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.**

© Copyright IBM Corporation 2017

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

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 4

Pre-requisites 5

Introduction 6

Java Application 7

Docker and Containers 13

IBM Bluemix Container Service 18

IBM Bluemix Continuous Delivery Service 26

IBM Bluemix Active Deploy Service 47

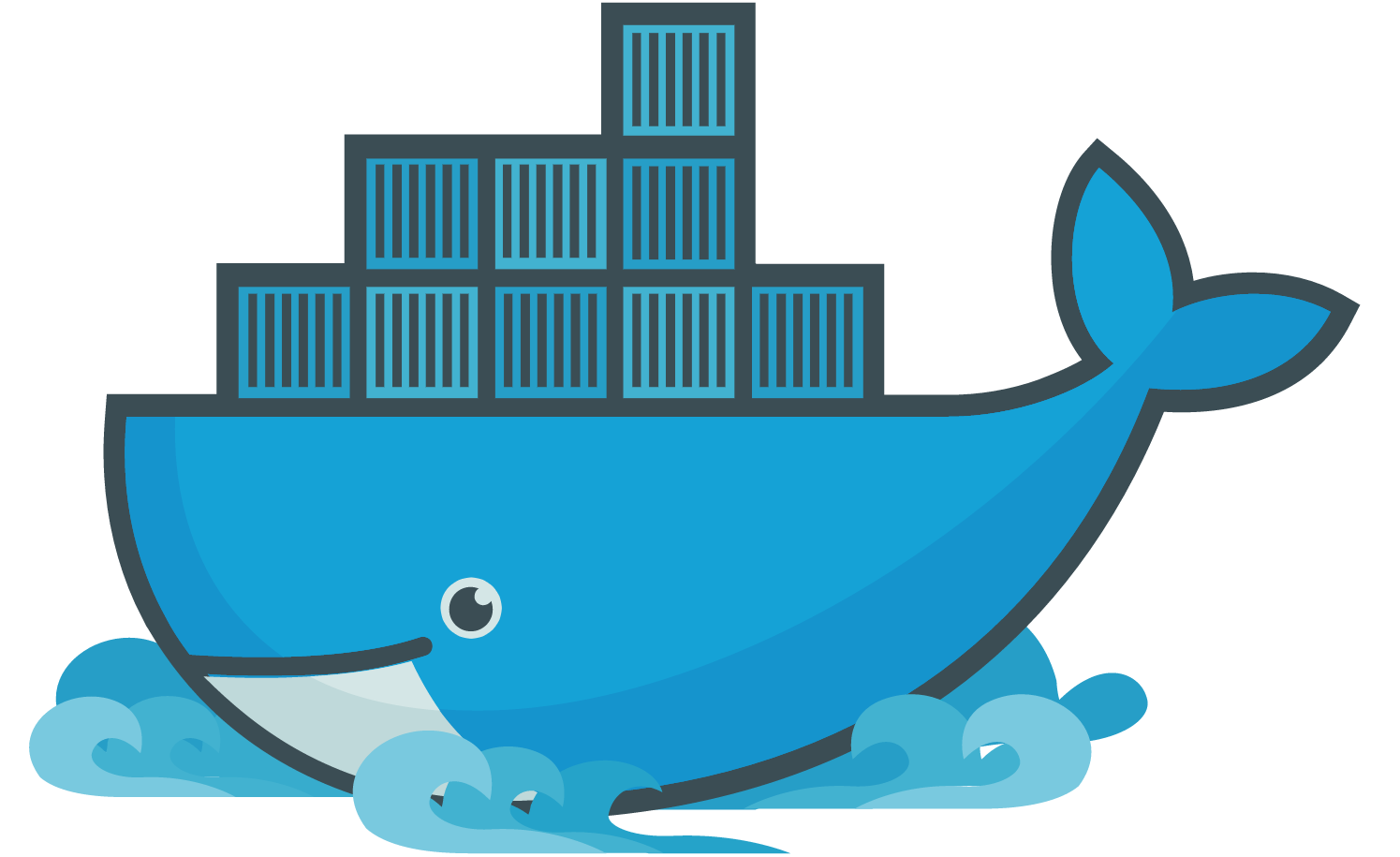
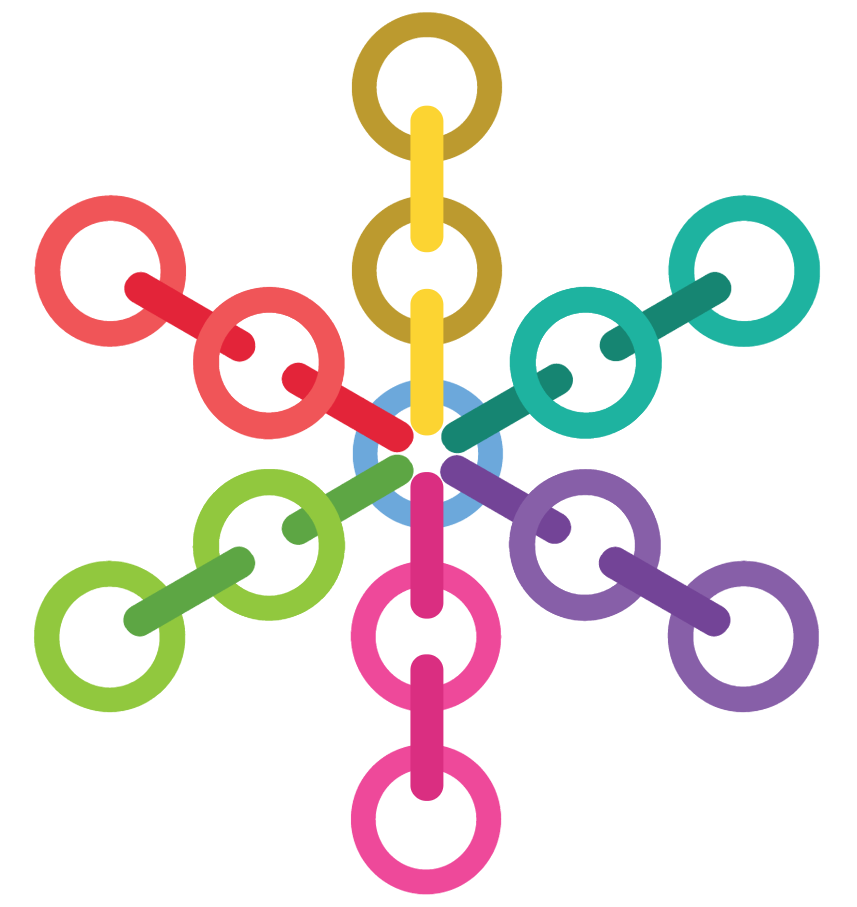
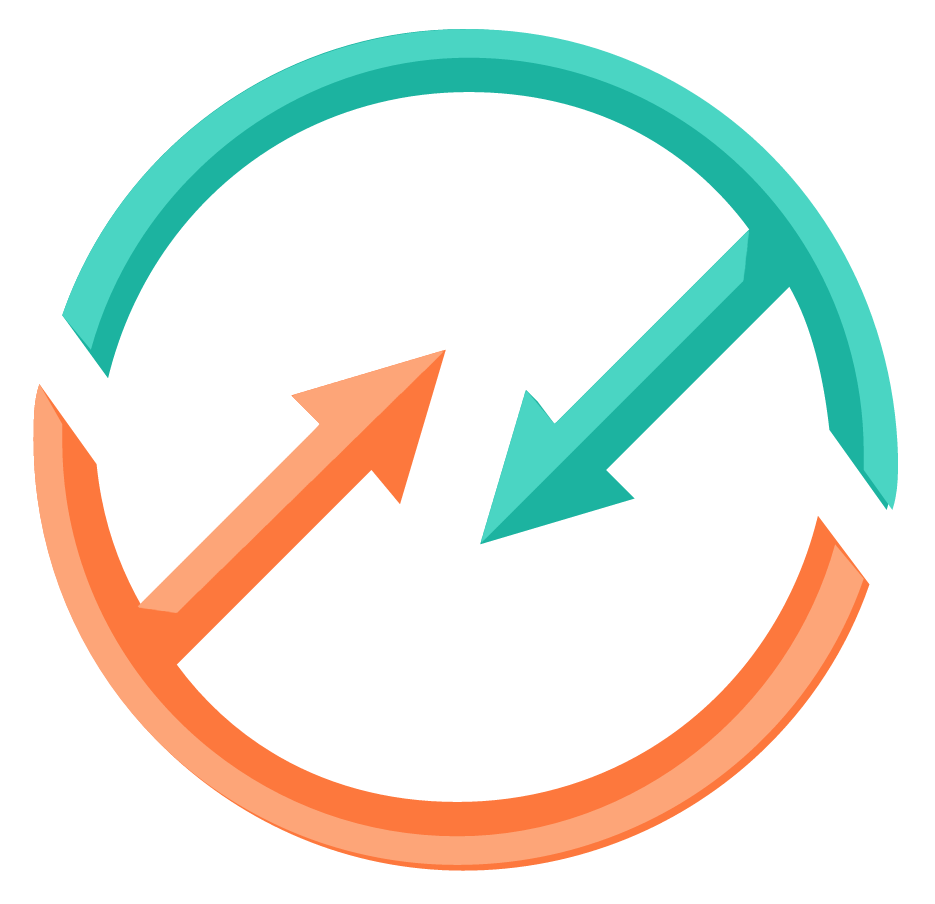
References 56

