery to the IBM Bluemix cloud.

In this section, we will create a toolchain using the IBM Bluemix Continuous Delivery Service to automate the build and deployment to the IBM Bluemix public cloud of the 2017 InterConnect menu app. We will go through the creation and configuration of the GitHub and Delivery Pipeline tools for each of the three Java apps within our toolchain.

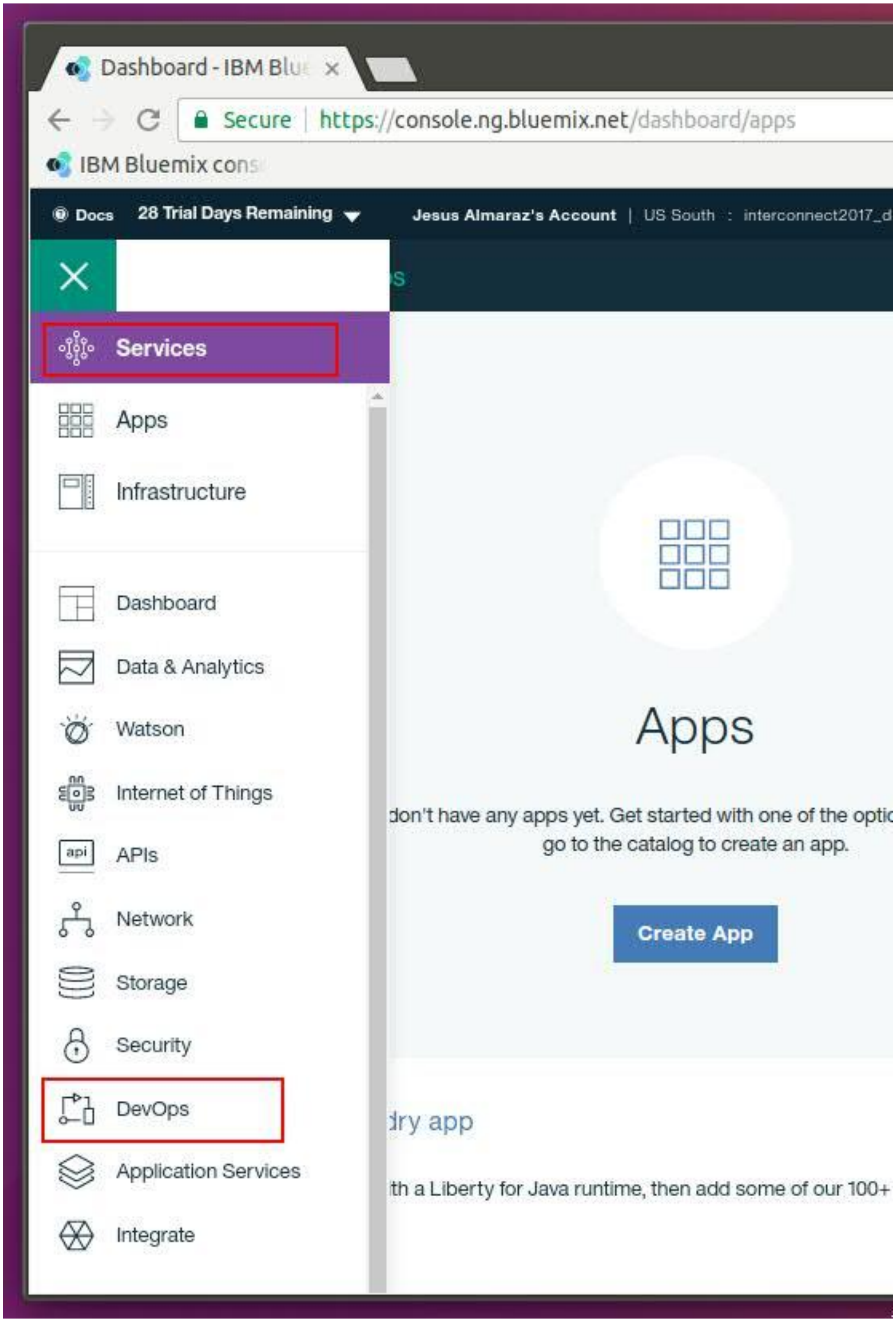
1. Create a toolchain from the IBM Bluemix Continuous Delivery Service
    1.1. Open your Bluemix console in your web browser



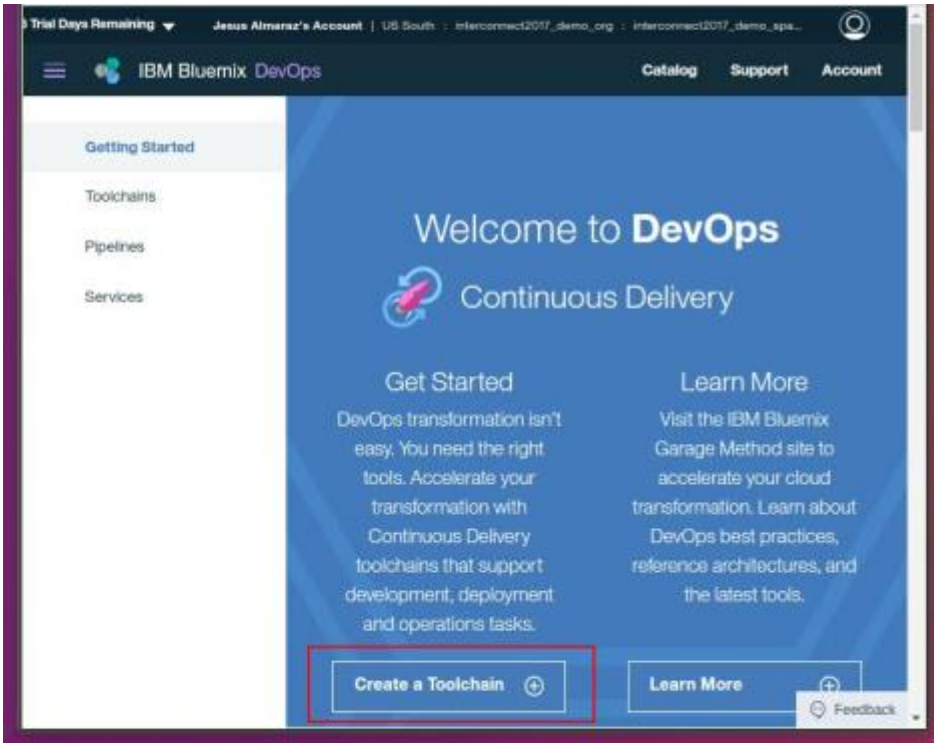
    1.2. Open the left hand side menu



1.3. Click on *Services* and then on the *DevOps* subsection of it


Dashboard - IBM Blu
Secure
https://console.ng.bluemix.net/dashboard/apps
IBM Bluemix cons
Docs
28 Trial Days Remaining
Jesus Almaraz's Account | US South : interconnect2017_d
Services
Apps
Infrastructure
Dashboard
Data & Analytics
Watson
Internet of Things
APIs
Network
Storage
Security
DevOps
Application Services
Integrate
Apps
don't have any apps yet. Get started with one of the optic
go to the catalog to create an app.
Create App
dry app
th a Liberty for Java runtime, then add some of our 100+

1.4. Click on *Create a Toolchain*



Welcome to DevOps

Continuous Delivery

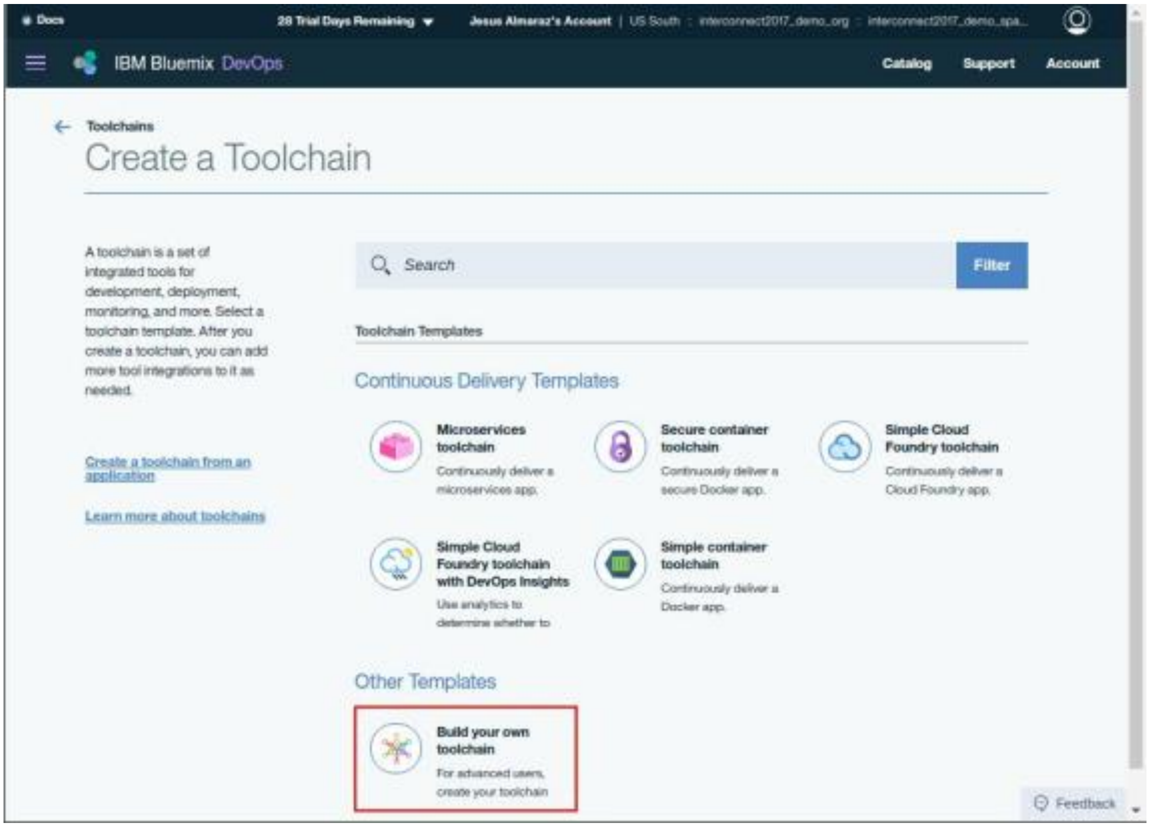
Get Started

DevOps transformation isn't easy. You need the right tools. Accelerate your transformation with Continuous Delivery toolchains that support development, deployment and operations tasks.

Learn More

Visit the IBM Bluemix Garage Method site to accelerate your cloud transformation. Learn about DevOps best practices, reference architectures, and the latest tools.

1.5. Click on *Build your own toolchain*


Docs
28 Trial Days Remaining
Jesus Almaraz's Account | US South : interconnect2017_demo_org : interconnect2017_demo_spa...
IBM Bluemix DevOps
Catalog
Support
Account
Toolchains
Search
Filter
Create a toolchain from an application
Learn more about toolchains
Feedback

Create a Toolchain

A toolchain is a set of integrated tools for development, deployment, monitoring, and more. Select a toolchain template. After you create a toolchain, you can add more tool integrations to it as needed.

Toolchain Templates

Continuous Delivery Templates

Microservices toolchain
Continuously deliver a microservices app.

Secure container toolchain
Continuously deliver a secure Docker app.

Simple Cloud Foundry toolchain
Continuously deliver a Cloud Foundry app.

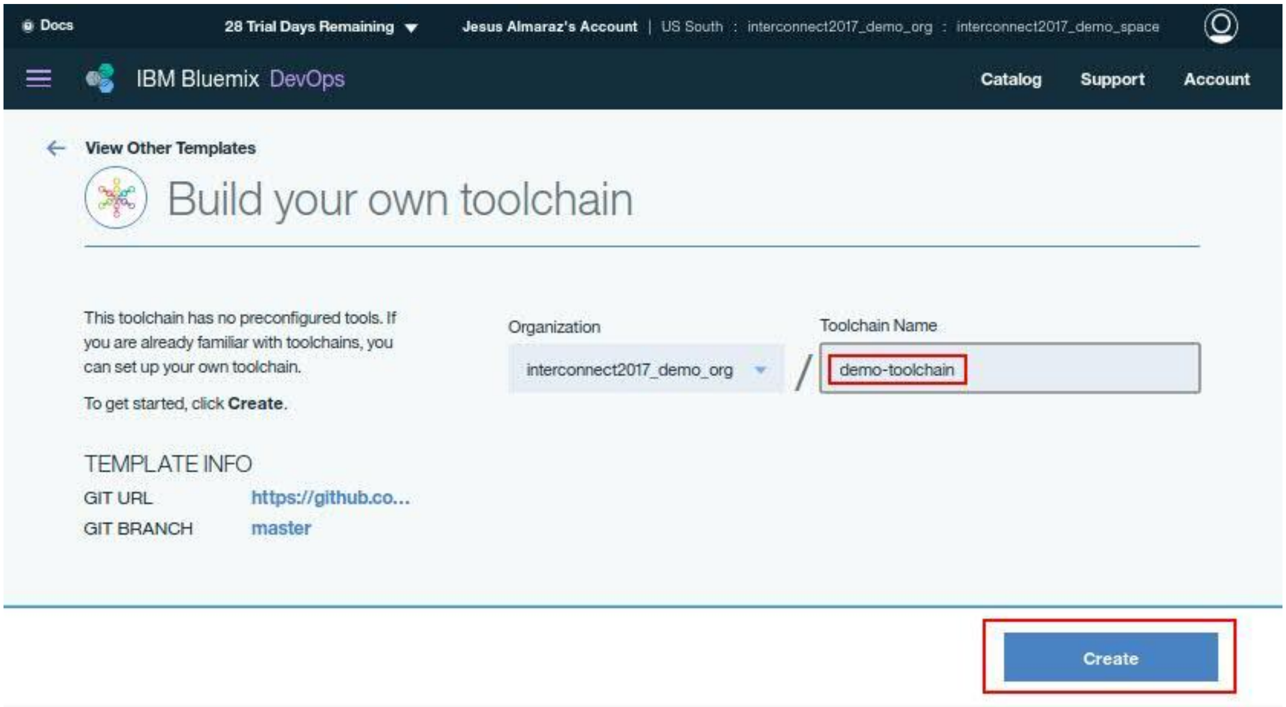
Simple Cloud Foundry toolchain with DevOps Insights
Use analytics to determine whether to

Simple container toolchain
Continuously deliver a Docker app.

Other Templates

Build your own toolchain
For advanced users, create your toolchain

1.6. Make sure you select your organization, give an appropriate name to your toolchain and click on *Create*



This toolchain has no preconfigured tools. If you are already familiar with toolchains, you can set up your own toolchain.

To get started, click **Create**.

TEMPLATE INFO

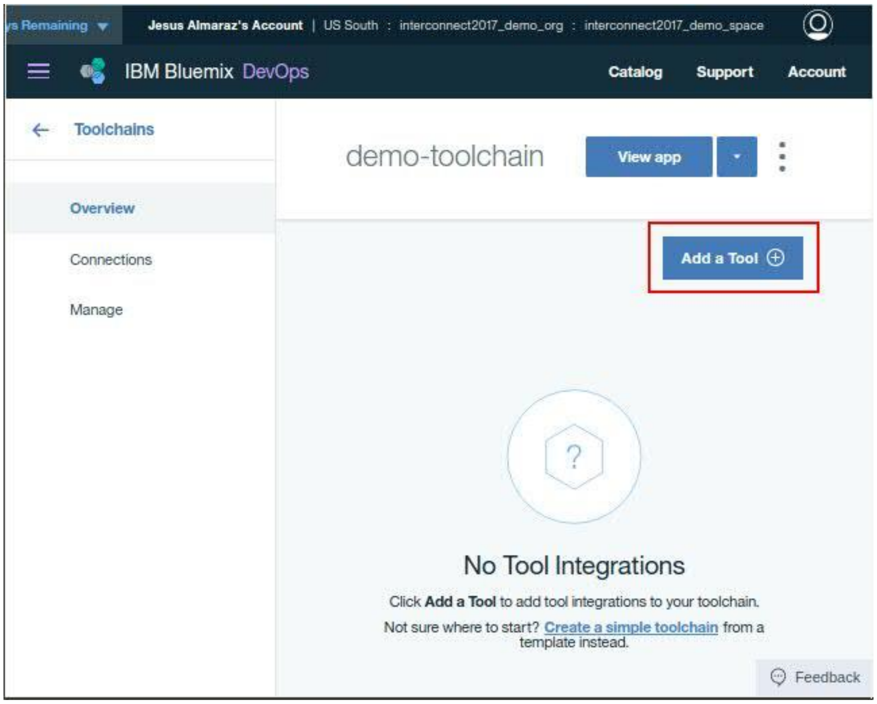
GIT URL	https://github.co...
GIT BRANCH	master

Organization: interconnect2017_demo_org

Toolchain Name: demo-toolchain

Now that we have our toolchain, we need to create a GitHub and Delivery pipeline for each of the three microservices and configure them appropriately.