External material 56

# 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

This lab has been designed to be carried out on a UNIX-like system. Before getting started we must ensure we have installed the following tools in our system:

* **Java** (click [here](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)))
* **Maven** (click [here](https://maven.apache.org/install.html)))
* **Git** (click[here](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git))
* **Docker** (click [here](https://docs.docker.com/engine/installation/)))
* **Cloud Foundry CLI** (click [here](https://github.com/cloudfoundry/cli/releases)))
* **IBM Containers Cloud Foundry CLI plugin**: (click [here](https://console.ng.bluemix.net/docs/cli/index.html#cli))

Run the following commands in your terminal:

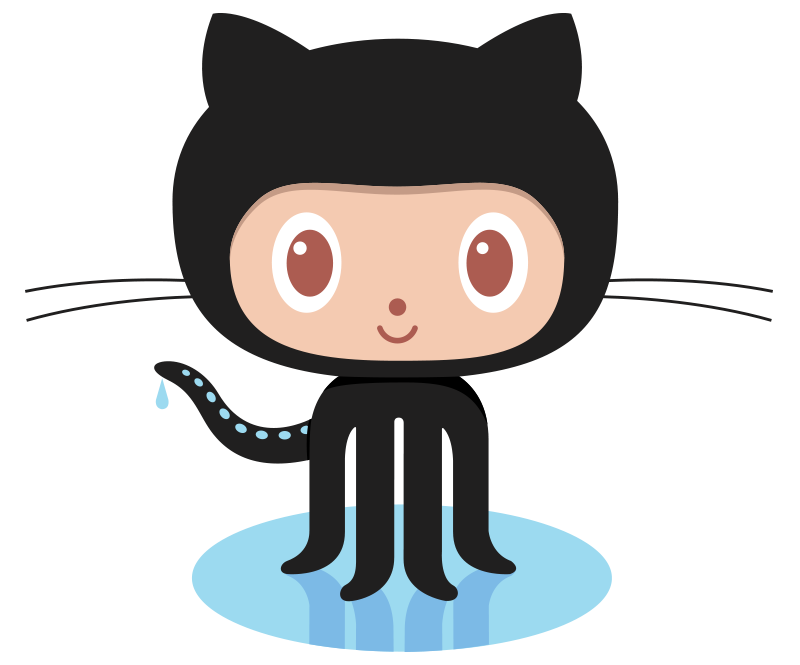
$> *cf add-plugin-repo bluemix-cf https://plugins.ng.bluemix.net*

$> *cf install-plugin IBM-Containers -r bluemix-cf*

We will need a Bluemix and GitHub account:

* **Bluemix account** - <https://console.ng.bluemix.net/registration/>  
  (Create space in US South region)
* **GitHub account** - <https://github.com/join?source=header-home>

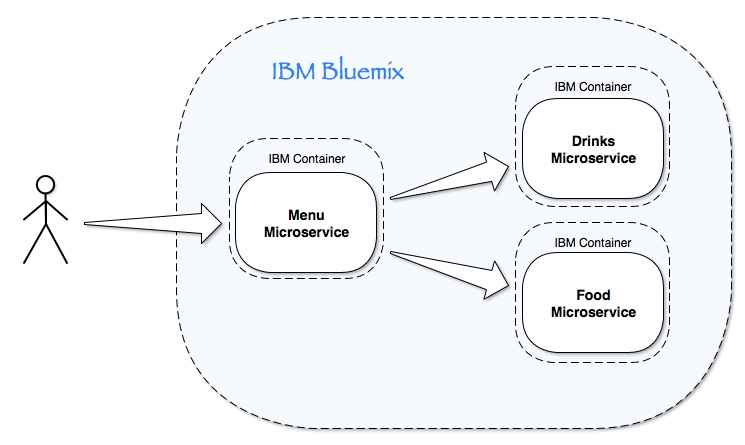
Finally, we need to create a workspace (a simple linux folder) which we are going to use to work with the code and where we are going to place the util scripts needed to complete the lab. In order to get the util scripts, go to the GitHub repo for this lab ([here](https://github.com/jesusmah/2720-InterConnect-2017)), download the *scripts.zip* file and unzip it within your workspace.

# 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](https://spring.io/) 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](https://projects.spring.io/spring-boot/#quick-start), Spring Boot helps you get things done faster:

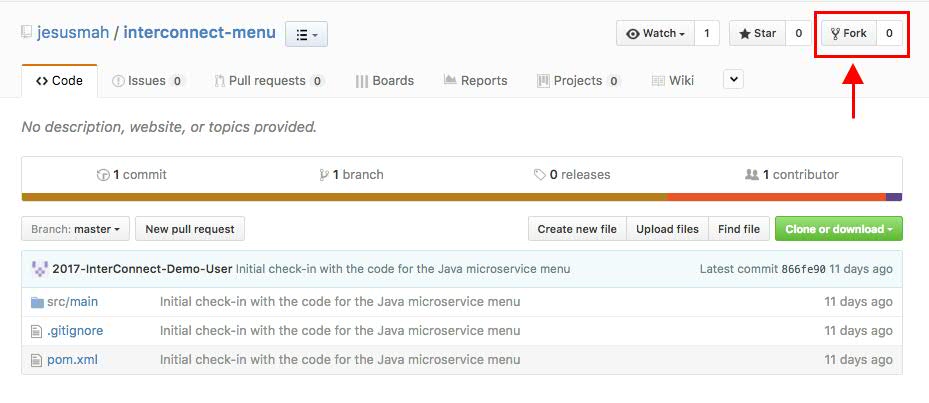
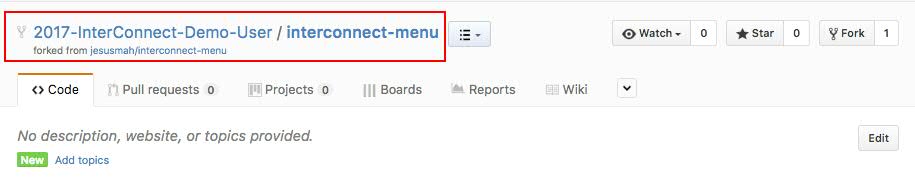
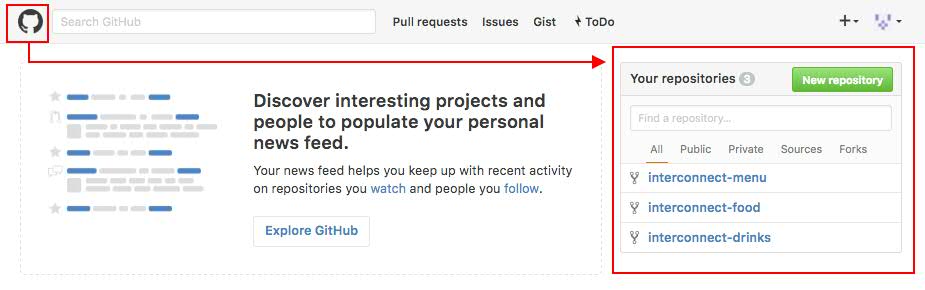
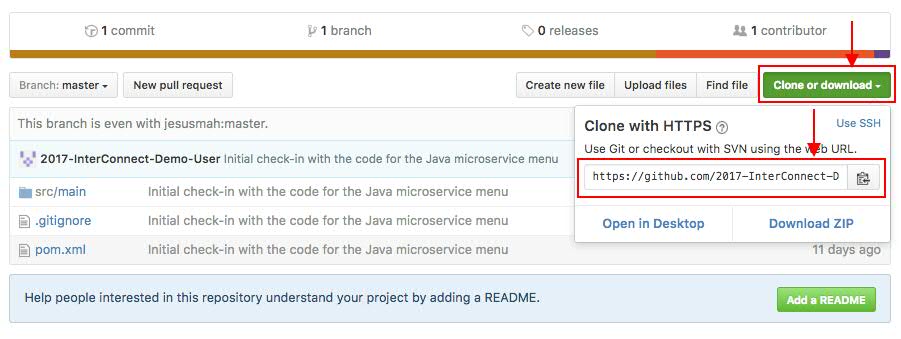
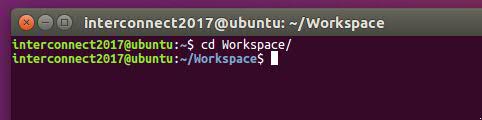
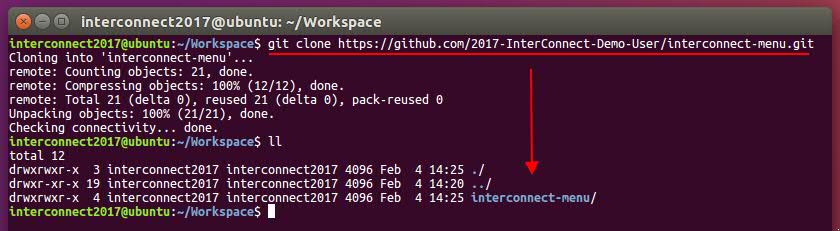
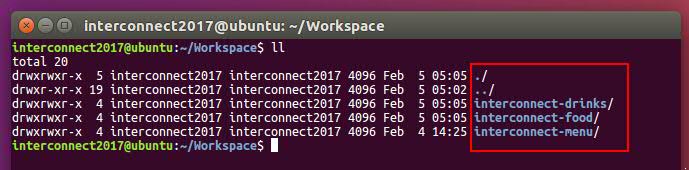
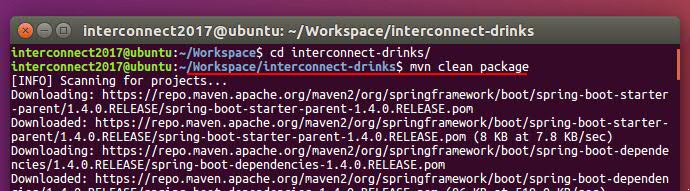
* simplifies your Spring dependencies, no more version collisions
* can be run straight from a command line without an application container
* build more with less code - no need for XML, not even web.xml, auto-configuration
* useful tools for running in production, database initialization, environment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

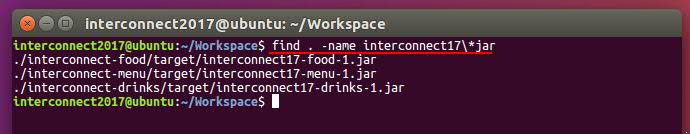
GitHub repositories for the Java microservices:

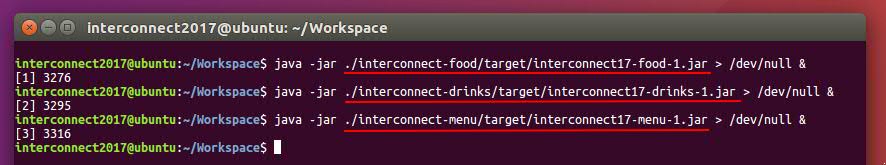
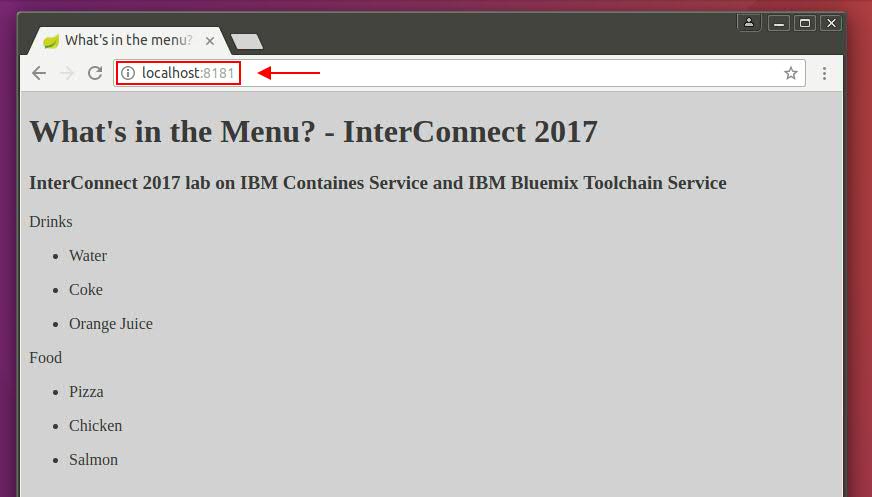
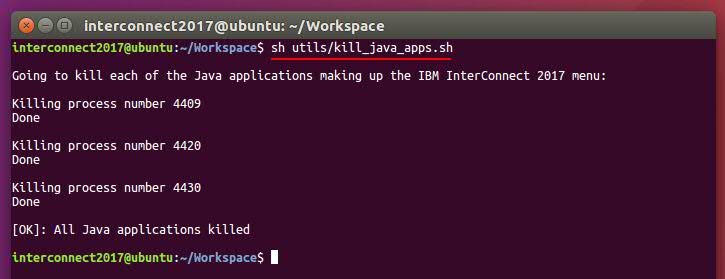
* **Menu** microservice: <https://github.com/jesusmah/interconnect-menu>
* **Drinks** microservice: <https://github.com/jesusmah/interconnect-drinks>
* **Food** microservice: <https://github.com/jesusmah/interconnect-food>

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

1. Fork GitHub repository
   1. Open the GitHub repo by clicking the corresponding link above.
   2. Click on the Fork button on the top right corner of the browser.  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      You should now see the Java project in your GitHub account:
   3. **Repeat previous steps for the other two GitHub repositories**.After forking the three GitHub repository, your GitHub repository should look like:
2. Clone or download the code to your workstation
   1. Click on the Clone or download button and copy the address
   2. Open a terminal and change directory to your workspace
   3. Clone or download the code from your GitHub repository by executing  
        
      $ git clone *your-repository-url*
   4. **Repeat previous steps for the other two GitHub repositories.**Your workspace should now look like:
3. Build Java applications
   1. Change directory into one of the three projects
   2. Build the Java application by executing  
        
      $ mvn clean package

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

* 1. **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:

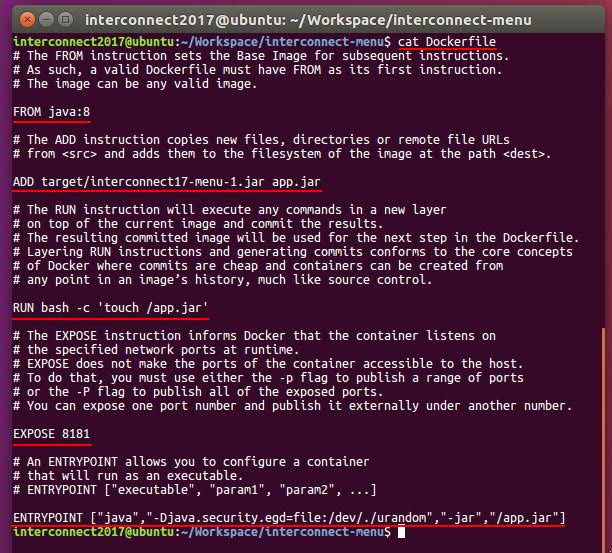
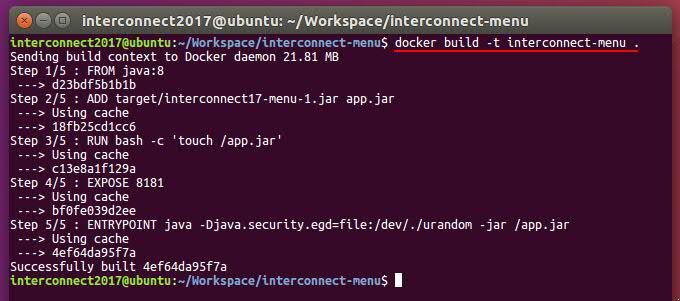
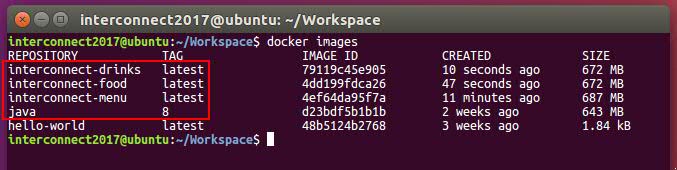
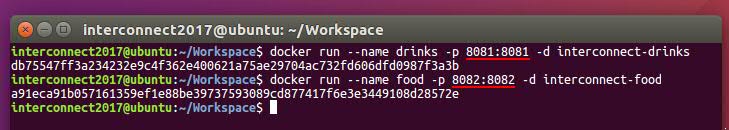
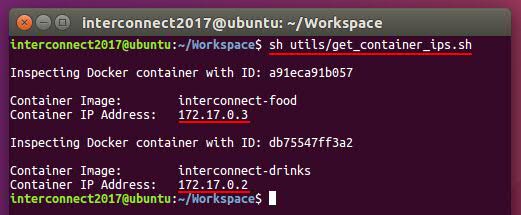
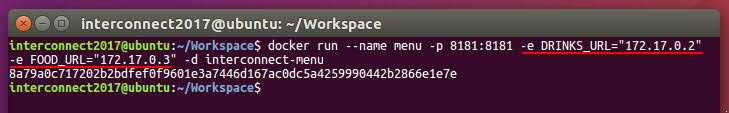
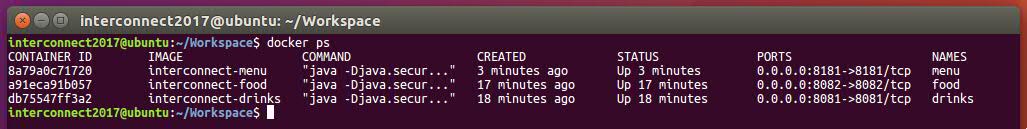
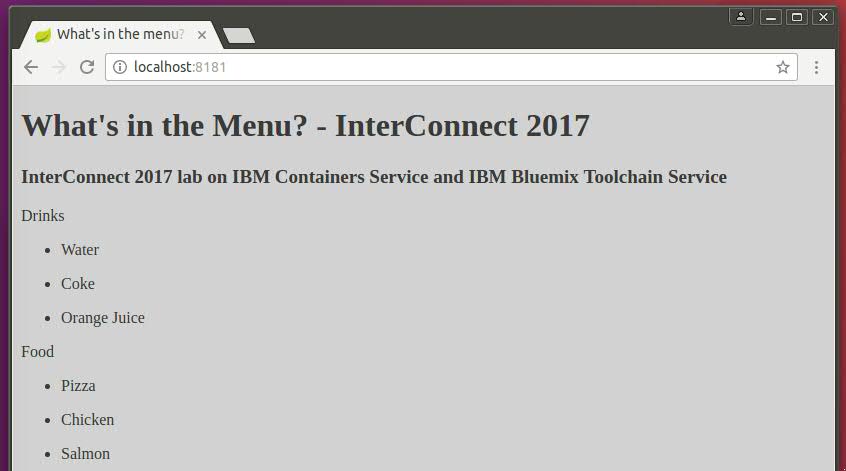
1. Execute the 2017 IBM InterConnect menu Java app
   1. Execute each of the three jar files in the background with their standard output redirected to */dev/null* by issuing the following command:  
        