2. Create a GitHub tool for each of the three microservices
   2.1. Click on *Add a Tool*

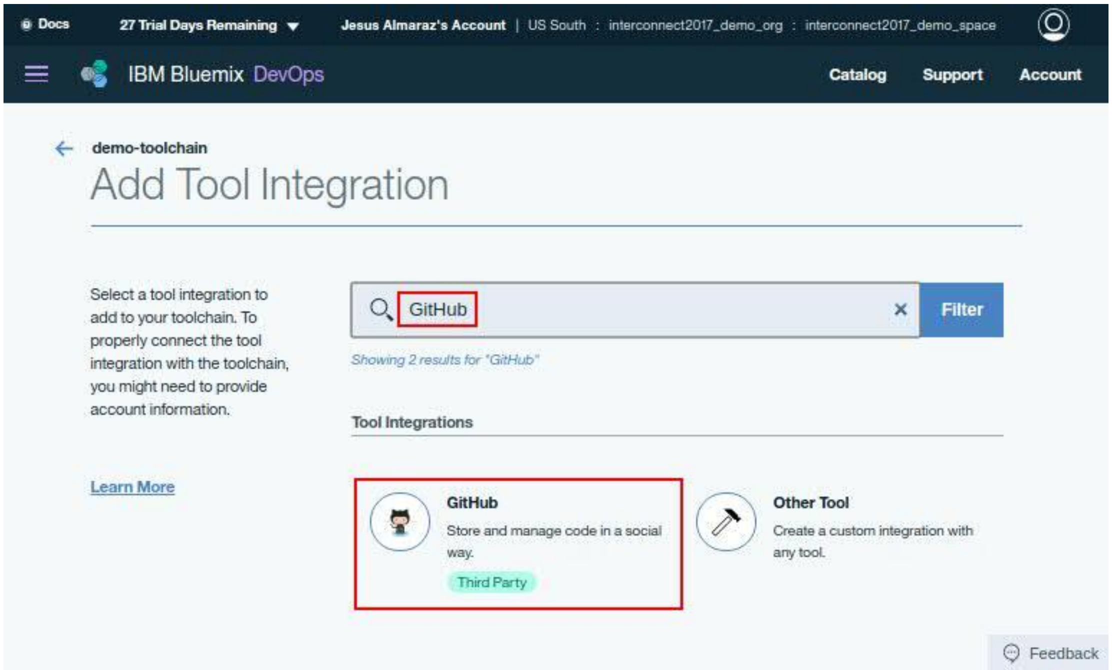


No Tool Integrations

Click **Add a Tool** to add tool integrations to your toolchain.

Not sure where to start? Create a simple toolchain from a template instead.

2.2. Search for GitHub and click on it



Docs 27 Trial Days Remaining ▼ Jesus Almaraz's Account | US South : interconnect2017_demo_org : interconnect2017_demo_space

IBM Bluemix DevOps Catalog Support Account

← demo-toolchain

Add Tool Integration

Select a tool integration to add to your toolchain. To properly connect the tool integration with the toolchain, you might need to provide account information.

Learn More

GitHub × Filter

Showing 2 results for "GitHub"

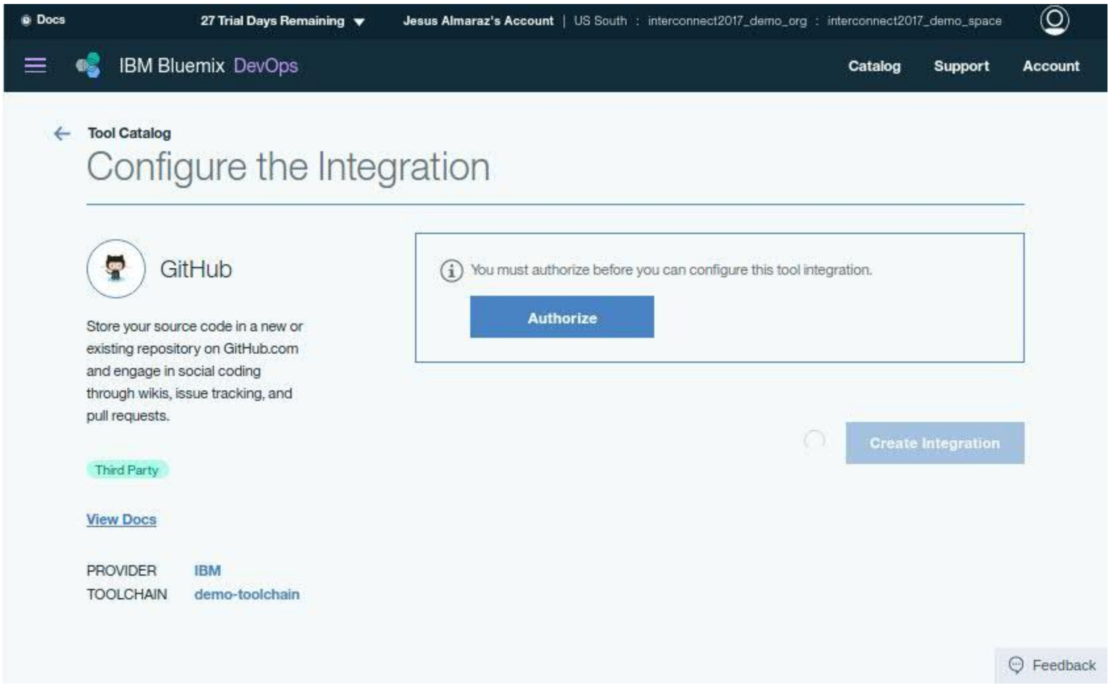
Tool Integrations

GitHub Store and manage code in a social way. Third Party

Other Tool Create a custom integration with any tool.

Feedback

2.3. You need to authorize your toolchain's access to your GitHub account



Docs 27 Trial Days Remaining ▼ Jesus Almaraz's Account | US South : interconnect2017_demo_org : interconnect2017_demo_space

IBM Bluemix DevOps Catalog Support Account

← Tool Catalog

Configure the Integration

GitHub

Store your source code in a new or existing repository on GitHub.com and engage in social coding through wikis, issue tracking, and pull requests.

Third Party

View Docs

PROVIDER IBM

TOOLCHAIN demo-toolchain

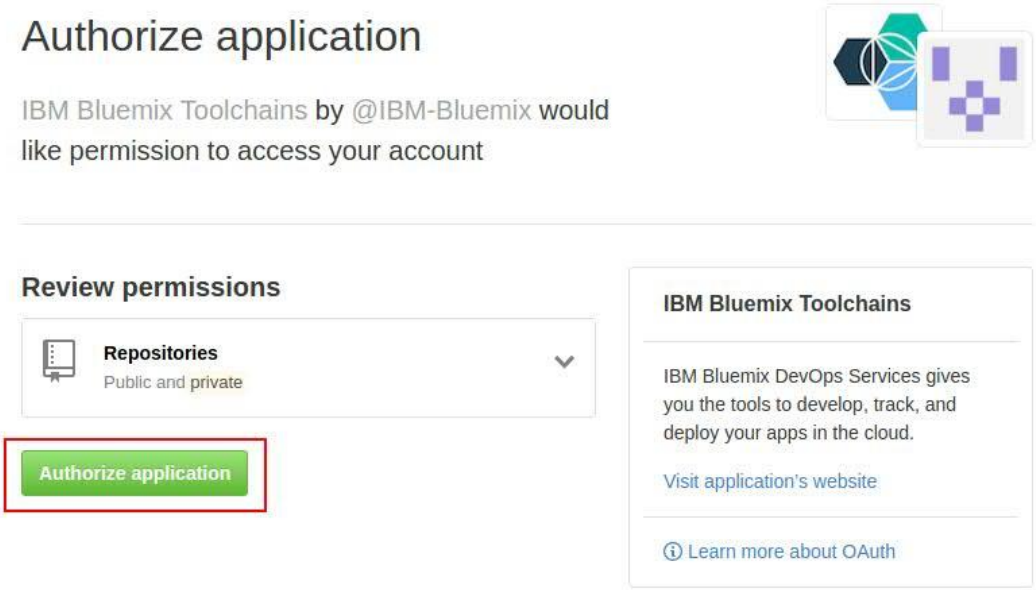
You must authorize before you can configure this tool integration.

Authorize

Create Integration

Feedback

2.4. Enter your GitHub credentials and click on *Authorize application*



Authorize application

IBM Bluemix Toolchains by @IBM-Bluemix would like permission to access your account

Review permissions

Repositories
Public and private

Authorize application

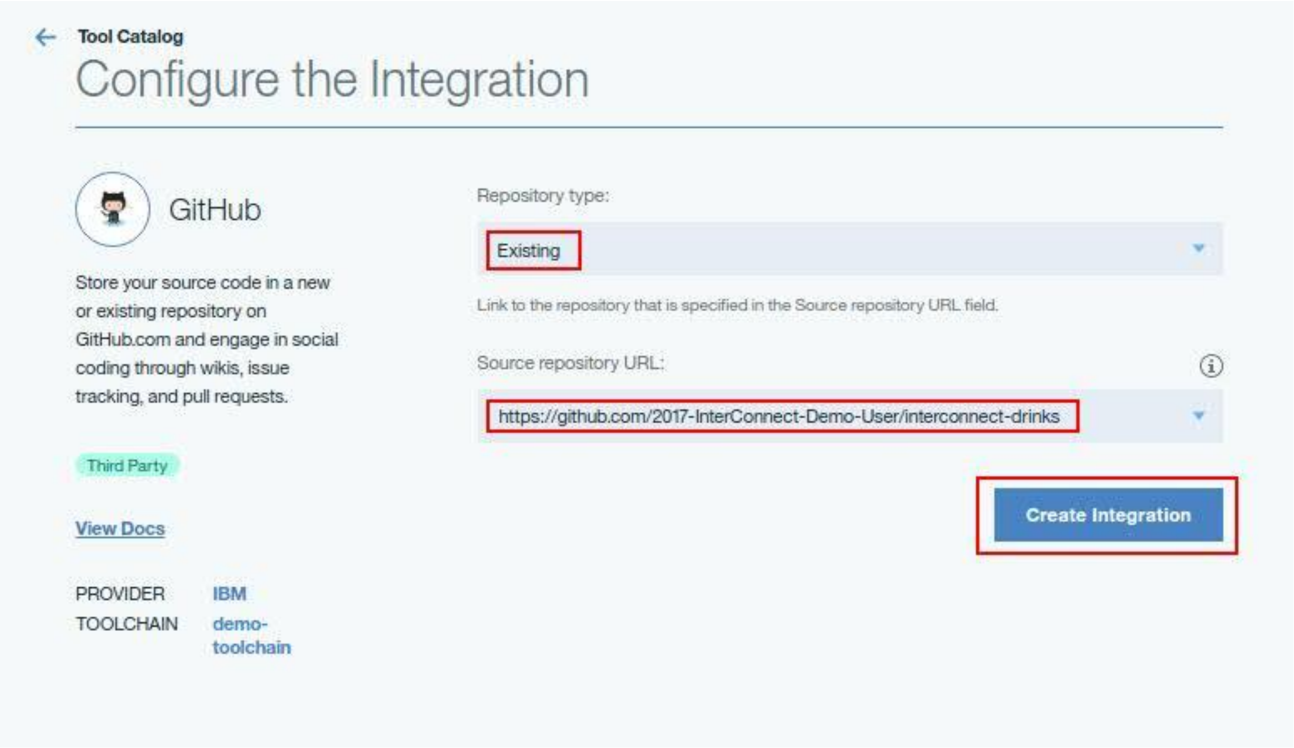
IBM Bluemix Toolchains

IBM Bluemix DevOps Services gives you the tools to develop, track, and deploy your apps in the cloud.

Visit application's website

ⓘ Learn more about OAuth

2.5. Select *Existing* for *Repository Type* and select one of the three Java app Github repositories. Then, click on *Create Integration*



← Tool Catalog

Configure the Integration

GitHub

Store your source code in a new or existing repository on GitHub.com and engage in social coding through wikis, issue tracking, and pull requests.

Third Party

View Docs

PROVIDER	IBM
TOOLCHAIN	demo-toolchain

Repository type:

Existing

Link to the repository that is specified in the Source repository URL field.

Source repository URL:

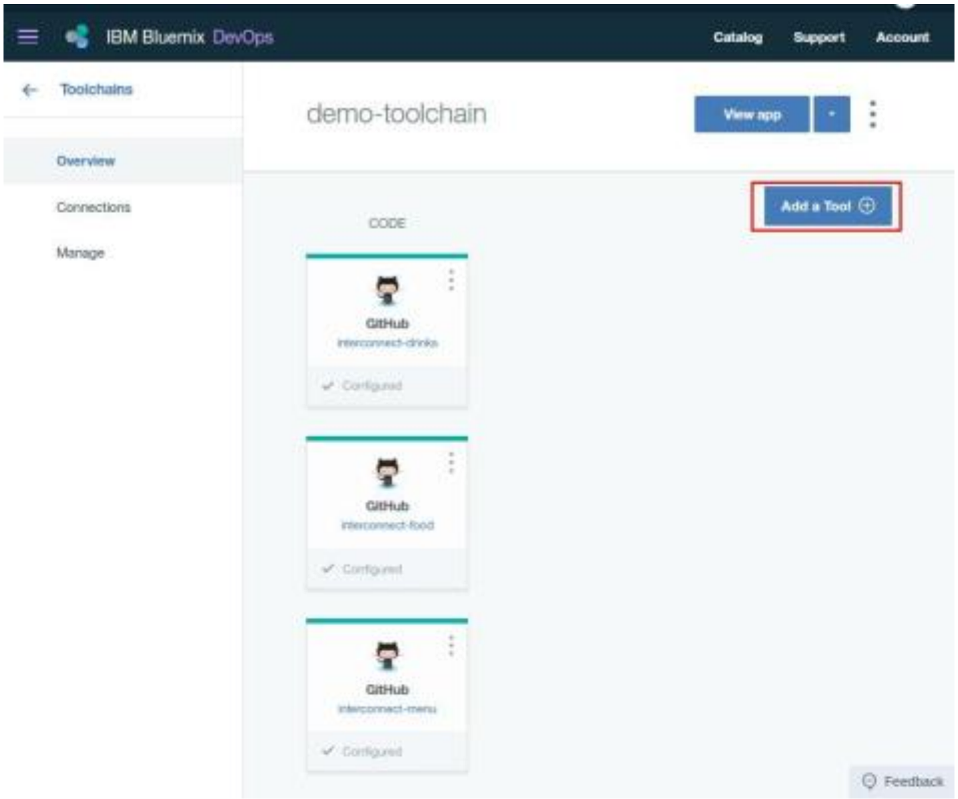
https://github.com/2017-InterConnect-Demo-User/interconnect-drinks

Create Integration

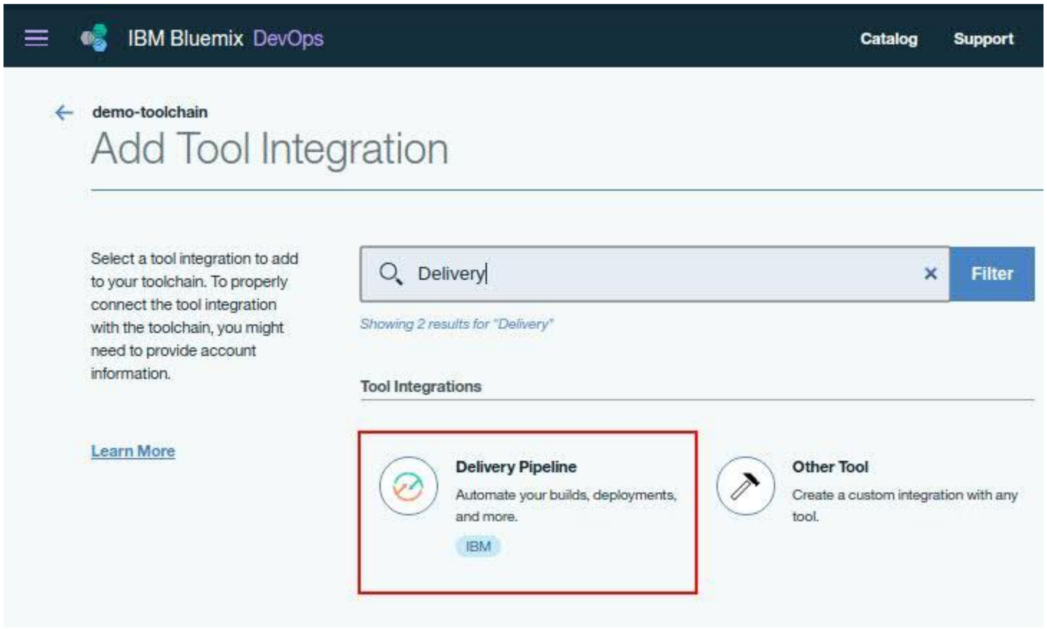
2.6. Repeat previous steps to create a GitHub tool for each of the three Java apps GitHub repositories

3. Create a Delivery Pipeline tool for each of the three Java apps

3.1. Click on *Add a Tool*



3.2. Search for Delivery Pipeline and click on it



Select a tool integration to add to your toolchain. To properly connect the tool integration with the toolchain, you might need to provide account information.