      $ java –jar *jar-file* > /dev/null &
   2. Check the 2017 IBM InterConnect menu application is working by pointing your browser to *localhost:8181*
2. Kill the local 2017 IBM InterConnect menu Java application
   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

# Docker and Containers

[**Docker**](https://www.docker.com/what-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.

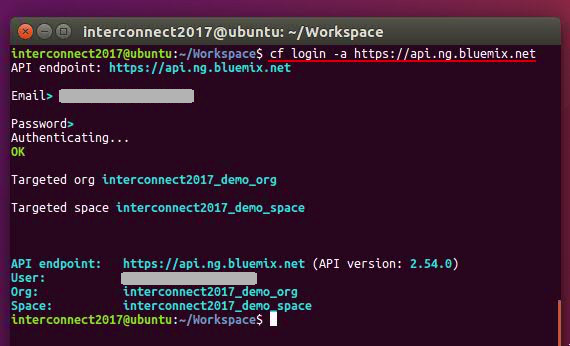
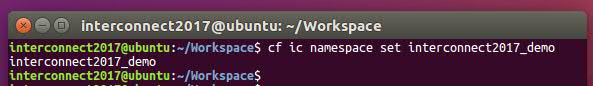
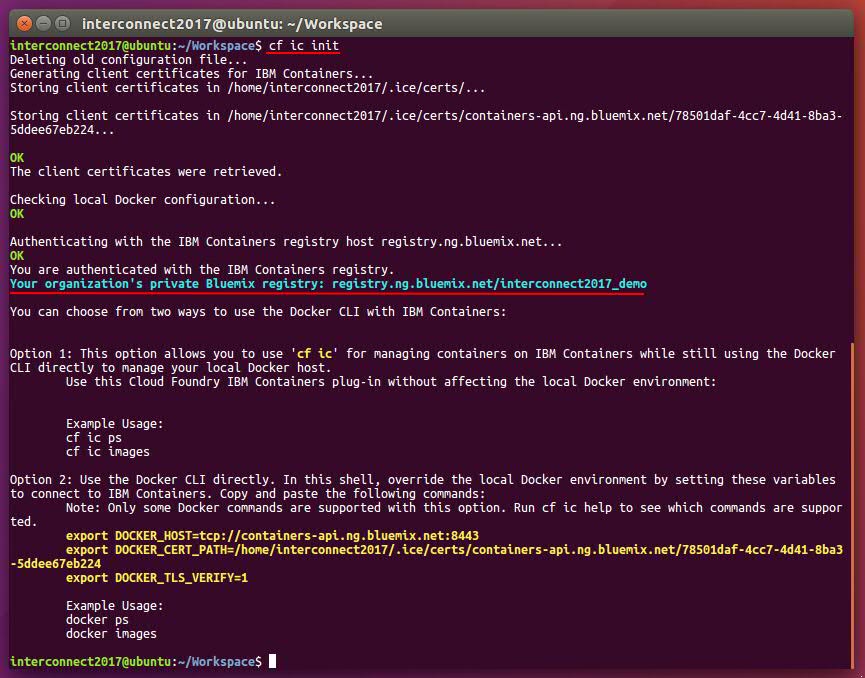
In this section, we will go through the process of containerizing our three Java applications by using a [Dockerfile](https://docs.docker.com/engine/reference/builder/). Later, we will create the Docker images for the three Java apps from their Dockerfiles to finally run the 2017 IBM InterConnect dockerized menu app locally in our workstations.

1. Containerize the Java applications
   1. Verify and understand Dockerfiles in each of the projects.  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      
   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  
        
        
        
      
   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  
        
        
        
        
        
        
        
        
        
        
      
2. Run and test the dockerized/containerized 2017 IBM InterConnect menu Java app
   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>*
   2. Get drinks and food containers IPs by executing the *get\_container\_ips.sh* located within the *utils* folder
   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.*
   4. At this point, if you execute *docker ps* you should see the following
   5. Check the dockerized/containerized 2017 IBM InterConnect menu application is working by pointing your browser to *localhost:8181*