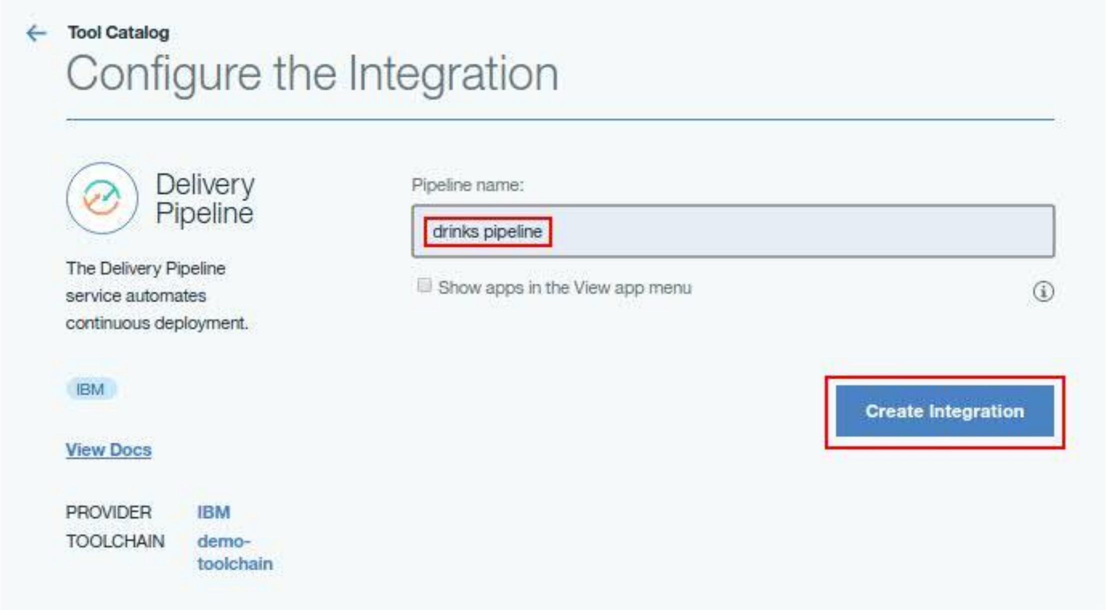
Delivery

Showing 2 results for "Delivery"

Delivery Pipeline: Automate your builds, deployments, and more.

Other Tool: Create a custom integration with any tool.

3.3. Give an appropriate name to your delivery pipeline and click on *Create Integration*



Configure the Integration

Delivery Pipeline

The Delivery Pipeline service automates continuous deployment.

PROVIDER IBM

TOOLCHAIN demo-toolchain

Pipeline name: drinks pipeline

□ Show apps in the View app menu

3.4. Repeat previous steps to create a Delivery Pipeline for each of the three Java apps

Now, we need to configure each of the three Delivery Pipelines in order to get them to do the build and deployment of each of the Java apps. This will consist of three stages:

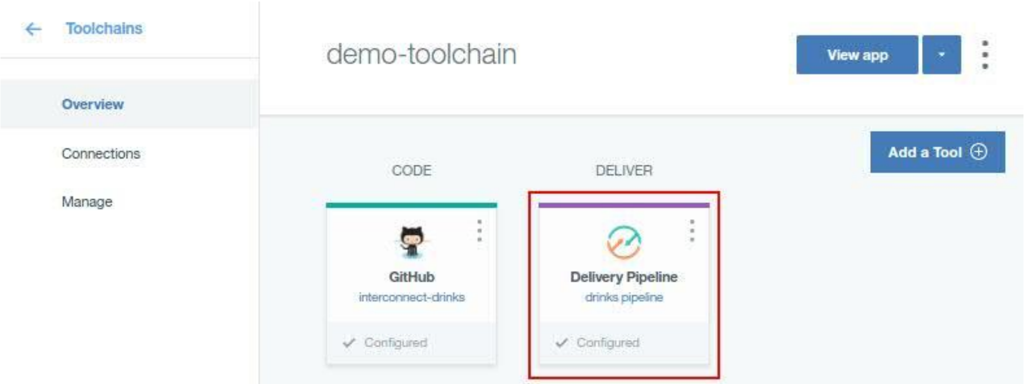
I. Build – where we will build the needed artefacts of the Java app from the code stored in its GitHub repository.

II. Create Docker image – where we will create a Docker image containing the built artefacts from step I.

III. Deploy – where we will deploy the Docker image created in step II onto the IBM Bluemix public cloud.

4. Configure the Delivery Pipelines

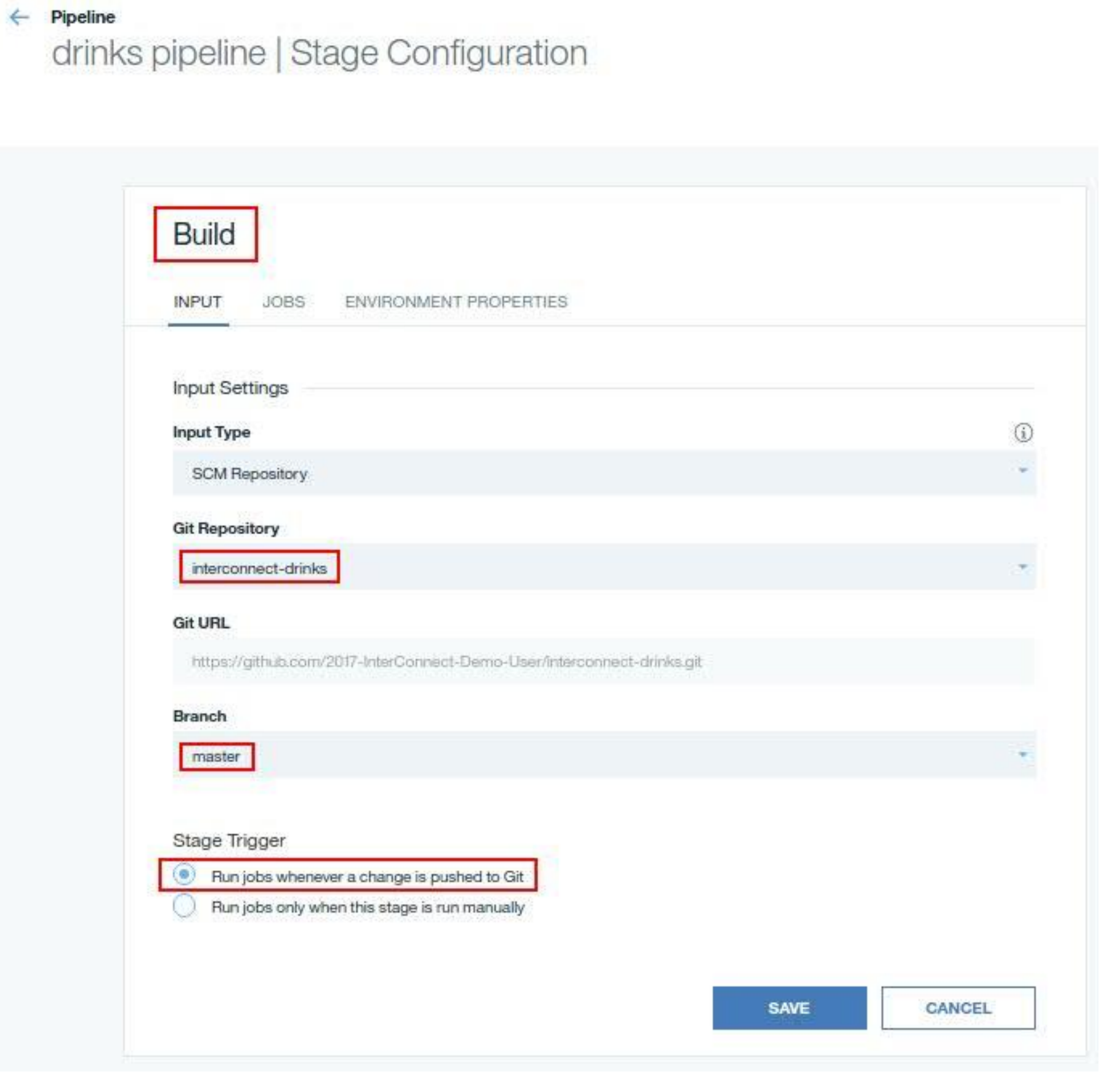
   4.1. Click on the Delivery Pipeline tool display on your toolchain dashboard



4.2. Create *Build* stage

4.2.1. Click on *Add Stage*

4.2.2. Name your stage as Build and make sure it is configured to use the appropriate Git Repository and Branch


← Pipeline

drinks pipeline | Stage Configuration

Build

Input Settings

Input Type: SCM Repository

Git Repository: interconnect-drinks

Git URL: https://github.com/2017-InterConnect-Demo-User/interconnect-drinks.git

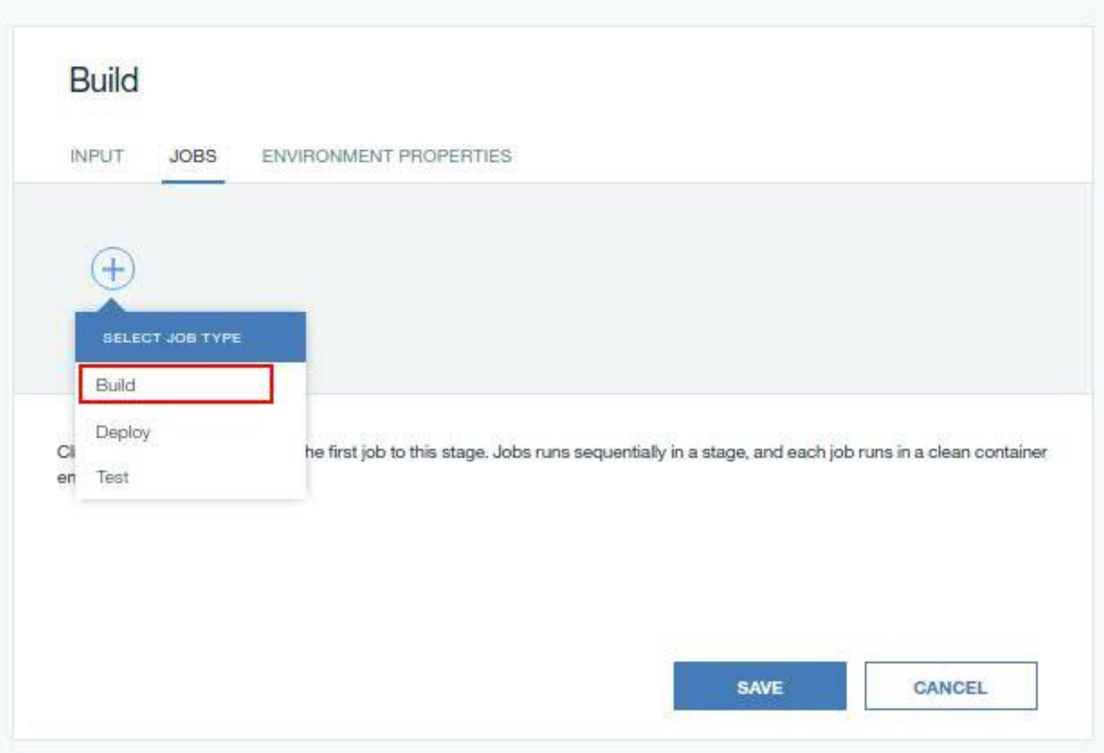
Branch: master

Stage Trigger

● Run jobs whenever a change is pushed to Git

○ Run jobs only when this stage is run manually

4.2.3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Build* for the job type

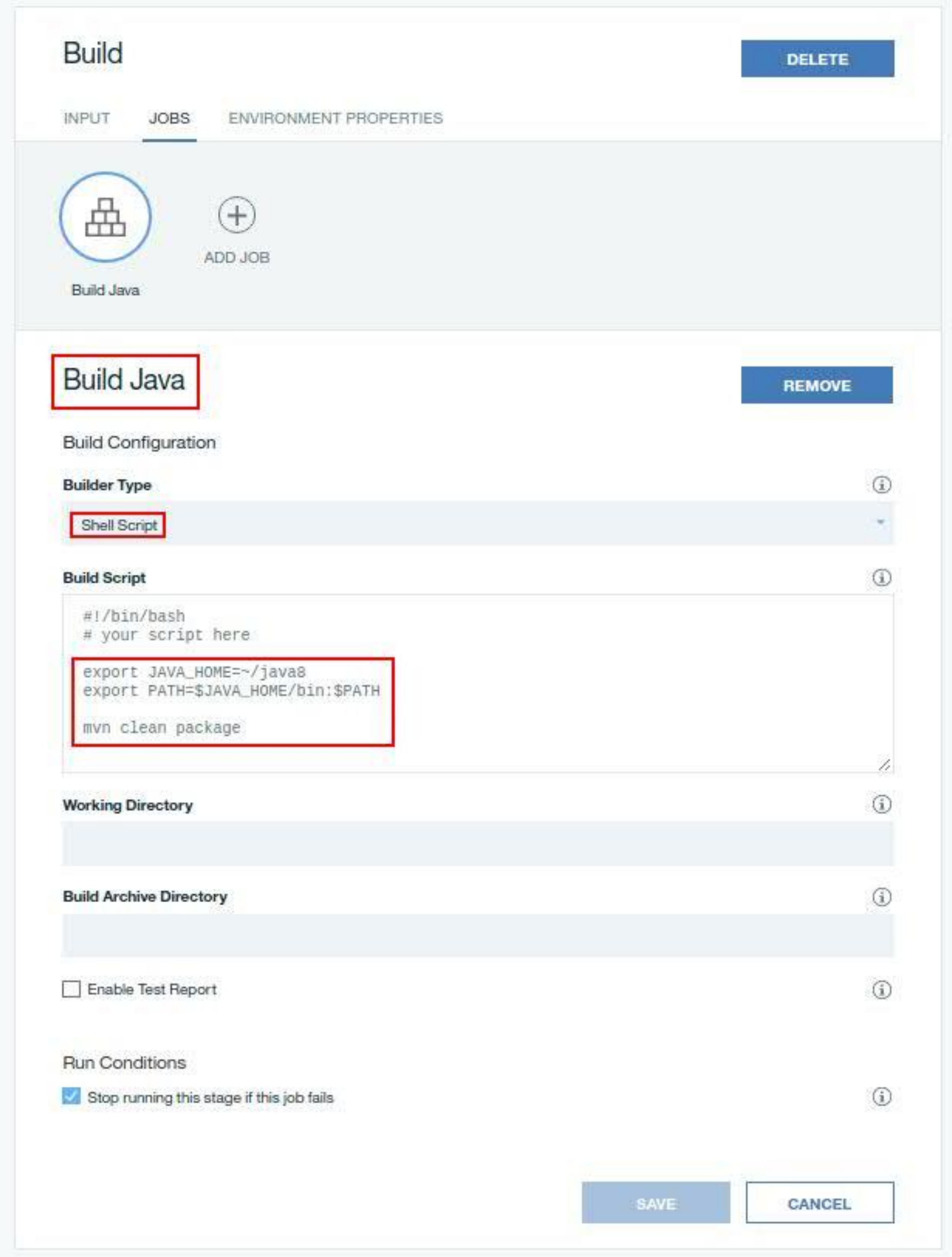


Cl he first job to this stage. Jobs runs sequentially in a stage, and each job runs in a clean container en

4.2.4. Name your job appropriately, select *Shell Script* for the *Builder Type* and add the following code to the out of the box *Build Script* provided

```
export JAVA_HOME=~/java8
export PATH=$JAVA_HOME/bin:$PATH
mvn clean package
```

FYI: export directives are needed since the default java version in the Delivery Pipelines tool is 1.7



Build Java

Build Configuration

Builder Type: Shell Script

Build Script

```
#!/bin/bash
# your script here

export JAVA_HOME=~/java8
export PATH=$JAVA_HOME/bin:$PATH

mvn clean package
```

Working Directory

Build Archive Directory

□ Enable Test Report

Run Conditions

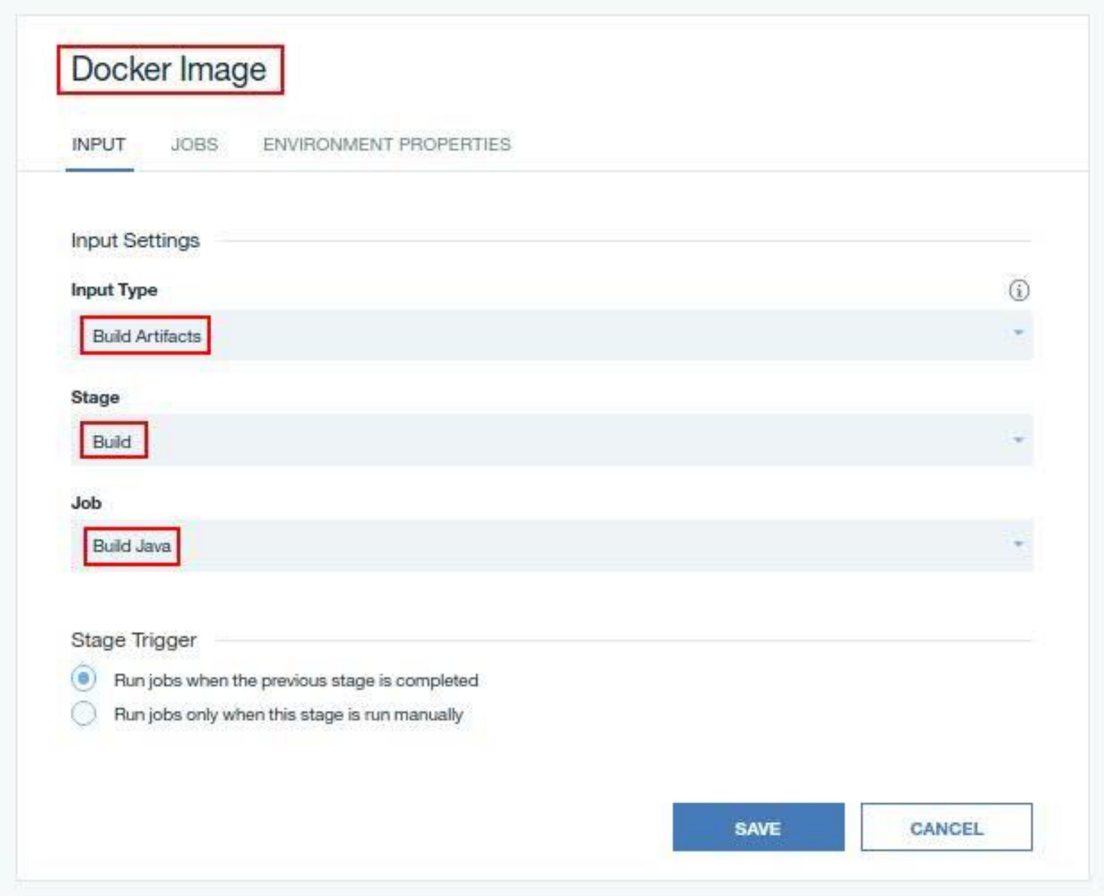
☑ Stop running this stage if this job fails

4.2.5. Click on *Save*

4.3. Create *Docker Image* stage

4.3.1. Click on *Add Stage*

4.3.2. Name your stage as Docker Image and make sure to configure the input of this stage to be the output of the previous Build stage


Docker Image

INPUT JOBS ENVIRONMENT PROPERTIES

Input Settings

Input Type: Build Artifacts

Stage: Build

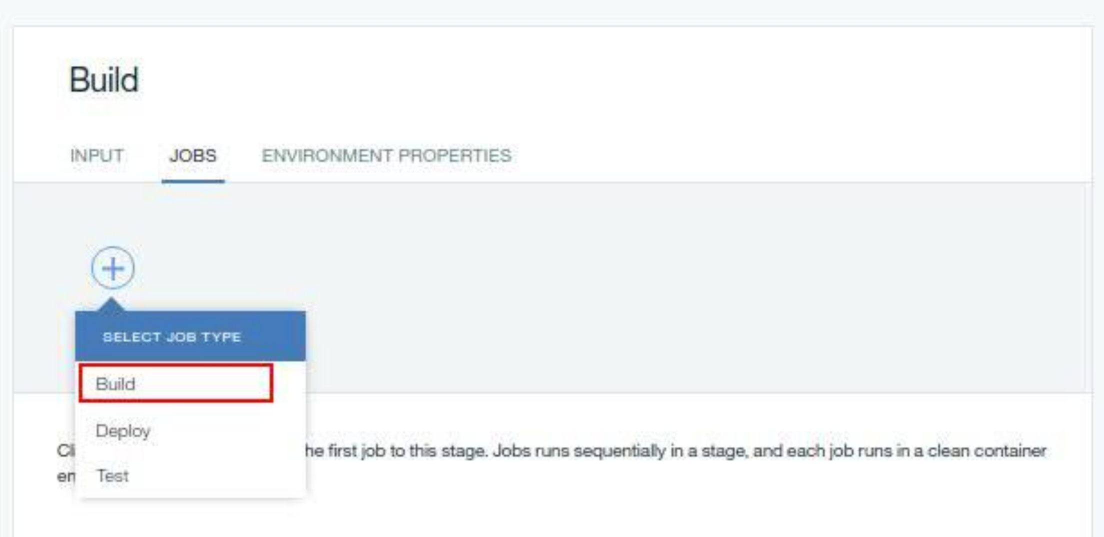
Job: Build Java

Stage Trigger

● Run jobs when the previous stage is completed

○ Run jobs only when this stage is run manually

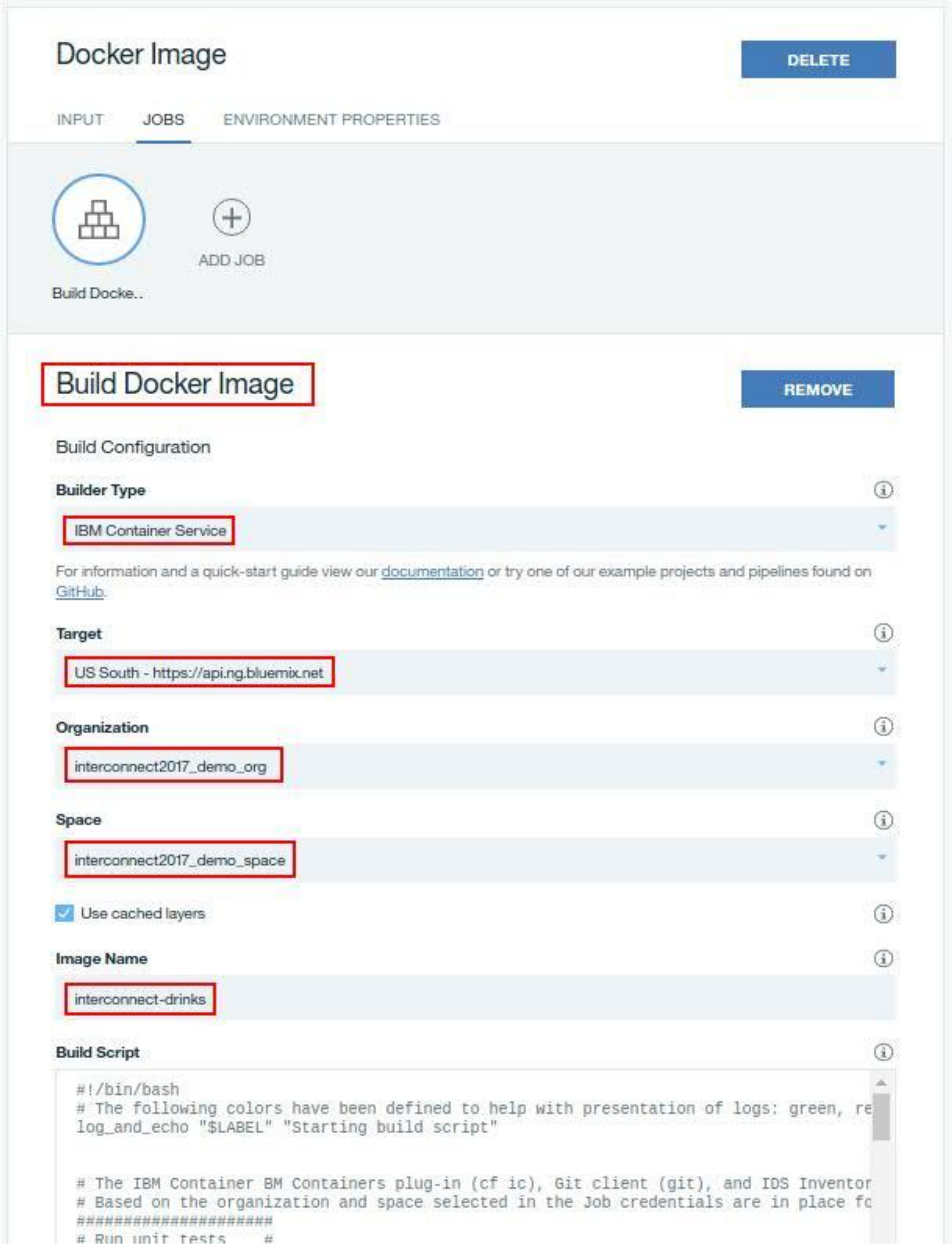
4.3.3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Build* for the job type


Build

INPUT JOBS ENVIRONMENT PROPERTIES

Cl… he first job to this stage. Jobs runs sequentially in a stage, and each job runs in a clean container en…

4.3.4. Name your job appropriately, select *IBM Container Service* as the *Builder Type*, select appropriate options for *Target, Organization and Space* fields and provide an appropriate image name. **Leave the out of the box *Build Script* as it is**



Build Docker Image

Build Configuration

Builder Type: IBM Container Service

For information and a quick-start guide view our documentation or try one of our example projects and pipelines found on GitHub.

Target: US South - https://api.ng.bluemix.net

Organization: interconnect2017_demo_org

Space: interconnect2017_demo_space

☑ Use cached layers

Image Name: interconnect-drinks

Build Script

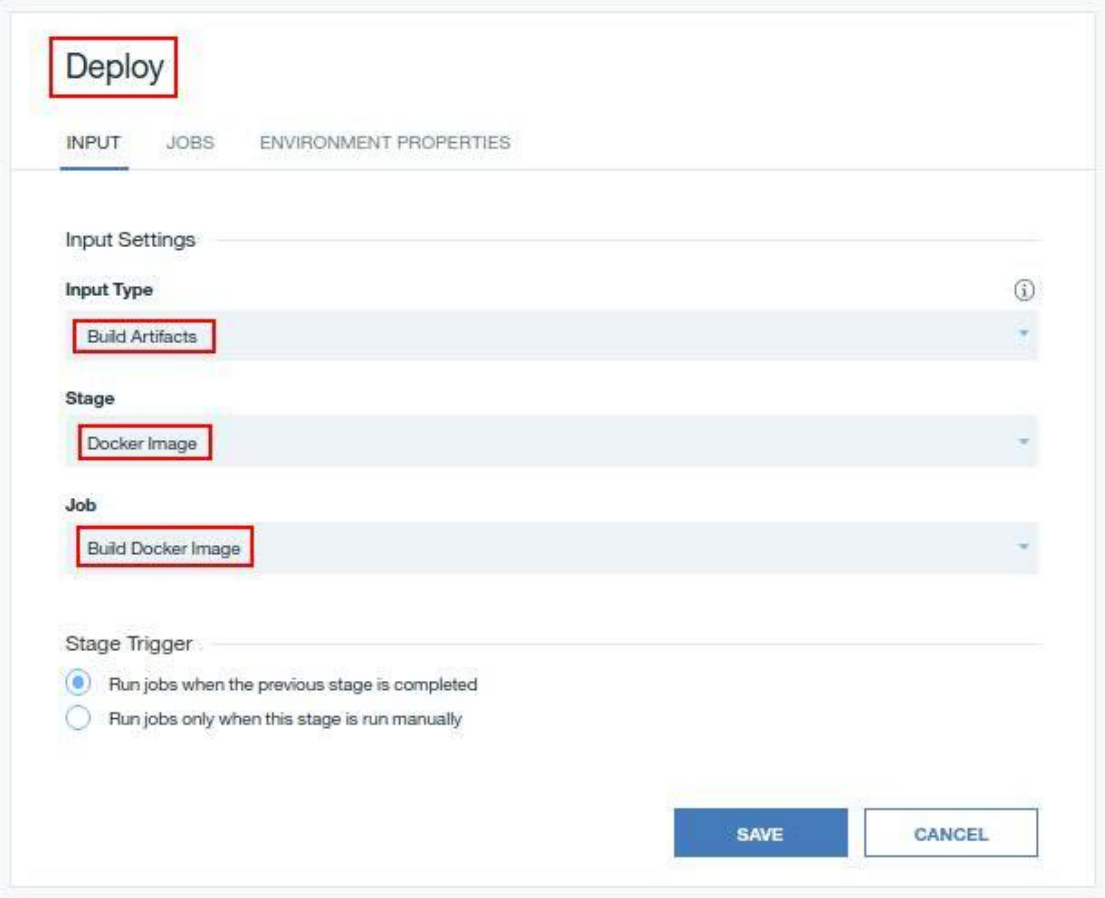
```
#!/bin/bash
# The following colors have been defined to help with presentation of logs: green, re
log_and_echo "$LABEL" "Starting build script"


# The IBM Container BM Containers plug-in (cf ic), Git client (git), and IDS Inventor
# Based on the organization and space selected in the Job credentials are in place fo
#####################
# Run unit tests    #
```

4.3.5. Click on *Save*

4.4. Create Deploy stage

4.4.1. Click on *Add Stage*

4.4.2. Name your stage as *Deploy* and make sure to configure the input for this stage to be the output of the previous *Docker Image* stage



Deploy

INPUT JOBS ENVIRONMENT PROPERTIES

Input Settings

Input Type: Build Artifacts

Stage: Docker Image

Job: Build Docker Image

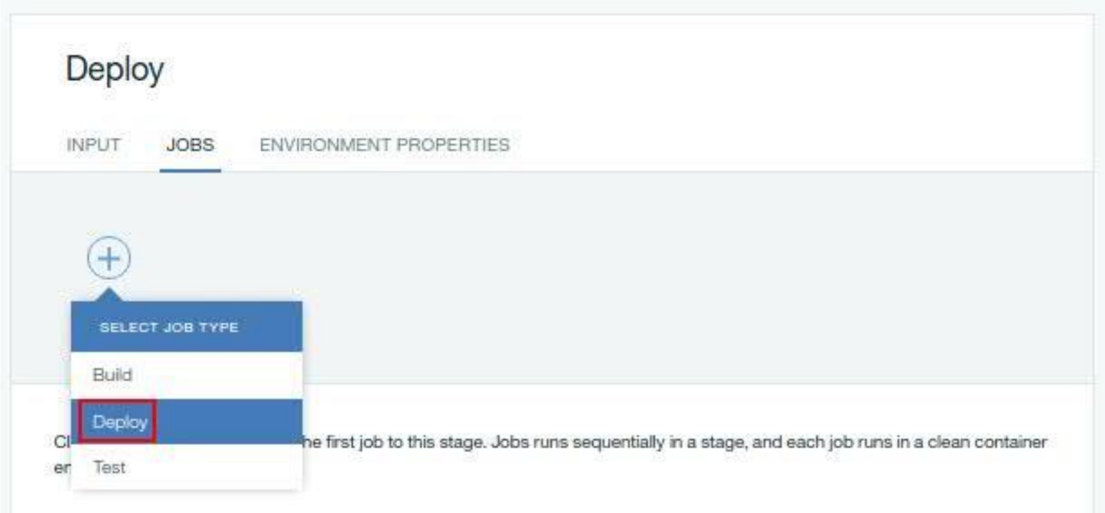
Stage Trigger

● Run jobs when the previous stage is completed

○ Run jobs only when this stage is run manually

SAVE CANCEL

4.4.3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Deploy* for the job type



Deploy

INPUT JOBS ENVIRONMENT PROPERTIES

SELECT JOB TYPE

Build

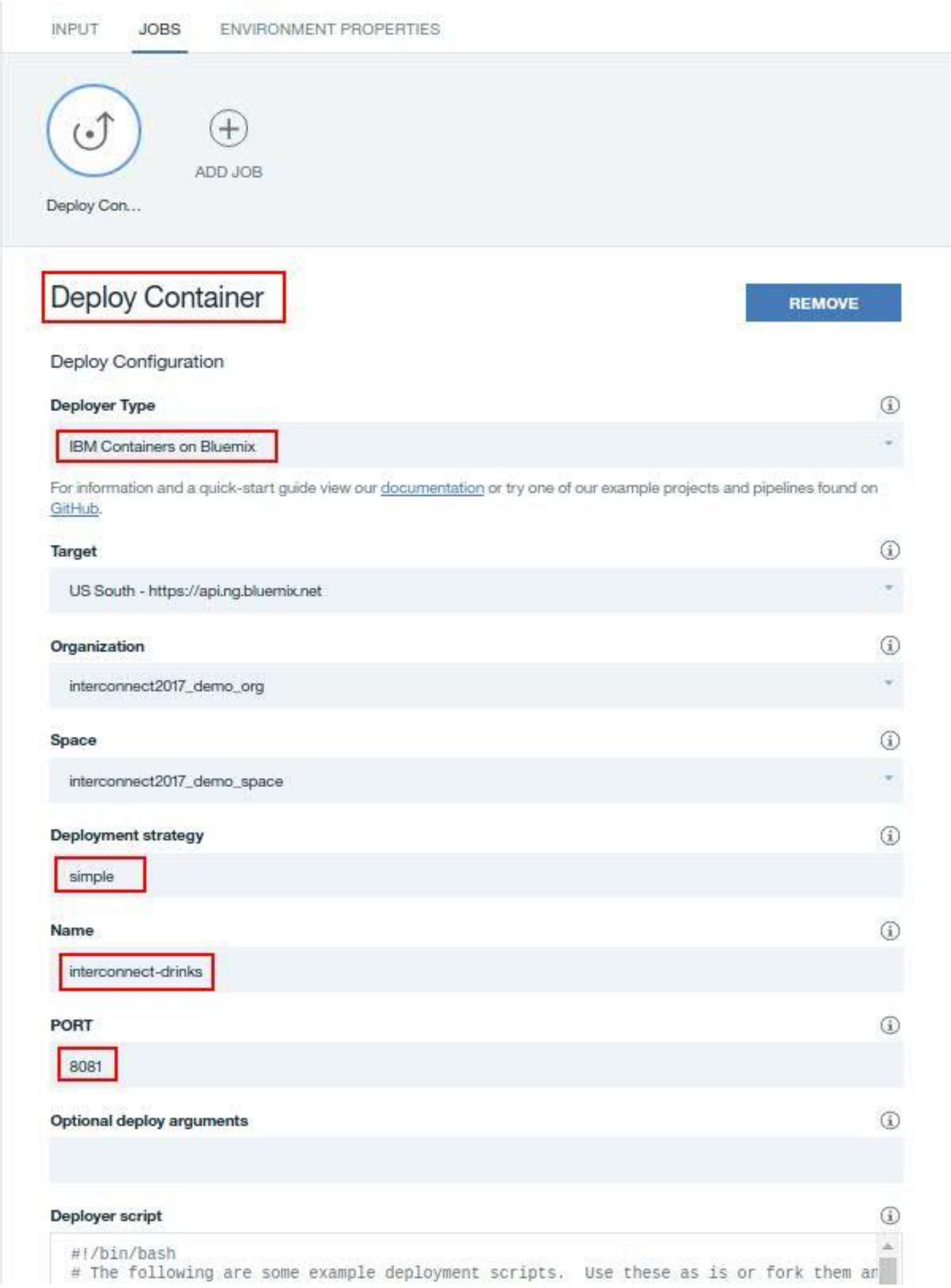
Deploy

Test

Cl he first job to this stage. Jobs runs sequentially in a stage, and each job runs in a clean container

er

4.4.4. Name your job appropriately, select *IBM Containers on Bluemix* as the *Deployer Type*, set your *Bluemix Target, Organization and Space*, use ***simple*** as the *Deployment strategy* and introduce a name for your container group plus the port it will expose.