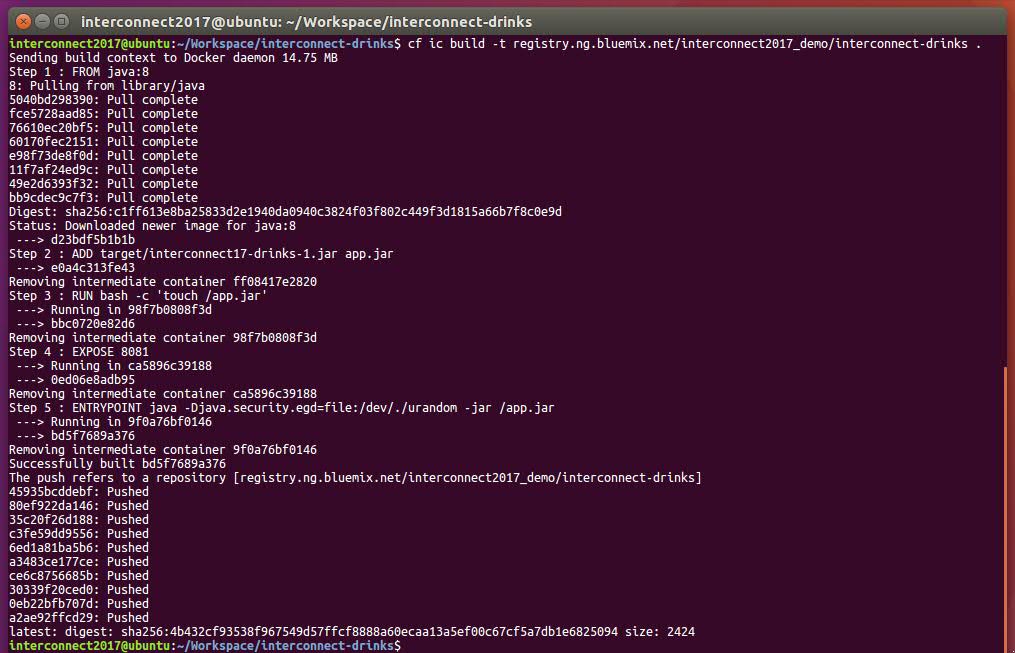
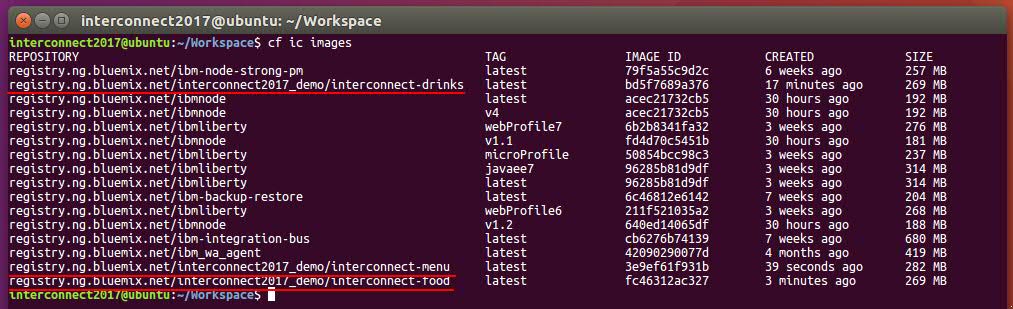
# IBM Bluemix Container Service

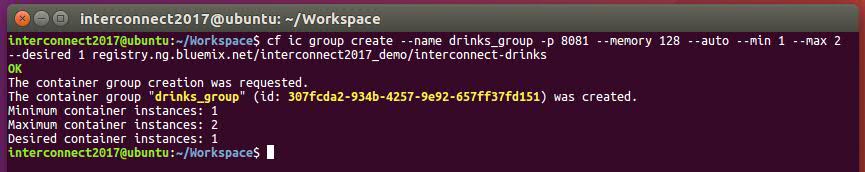
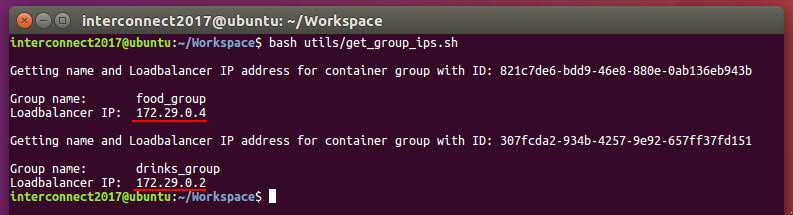
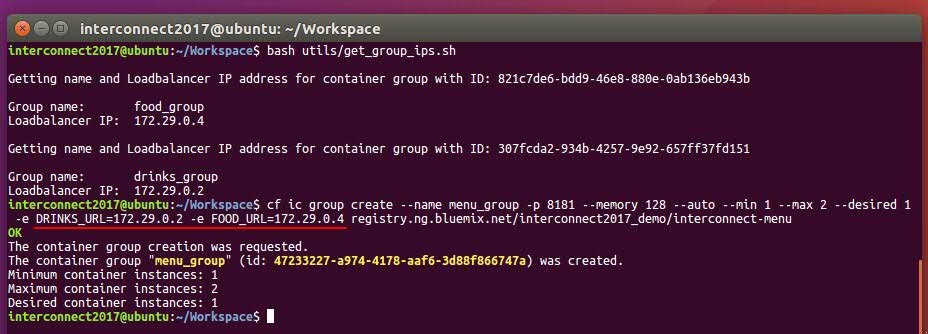
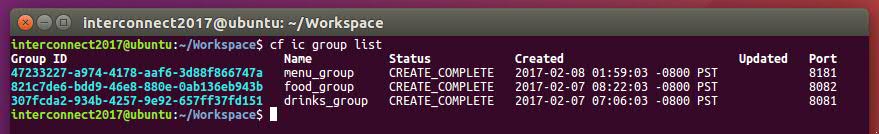
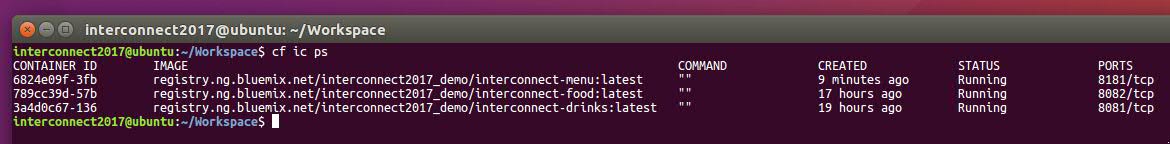
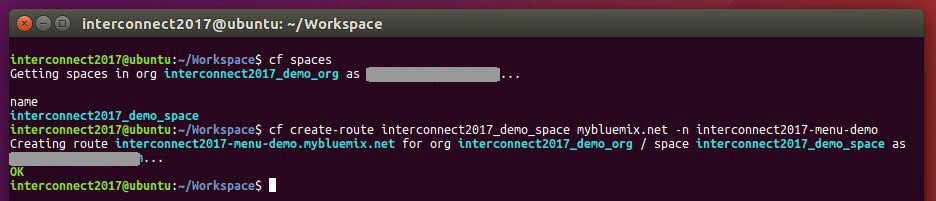
[**IBM Bluemix Container Service**](https://www.ibm.com/cloud-computing/bluemix/containers)adapts Docker technology to enterprise application development,extending the benefits of resource allocation and isolation across public and private deployments, providing a private registry for your Docker images and the tools necessary to manage the entire life cycle of containerized applications you deploy in the cloud.