Deploy Container

Deploy Configuration

Deployer Type
IBM Containers on Bluemix

For information and a quick-start guide view our documentation or try one of our example projects and pipelines found on GitHub.

Target
US South - https://api.ng.bluemix.net

Organization
interconnect2017_demo_org

Space
interconnect2017_demo_space

Deployment strategy
simple

Name
interconnect-drinks

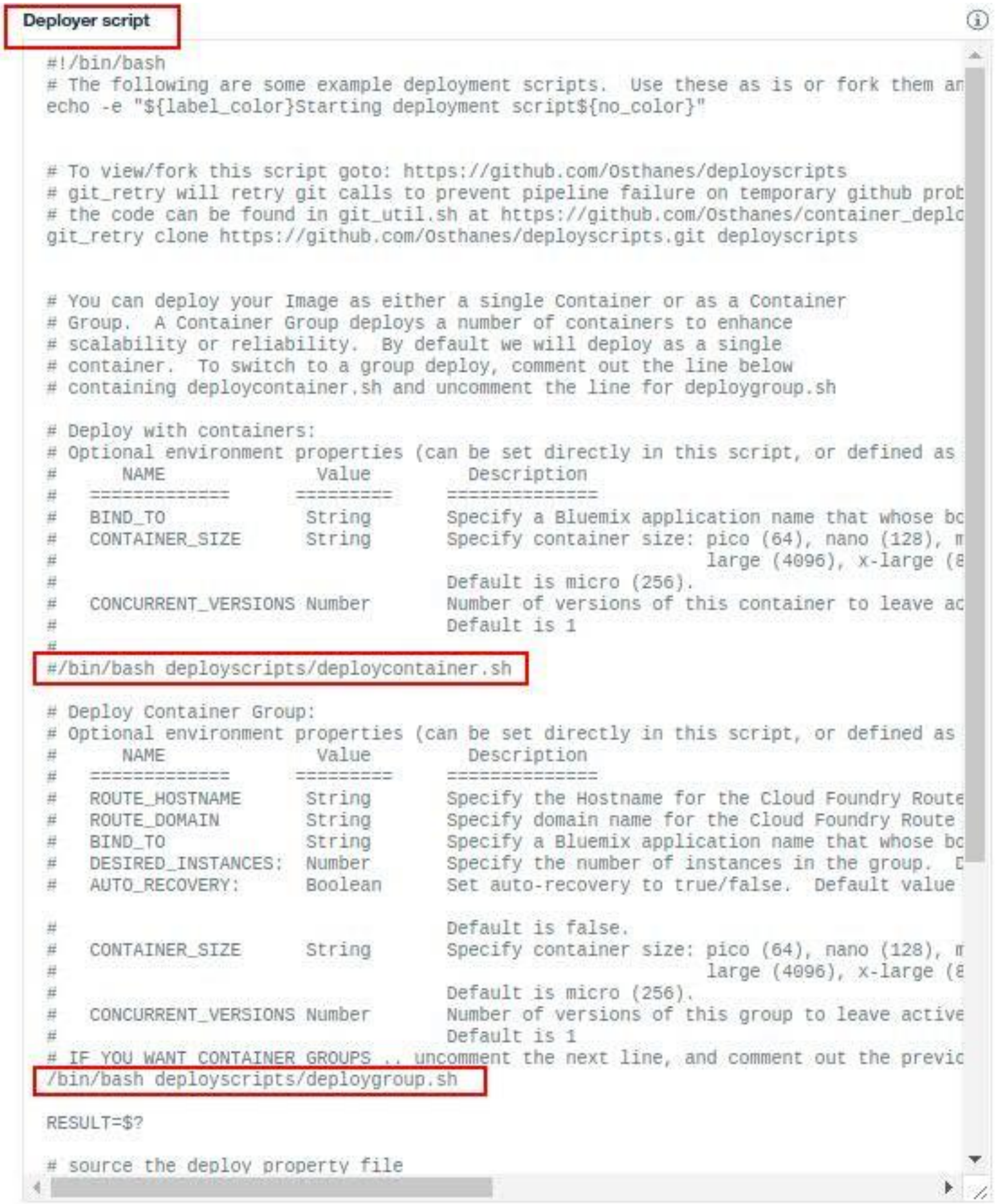
PORT
8081

Optional deploy arguments

Deployer script

```
#!/bin/bash
# The following are some example deployment scripts.  Use these as is or fork them ar
```

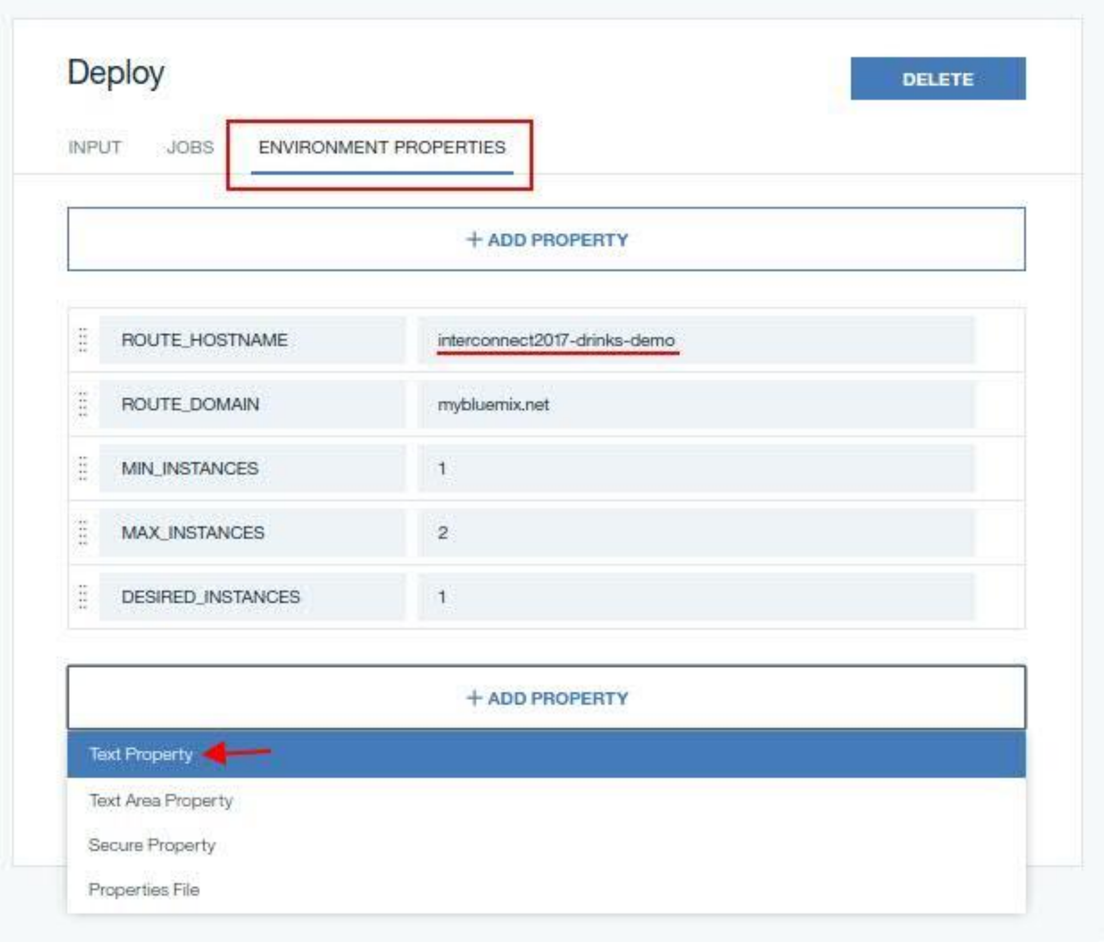
In order to get our Docker image deployed as a container group (as we have already done in this lab) we need to **comment** the line in the Deployer script that comes out of the box which deploys the image as a single container and **uncomment** the line to get it deployed as a container group



Deployer script

```
#!/bin/bash
# The following are some example deployment scripts.  Use these as is or fork them an
echo -e "${label_color}Starting deployment script${no_color}"


# To view/fork this script goto: https://github.com/Osthanes/deployscripts
# git_retry will retry git calls to prevent pipeline failure on temporary github prob
# the code can be found in git_util.sh at https://github.com/Osthanes/container_deplo
git_retry clone https://github.com/Osthanes/deployscripts.git deployscripts


# You can deploy your Image as either a single Container or as a Container
# Group.  A Container Group deploys a number of containers to enhance
# scalability or reliability.  By default we will deploy as a single
# container.  To switch to a group deploy, comment out the line below
# containing deploycontainer.sh and uncomment the line for deploygroup.sh

# Deploy with containers:
# Optional environment properties (can be set directly in this script, or defined as
#      NAME              Value        Description
#   =============      =========    ==============
#   BIND_TO             String       Specify a Bluemix application name that whose bo
#   CONTAINER_SIZE      String       Specify container size: pico (64), nano (128), m
#                                                              large (4096), x-large (8
#                                    Default is micro (256).
#   CONCURRENT_VERSIONS Number       Number of versions of this container to leave ac
#                                    Default is 1
#
#/bin/bash deployscripts/deploycontainer.sh

# Deploy Container Group:
# Optional environment properties (can be set directly in this script, or defined as
#      NAME              Value        Description
#   =============      =========    ==============
#   ROUTE_HOSTNAME      String       Specify the Hostname for the Cloud Foundry Route
#   ROUTE_DOMAIN        String       Specify domain name for the Cloud Foundry Route
#   BIND_TO             String       Specify a Bluemix application name that whose bo
#   DESIRED_INSTANCES:  Number       Specify the number of instances in the group.  D
#   AUTO_RECOVERY:      Boolean      Set auto-recovery to true/false.  Default value

#                                    Default is false.
#   CONTAINER_SIZE      String       Specify container size: pico (64), nano (128), m
#                                                              large (4096), x-large (8
#                                    Default is micro (256).
#   CONCURRENT_VERSIONS Number       Number of versions of this group to leave active
#                                    Default is 1
# IF YOU WANT CONTAINER GROUPS .. uncomment the next line, and comment out the previo
/bin/bash deployscripts/deploygroup.sh

RESULT=$?

# source the deploy property file
```

4.4.5. We need to pass similar arguments as the ones used in the command line to the *Deployer*. Hence, click on the *ENVIRONMENT PROPERTIES* tab and add as many properties as the following picture displays by clicking on *ADD PROPERTY* and selecting **Text property**



ROUTE_HOSTNAME	interconnect2017-drinks-demo
ROUTE_DOMAIN	mybluemix.net
MIN_INSTANCES	1
MAX_INSTANCES	2
DESIRED_INSTANCES	1

IMPORTANT: *We could create the container groups without creating a public route to them by not specifying the ROUTE_HOSTNAME and ROUTE_DOMAIN variables since we cloud communicate with them through their container group's load balancer IP addresses. However, we would need to redeploy the menu container group any time we deploy a new version of either food, drinks or both since their load balancers IP addresses would change too. Moreover, Active Deploy only works with container groups with a public route attached to them. Therefore, and despite the security drop attaching a public route to a container group implies, the menu app will use the drinks and food public routes for communication. Bear in mind that your ROUTE_HOSTNAME value must be* ***unique*** *within the Bluemix region you are deploying your applications to.*

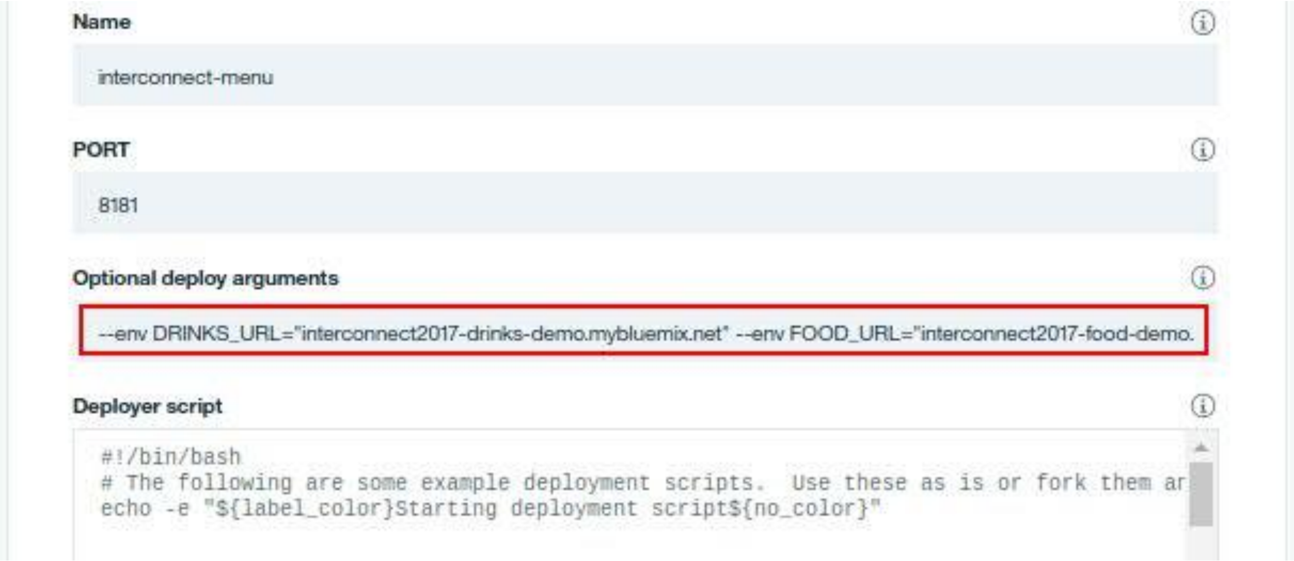
4.4.6. Click on *Save*

4.5. Repeat steps 4.1. to 4.4. for the drinks and food Java apps.

4.6. Repeat steps 4.1. to 4.4.3 for the menu Java app

4.7. We need a slightly different step 4.4.4. for the menu Java app since we need to pass the food and drinks public routes attached to their container groups to the menu app as environment variables like we have done before in this lab. Hence, add the following parameters in the *Optional deploy arguments* text field

```
--env
DRINKS_URL="{your_specified_route_in_drinks_deploy_stage}" --
env FOOD_URL="{ your_specified_route_in_food_deploy_stage }"
```



Name

interconnect-menu

PORT

8181

Optional deploy arguments

--env DRINKS_URL="interconnect2017-drinks-demo.mybluemix.net" --env FOOD_URL="interconnect2017-food-demo.

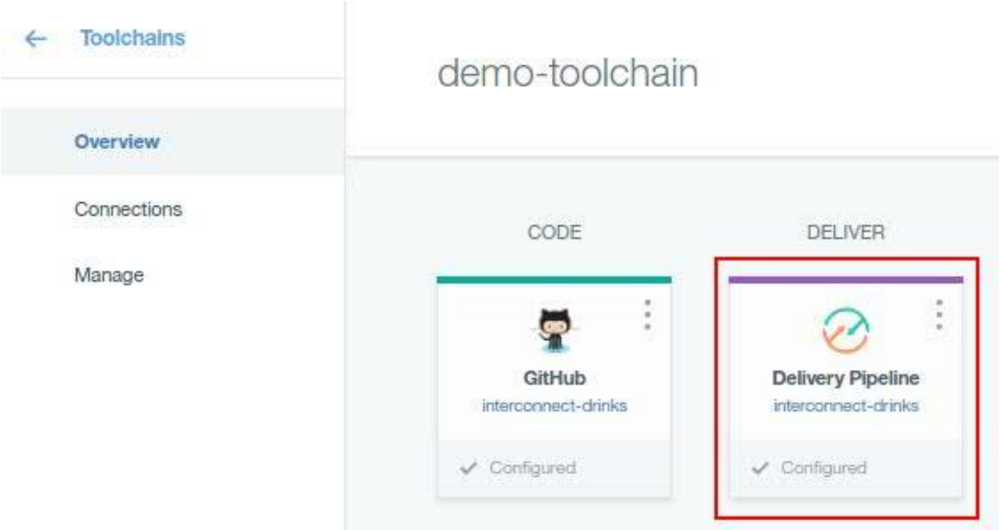
Deployer script

```
#!/bin/bash
# The following are some example deployment scripts.  Use these as is or fork them ar
echo -e "${label_color}Starting deployment script${no_color}"
```

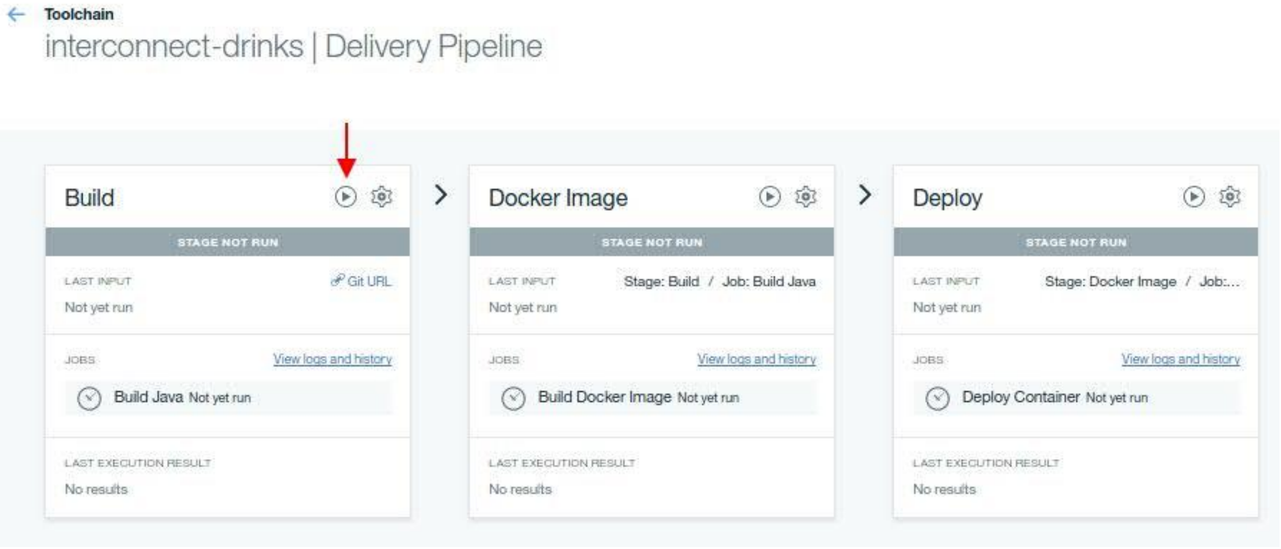
4.8. Repeat steps 4.4.5 and 4.4.6 for the menu app

5. **Execute** your delivery pipelines
   This will get your three java microservices built, dockerized and/or containerized and deployed as container groups in *IBM Bluemix public cloud* from their Docker images in just one click!

   5.1. Click on the food or drinks delivery pipeline icon in the Toolchain dashboard



5.2. Click on the play button on the top right corner of the Build stage, the first stage.



Toolchain

interconnect-drinks | Delivery Pipeline

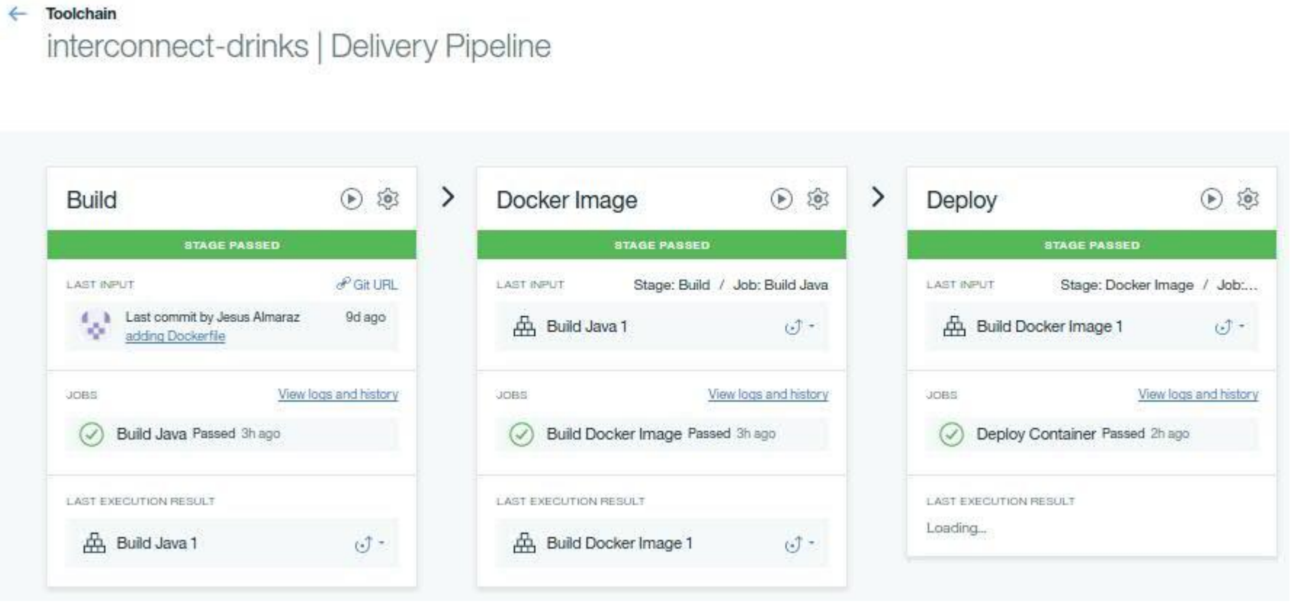
Build
STAGE NOT RUN
LAST INPUT — Git URL
Not yet run
JOBS — View logs and history
Build Java Not yet run
LAST EXECUTION RESULT
No results

Docker Image
STAGE NOT RUN
LAST INPUT — Stage: Build / Job: Build Java
Not yet run
JOBS — View logs and history
Build Docker Image Not yet run
LAST EXECUTION RESULT
No results

Deploy
STAGE NOT RUN
LAST INPUT — Stage: Docker Image / Job:...
Not yet run
JOBS — View logs and history
Deploy Container Not yet run
LAST EXECUTION RESULT
No results

** This will trigger the build process of your application and will then trigger the next stage and so on.*
*** **Refresh** the web page in order to see the progress along the pipeline*

5.3. After the delivery pipeline has finished, you should see the three stages in green



Toolchain

interconnect-drinks | Delivery Pipeline

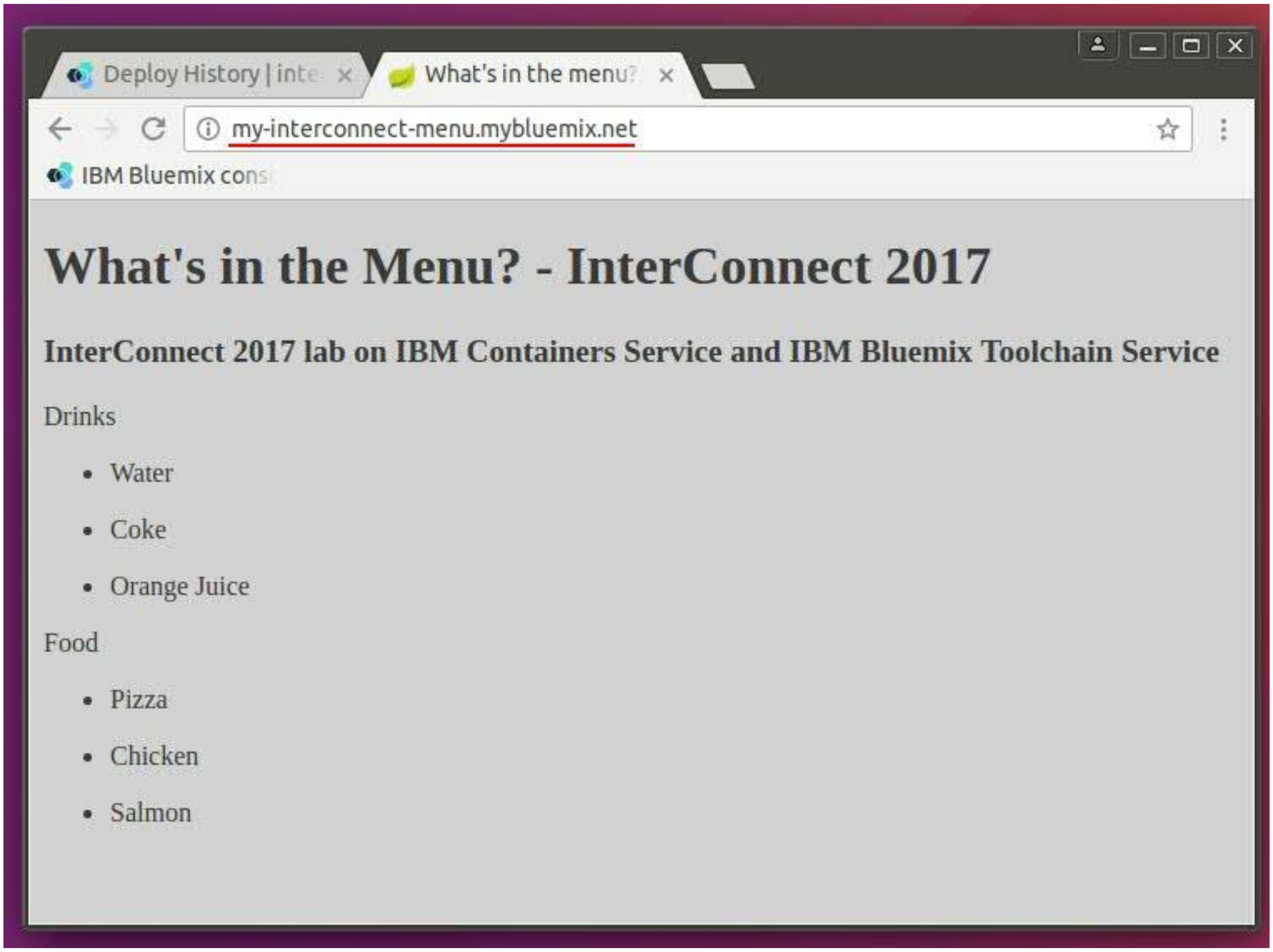
Build
STAGE PASSED
LAST INPUT — Git URL
Last commit by Jesus Almaraz 9d ago
adding Dockerfile
JOBS — View logs and history
Build Java Passed 3h ago
LAST EXECUTION RESULT
Build Java 1

Docker Image
STAGE PASSED
LAST INPUT — Stage: Build / Job: Build Java
Build Java 1
JOBS — View logs and history
Build Docker Image Passed 3h ago
LAST EXECUTION RESULT
Build Docker Image 1

Deploy
STAGE PASSED
LAST INPUT — Stage: Docker Image / Job:...
Build Docker Image 1
JOBS — View logs and history
Deploy Container Passed 2h ago
LAST EXECUTION RESULT
Loading...

5.4. Repeat the process for the food and menu delivery pipelines. Menu pipeline **must be executed the last one** since it needs other two container groups' load balancer's IP address.

5.5. Check the application has been properly built, containerized and deployed onto the IBM Bluemix public cloud by pointing your web browser to the *ROUTE_HOSTNAME+ROUTE_DOMAIN* values you specified in the menu delivery pipeline



What's in the Menu? - InterConnect 2017

InterConnect 2017 lab on IBM Containers Service and IBM Bluemix Toolchain Service

Drinks

- Water
- Coke
- Orange Juice

Food

- Pizza
- Chicken
- Salmon

IBM Bluemix Active Deploy Service

You can deploy a new version of your running apps or container groups with no downtime. IBM Bluemix Active Deploy Service uses a phased approach to deployments, giving you time to test and validate your updated app in a production environment.

With the Active Deploy service, you can deploy a new version of your app or container group by using a continual process, such that the new version is finalized only when it proves to work properly in production. In a nutshell, you have these options:

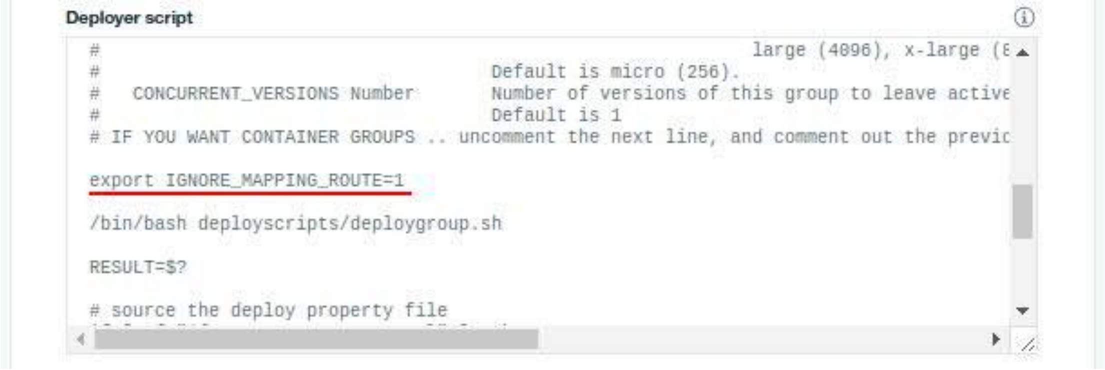
- Manage the rollout of changes in an automated and controlled fashion.
- Roll back your changes if needed.
- Have two versions be "live" at one time, each one with specific amounts of routed traffic.

Follow next steps in order to get Active Deploy integrated in your delivering pipelines so that you have an agile, continuous and zero-downtime deployment of your applications.

While this should be done for the three Java applications, we are going to make the following changes **only to the menu delivery pipeline** since that is enough to see how to integrate Active Deploy with Delivery Pipelines as well as the zero downtime deployment.

1. Modify the actual *Deploy Container job* within the *Deploy stage* to integrate with Active Deploy
   1.1. Add the following line just before the container group creation script kicks off so that the container group is created without a route for now

```
export IGNORE_MAPPING_ROUTE=1
```

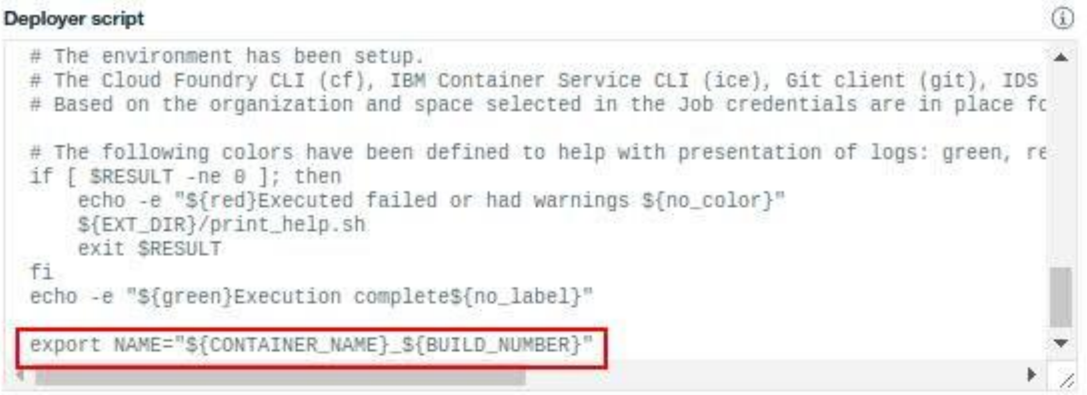


Deployer script

```
#                                                    large (4096), x-large (8
#                                   Default is micro (256).
#   CONCURRENT_VERSIONS Number      Number of versions of this group to leave active
#                                   Default is 1
# IF YOU WANT CONTAINER GROUPS .. uncomment the next line, and comment out the previc

export IGNORE_MAPPING_ROUTE=1

/bin/bash deployscripts/deploygroup.sh

RESULT=$?

# source the deploy property file
```