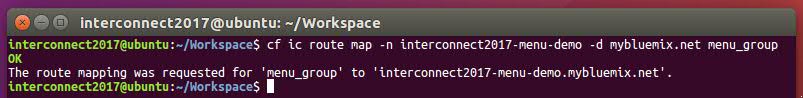
In this section, we will go through the same process of containerizing our three Java applications by using a [Dockerfile](https://docs.docker.com/engine/reference/builder/) but remotely in [IBM Bluemix](https://console.ng.bluemix.net/). We will now have our Docker images remotely in our private Docker 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.

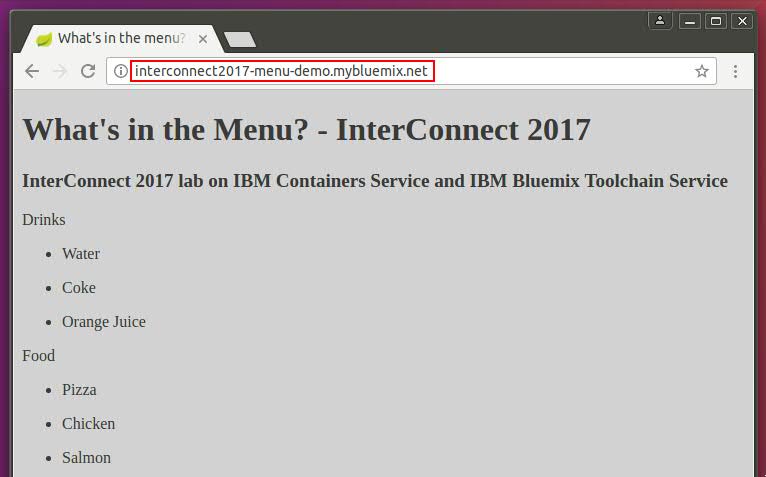
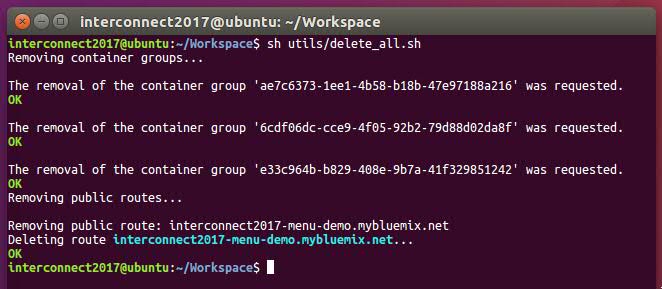
1. Open a terminal and log into IBM Bluemix by using your IBM Bluemix credentials (pre-requisite section) by executing  
     
   $ cf login –a *<API\_endpoint>*  
   Important: the API endpoint changes based upon the IBM Bluemix space region you want to work with. In this lab, we assume people will create a space in the US South region and work with it.   
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   
2. Create a namespace for your private Docker registry by executing  
     
   $ cf ic namespace set <*namespace\_id*>  
     
     
   
3. Initialize the IBM Bluemix Container Service CLI by executing cf ic init  
     
     
   
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  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
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  
  
  
  
  
  
  
  


1. 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
2. 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  
     
     
     
     
     
     
     
     
     
     
   
3. 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>*
4. Check the three container groups have been created by executing  
     
   $ cf ic group list
5. Check three containers have been created by executing  
     
   $ cf ic ps
6. 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.*

1. 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>*

1. 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
2.  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