   1.2. Add the following line at the end of the *Deployer Script* text field

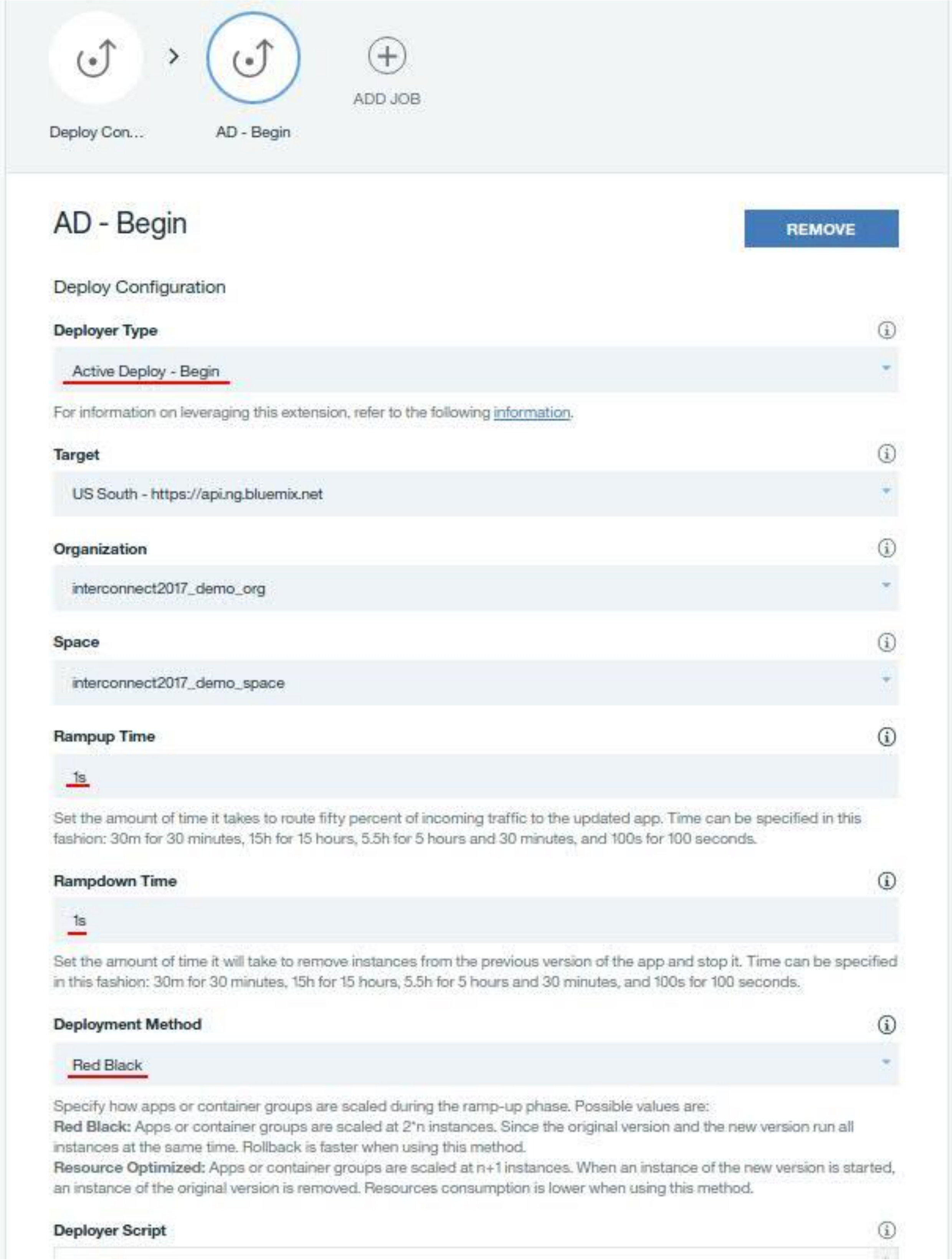
```
export NAME="${CONTAINER_NAME}_${BUILD_NUMBER}"
```



Deployer script

```
# The environment has been setup.
# The Cloud Foundry CLI (cf), IBM Container Service CLI (ice), Git client (git), IDS
# Based on the organization and space selected in the Job credentials are in place fc

# The following colors have been defined to help with presentation of logs: green, re
if [ $RESULT -ne 0 ]; then
    echo -e "${red}Executed failed or had warnings ${no_color}"
    ${EXT_DIR}/print_help.sh
    exit $RESULT
fi
echo -e "${green}Execution complete${no_label}"

export NAME="${CONTAINER_NAME}_${BUILD_NUMBER}"
```

1.3. Create a new Deploy job, give it an appropriate name, select *Active Deploy – Begin* as the *Deployer type* and configure it like the picture below shows



AD - Begin

REMOVE

Deploy Configuration

Deployer Type

Active Deploy - Begin

For information on leveraging this extension, refer to the following information.

Target

US South - https://api.ng.bluemix.net

Organization

interconnect2017_demo_org

Space

interconnect2017_demo_space

Rampup Time

1s

Set the amount of time it takes to route fifty percent of incoming traffic to the updated app. Time can be specified in this fashion: 30m for 30 minutes, 15h for 15 hours, 5.5h for 5 hours and 30 minutes, and 100s for 100 seconds.

Rampdown Time

1s

Set the amount of time it will take to remove instances from the previous version of the app and stop it. Time can be specified in this fashion: 30m for 30 minutes, 15h for 15 hours, 5.5h for 5 hours and 30 minutes, and 100s for 100 seconds.

Deployment Method

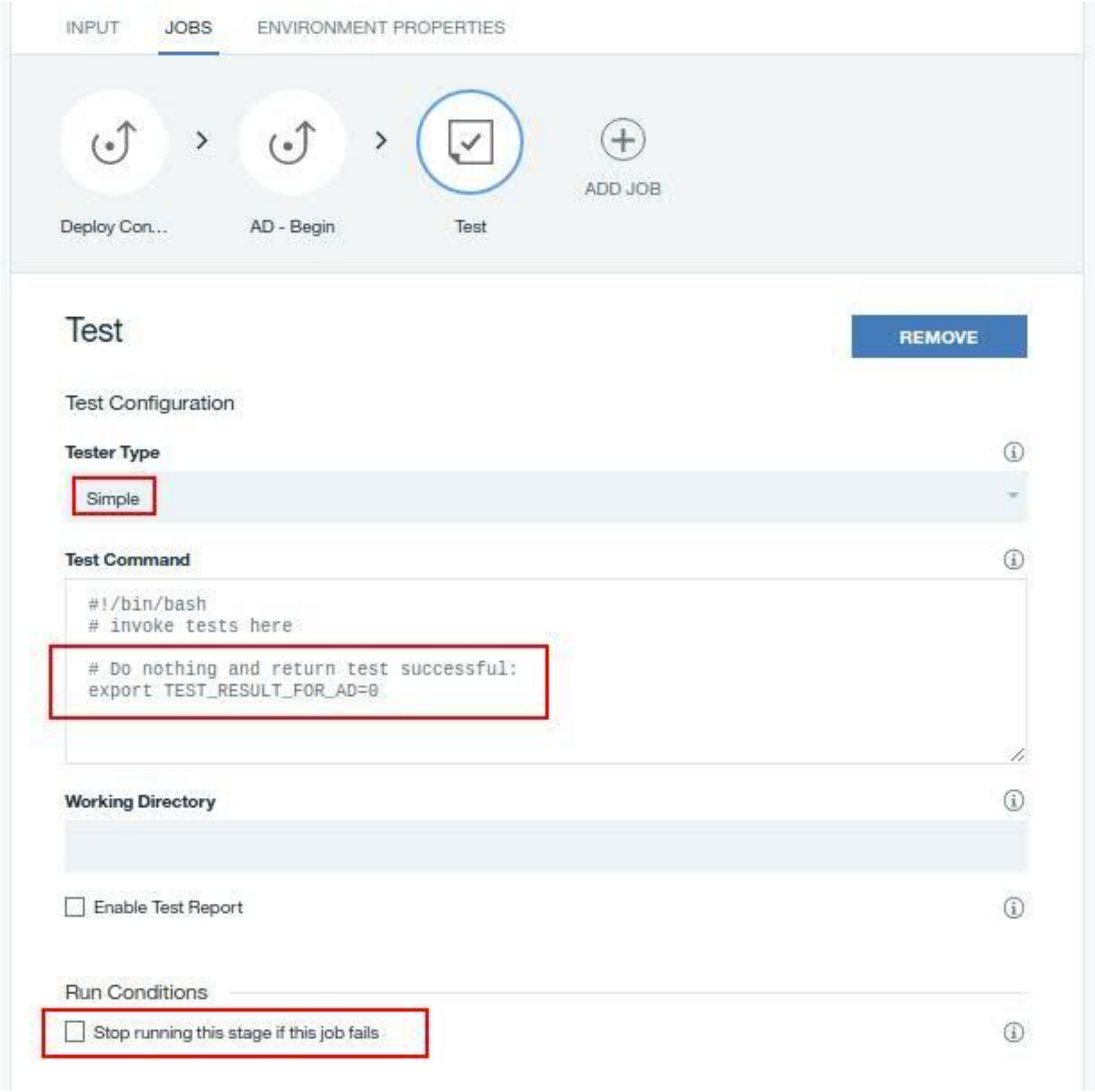
Red Black

Specify how apps or container groups are scaled during the ramp-up phase. Possible values are:
Red Black: Apps or container groups are scaled at 2*n instances. Since the original version and the new version run all instances at the same time. Rollback is faster when using this method.
Resource Optimized: Apps or container groups are scaled at n+1 instances. When an instance of the new version is started, an instance of the original version is removed. Resources consumption is lower when using this method.

Deployer Script

The *Active Deploy – Begin* job will create a new container group and route traffic to it.

1.4. Create a new Test job, name it appropriately and configure it as the picture below shows



Test

Test Configuration

Tester Type

Simple

Test Command

```
#!/bin/bash
# invoke tests here

# Do nothing and return test successful:
export TEST_RESULT_FOR_AD=0
```

Working Directory

□ Enable Test Report

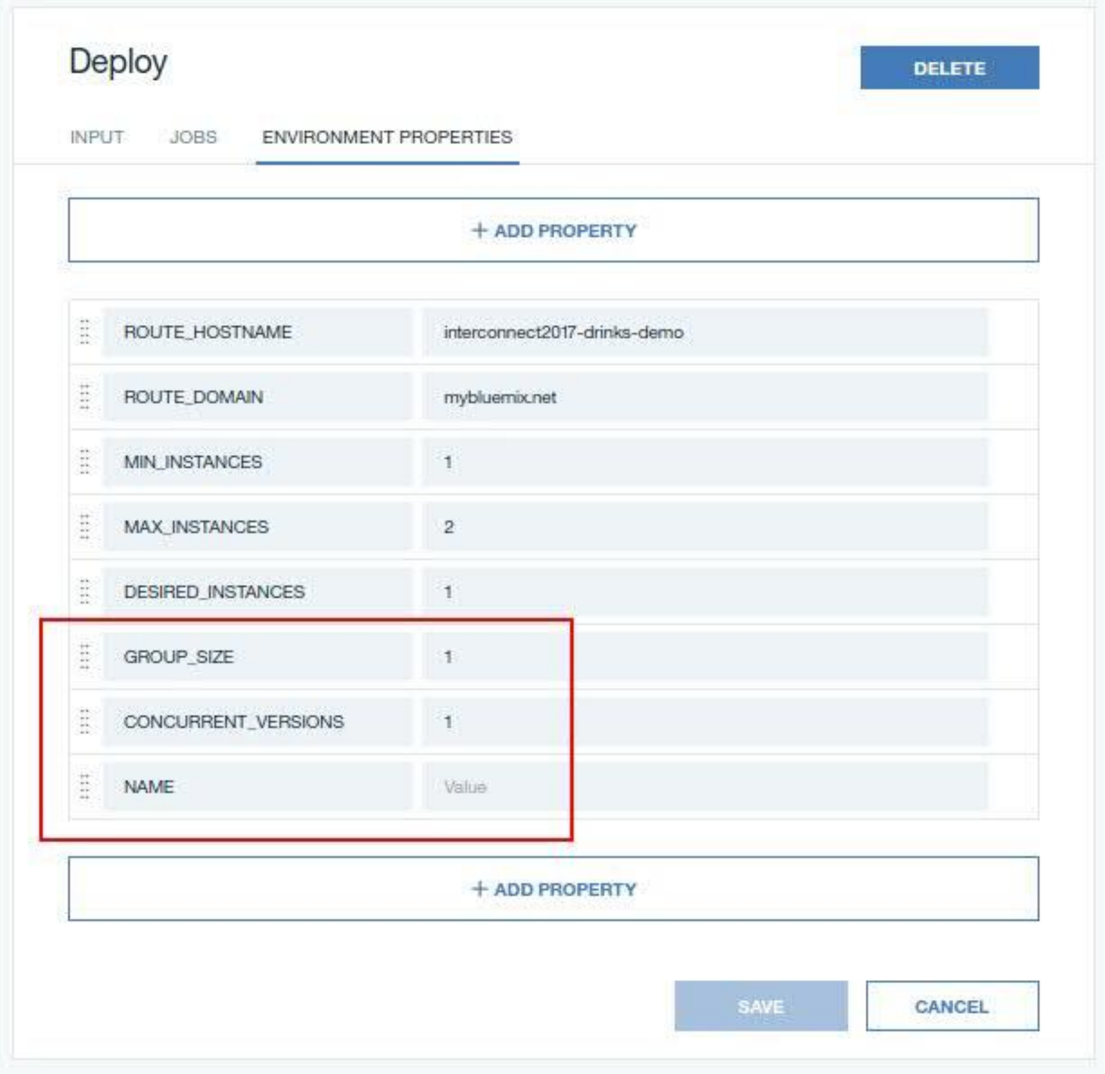
Run Conditions

□ Stop running this stage if this job fails

The directive in the *Test Command* field serves to perform any test now that we have both new and old app versions running and receiving traffic. Since we do not want to perform any test, we add the directive you see in the picture to return always success.

IMPORTANT: Clear the *Stop running this stage if this job fails* check box for all test jobs to allow the *Active Deploy - Complete* job to run. The *Active Deploy – Complete* job must run to be able to roll back the deployment in case tests fail, or decrease the instances of the original version of the app if tests succeed.

1.5. Create a new Deploy job, give it an appropriate name, select *Active Deploy – Complete* as the *Deployer type* and leave the rest as it is.
1.6. Go into the environment tab and add the following ones



Deploy

ROUTE_HOSTNAME	interconnect2017-drinks-demo
ROUTE_DOMAIN	mybluemix.net
MIN_INSTANCES	1
MAX_INSTANCES	2
DESIRED_INSTANCES	1
GROUP_SIZE	1
CONCURRENT_VERSIONS	1
NAME	Value

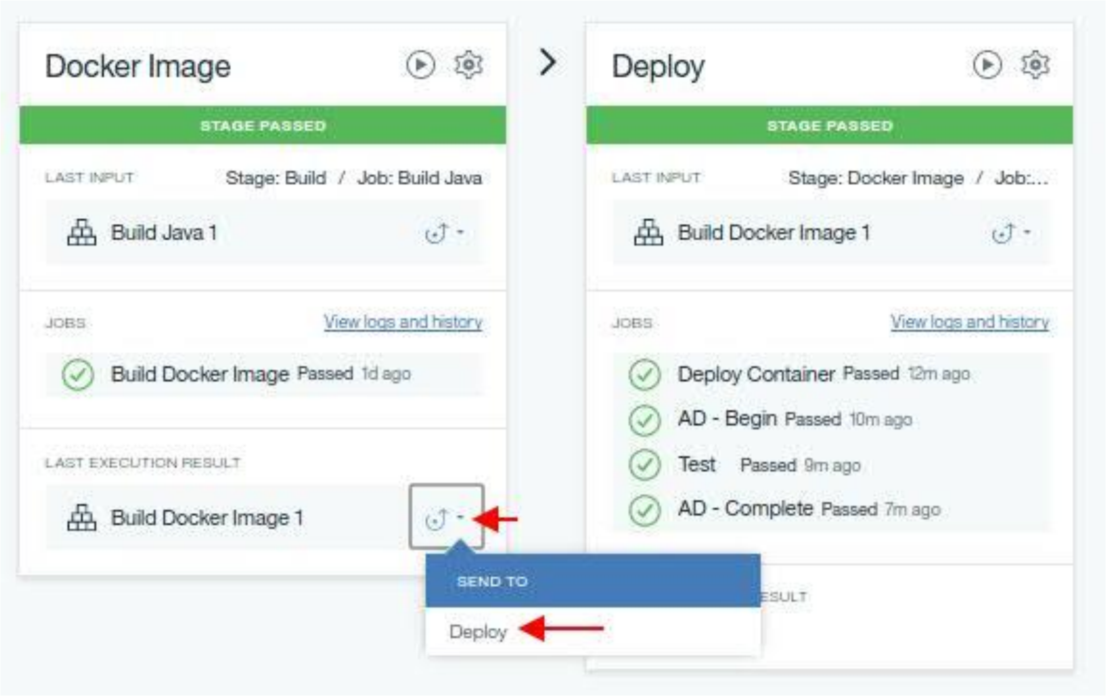
where *GROUP_SIZE* is the size of the new container group created by Active Deploy, the *CONCURRENT_VERSIONS* the number of concurrent versions Active Deploy will keep active and deployed and *NAME* the name of the container group.

We are setting the concurrent versions to be only one so that we see how the menu changes when we deploy a new version of it.

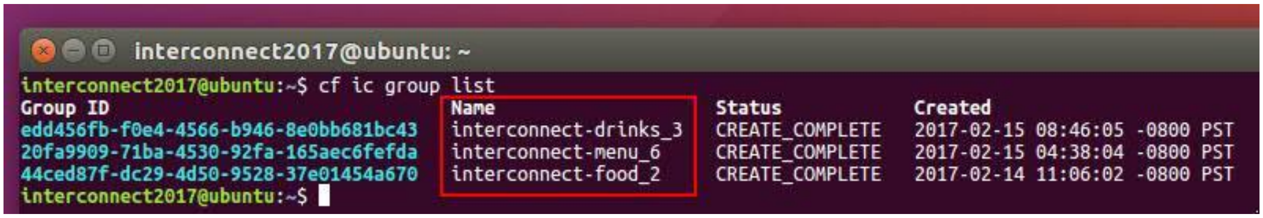
We leave the *NAME* value empty since it will get a value with the export directive added to the *deploy container* stage in previous step.

1.7. Click on Save

2. Execute the deploy stage so that new jobs get done and a new version of the menu app is deployed.
   2.1. Since there has not been any change in the code but in the deployment process, we can skip the *Build* and *Docker Image* stages and directly send the current output artifact of the *Docker Image* stage to the *Deploy* stage



   2.2. After the Deploy stage has finished, we should see the three new jobs within the stage in green (see picture above). Moreover, if we list the container groups on the CLI we will see new versions of the container groups

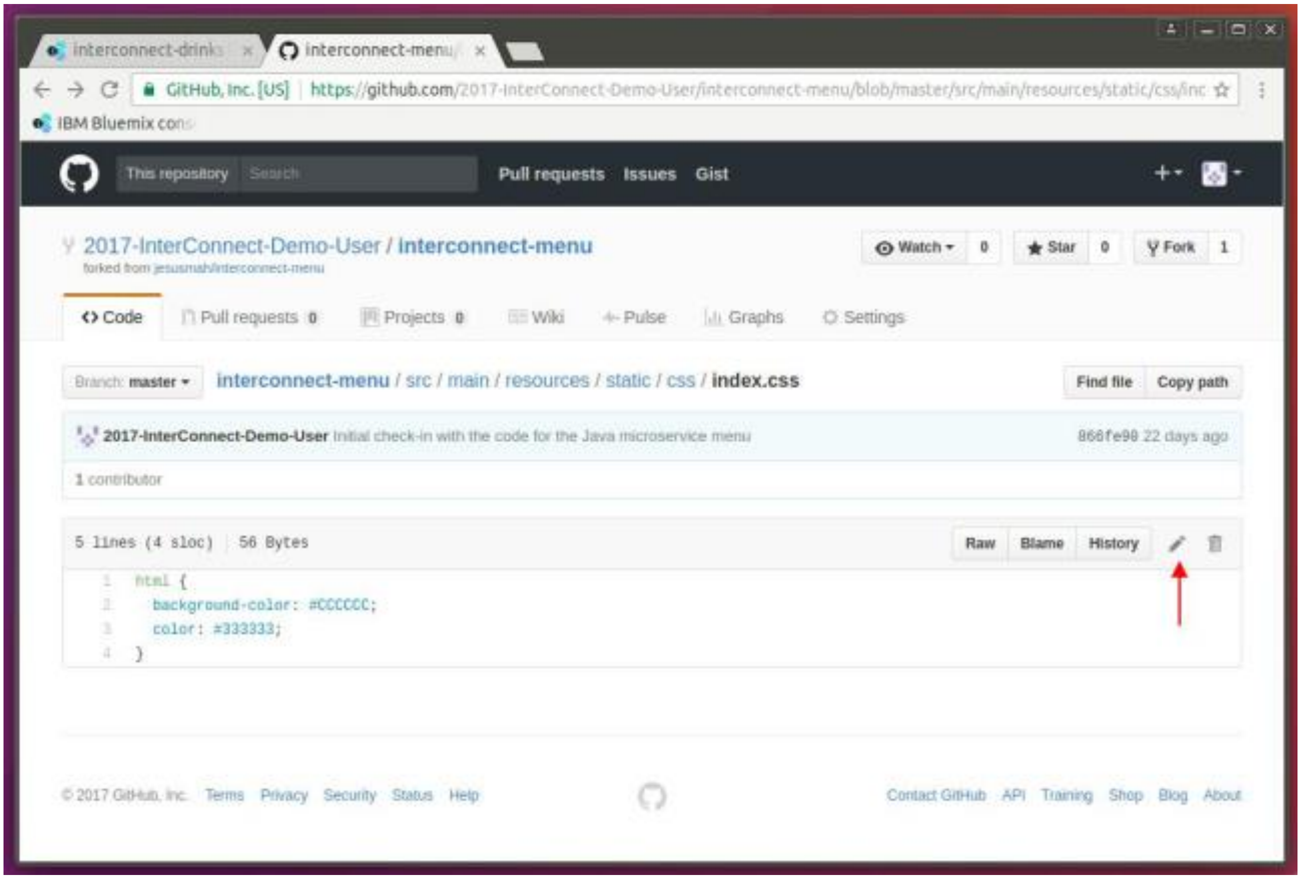


```
interconnect2017@ubuntu:~$ cf ic group list
Group ID                              Name                   Status           Created
edd456fb-f0e4-4566-b946-8e0bb681bc43  interconnect-drinks_3  CREATE_COMPLETE  2017-02-15 08:46:05 -0800 PST
20fa9909-71ba-4530-92fa-165aec6fefda  interconnect-menu_6    CREATE_COMPLETE  2017-02-15 04:38:04 -0800 PST
44ced87f-dc29-4d50-9528-37e01454a670  interconnect-food_2    CREATE_COMPLETE  2017-02-14 11:06:02 -0800 PST
interconnect2017@ubuntu:~$
```

3. End to end CI/CD example
   3.1. Open in your browser the following file belonging to the menu project and living in your GitHub repo

   `https://github.com/`*`{your_github_username}`*`/interconnect-menu/blob/master/src/main/resources/static/css/index.css`

   3.2. Click on the pencil to edit the file



2017-InterConnect-Demo-User / interconnect-menu

forked from jesusmah/interconnect-menu

interconnect-menu / src / main / resources / static / css / index.css

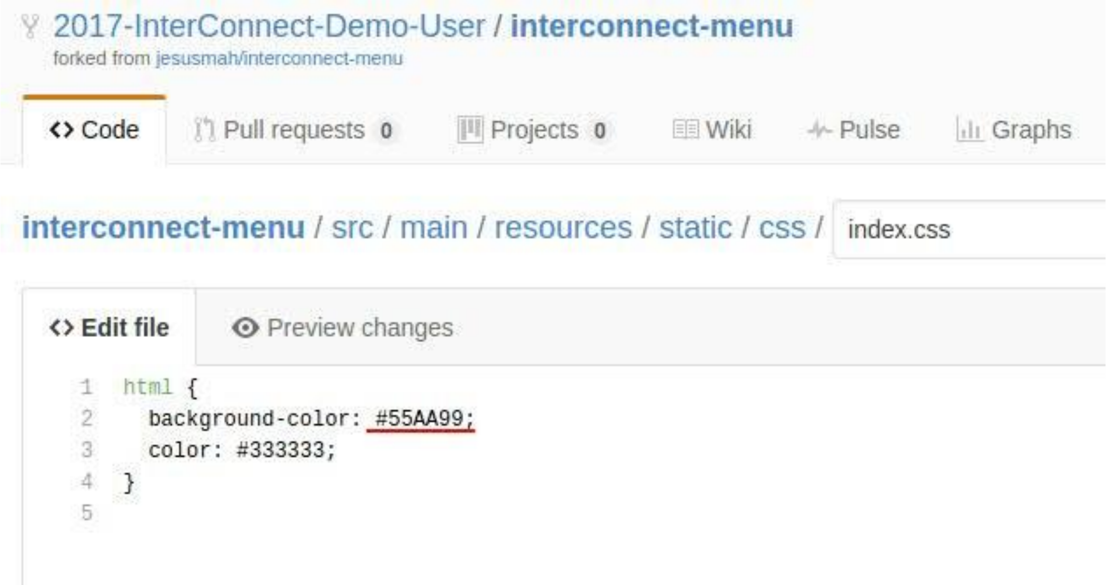
2017-InterConnect-Demo-User Initial check-in with the code for the Java microservice menu 866fe98 22 days ago

1 contributor

5 lines (4 sloc) 56 Bytes

```
html {
  background-color: #CCCCCC;
  color: #333333;
}
```

3.3. Change the background color to something different



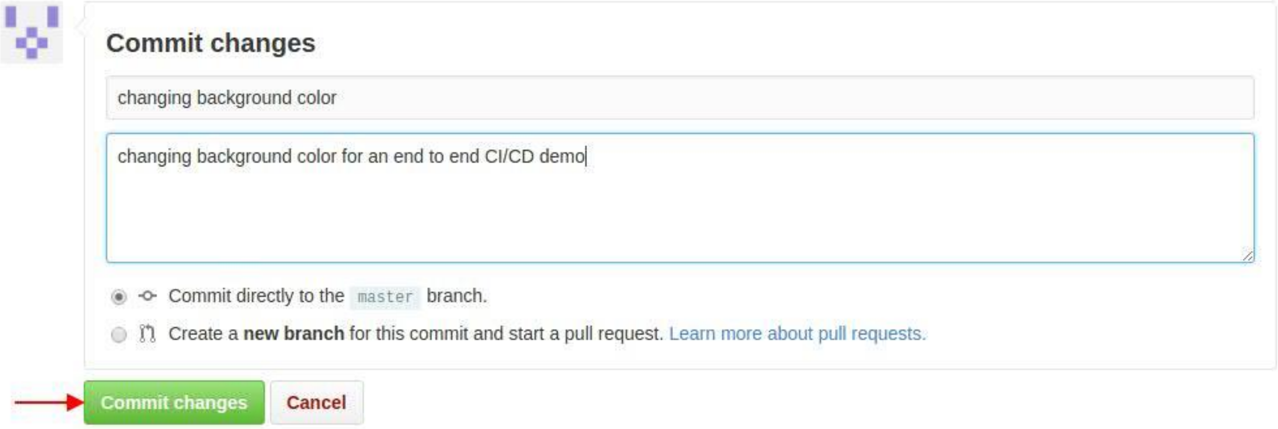
2017-InterConnect-Demo-User / interconnect-menu

forked from jesusmah/interconnect-menu

interconnect-menu / src / main / resources / static / css / index.css

```
html {
  background-color: #55AA99;
  color: #333333;
}

```

3.4. Scroll to the bottom of the page, include some comments about the change and commit the changes



Commit changes

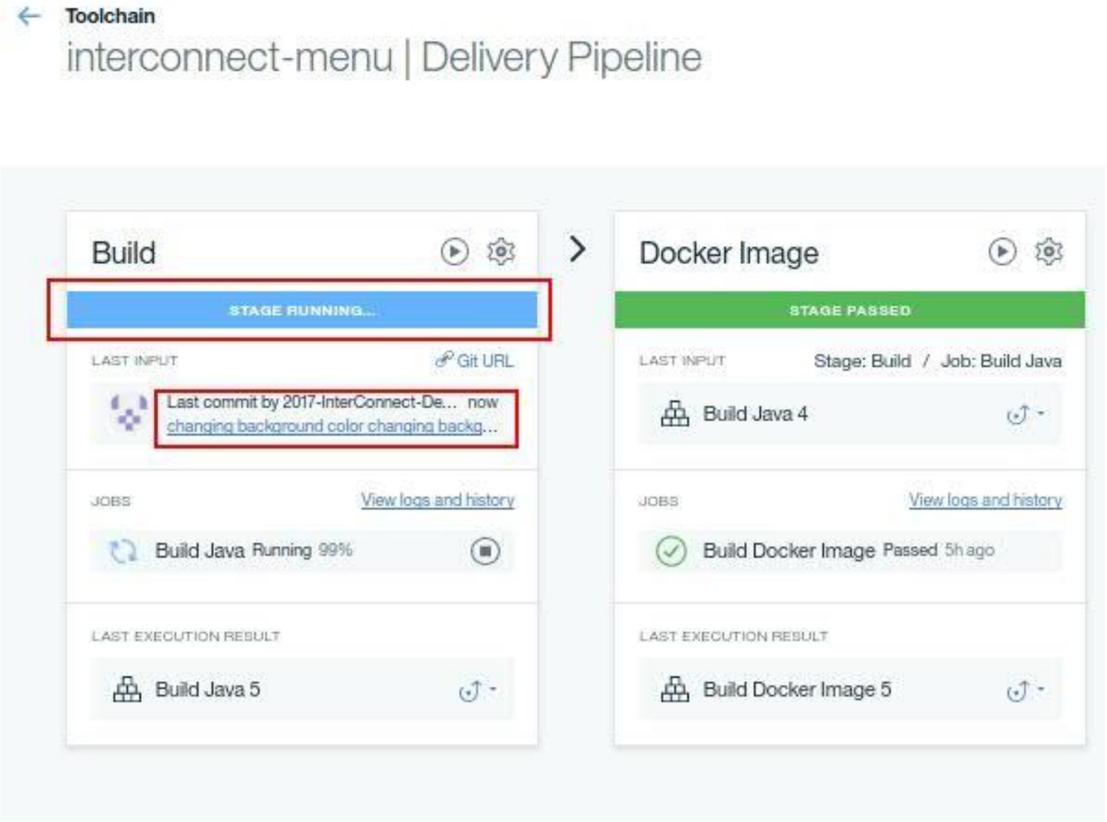
changing background color

changing background color for an end to end CI/CD demo

● Commit directly to the `master` branch.

○ Create a **new branch** for this commit and start a pull request. Learn more about pull requests.

3.5. Right after you commit the changes, your delivery pipeline should automatically get triggered



Toolchain

interconnect-menu | Delivery Pipeline

Build — STAGE RUNNING...

LAST INPUT — Git URL

Last commit by 2017-InterConnect-De... now
changing background color changing backg...

JOBS — View logs and history

Build Java Running 99%

LAST EXECUTION RESULT

Build Java 5

Docker Image — STAGE PASSED

LAST INPUT — Stage: Build / Job: Build Java

Build Java 4

JOBS — View logs and history

Build Docker Image Passed 5h ago

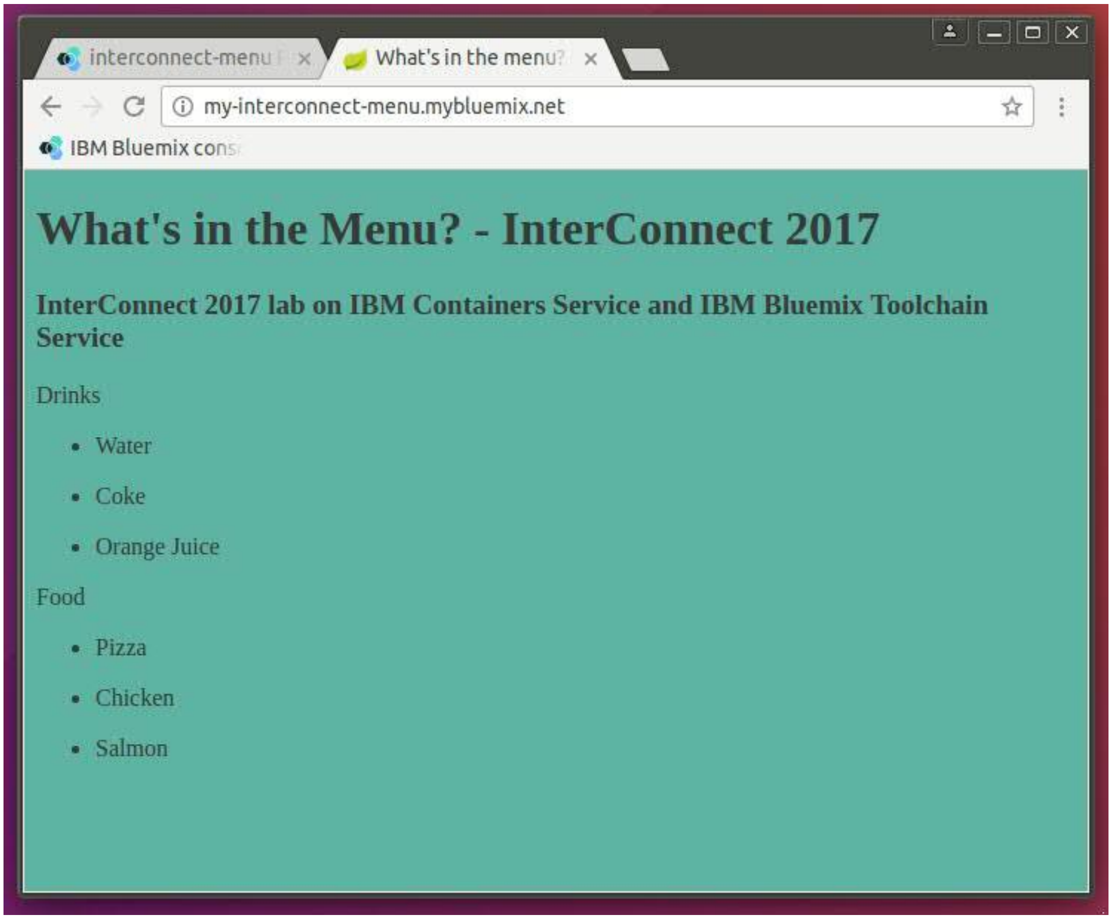
LAST EXECUTION RESULT

Build Docker Image 5

3.6. Open your browser and point it to the public route your defined for the menu app

3.7. Refresh your browser every now and then until the delivery pipeline has successfully finish so that you verify new changes have been made with zero-downtime. That is, every time you refresh the browser the application is working.

3.8. Once the delivery pipeline has successfully finished, the background of your menu app should have changed on a disruptive manner



What's in the Menu? - InterConnect 2017

InterConnect 2017 lab on IBM Containers Service and IBM Bluemix Toolchain Service

Drinks

- Water
- Coke
- Orange Juice

Food

- Pizza
- Chicken
- Salmon

References

Spring docs
Spring Boot docs
GitHub docs
Docker docs
Dockerfile ref
IBM Bluemix
IBM Bluemix Container Service docs
IBM Bluemix Active Deploy docs
DevOps and toolchains
IBM Bluemix Delivery Pipeline docs

External material

IBM Cloud Architecture Center
IBM Bluemix Garage Method
IBM Microservices TV
Meeting Microservices Blog
Microservices, meet DevOps Blog