# IBM Bluemix Continuous Delivery Service

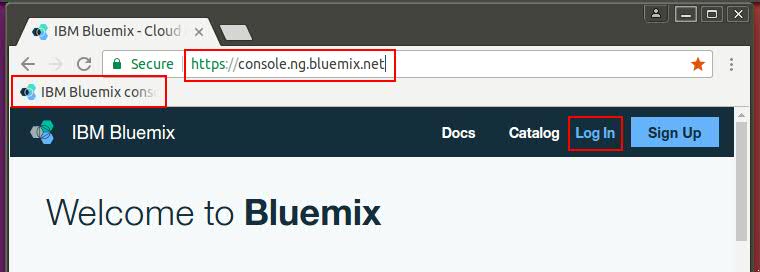
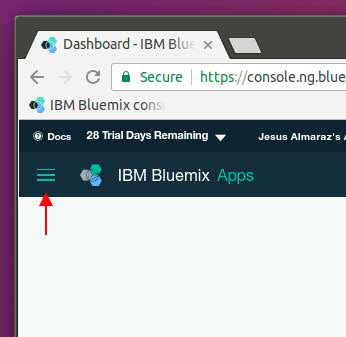
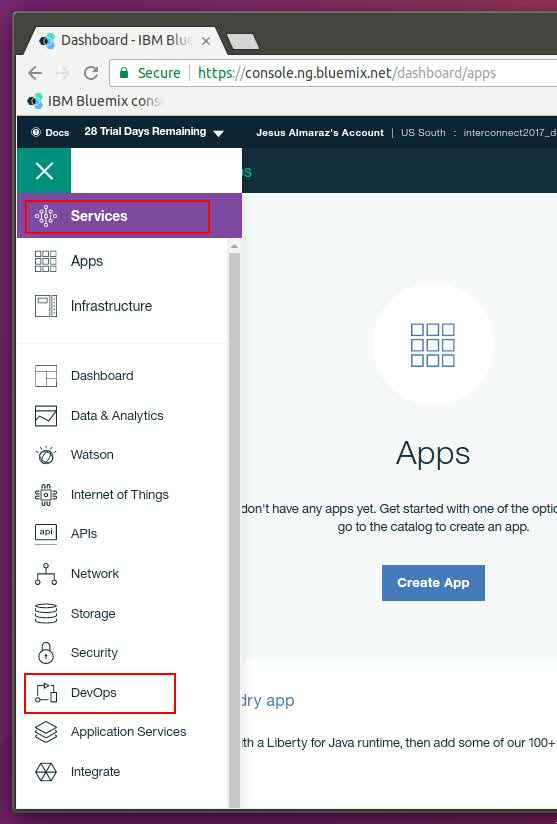
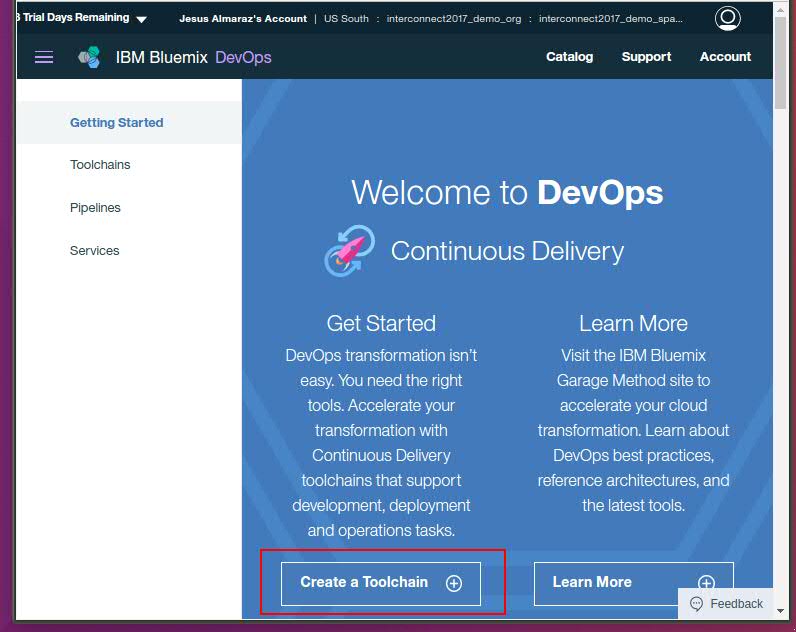
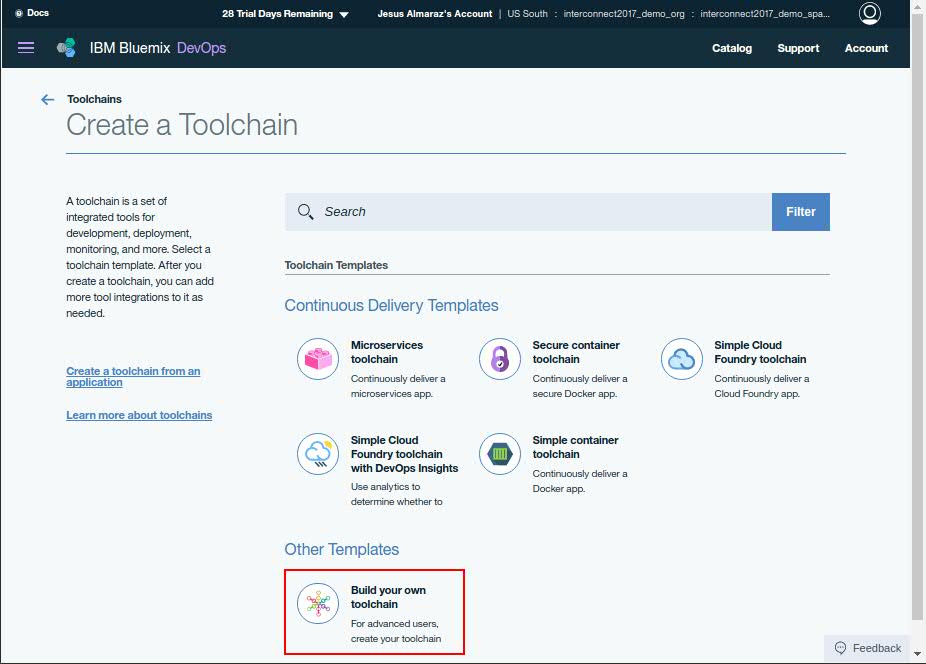
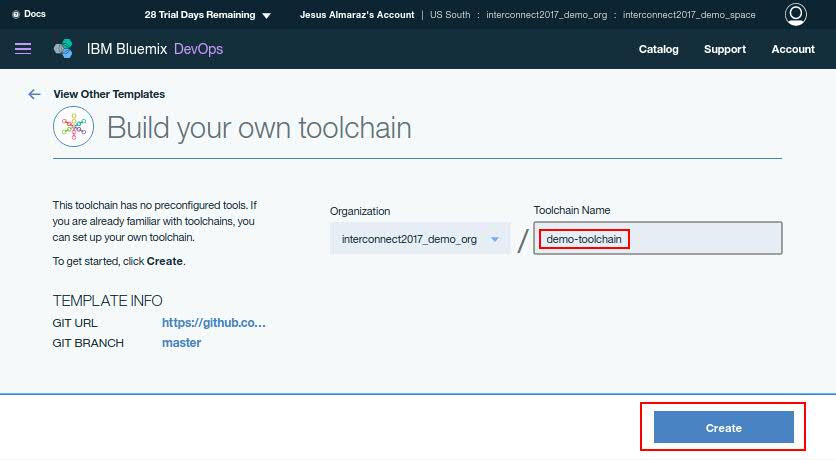
Use Continuous Delivery to automate builds, unit tests, deployments, and more. Edit and push code through the rich web based IDE. Create toolchains to enable tool integrations that support your development, deployment, and operation tasks.

A [toolchain](https://www.ibm.com/devops/method/category/tools/) 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.

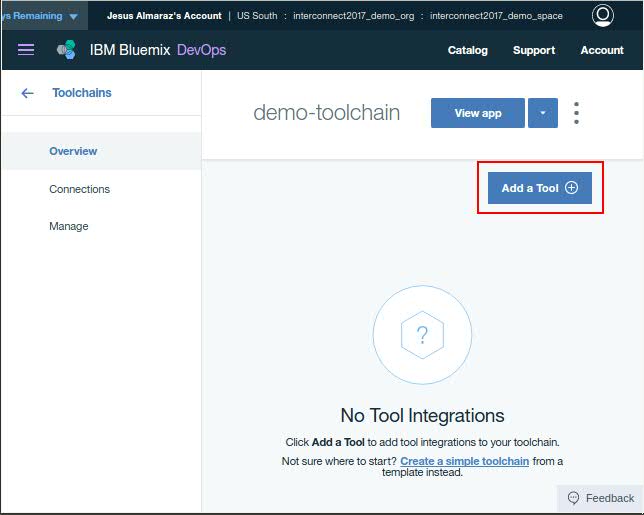
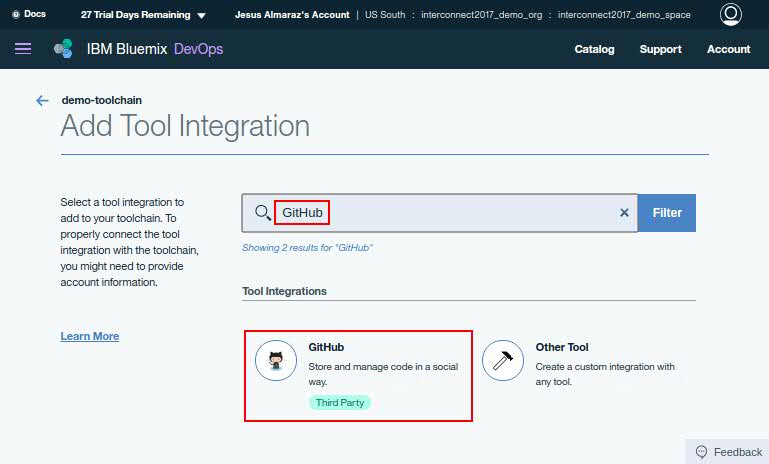
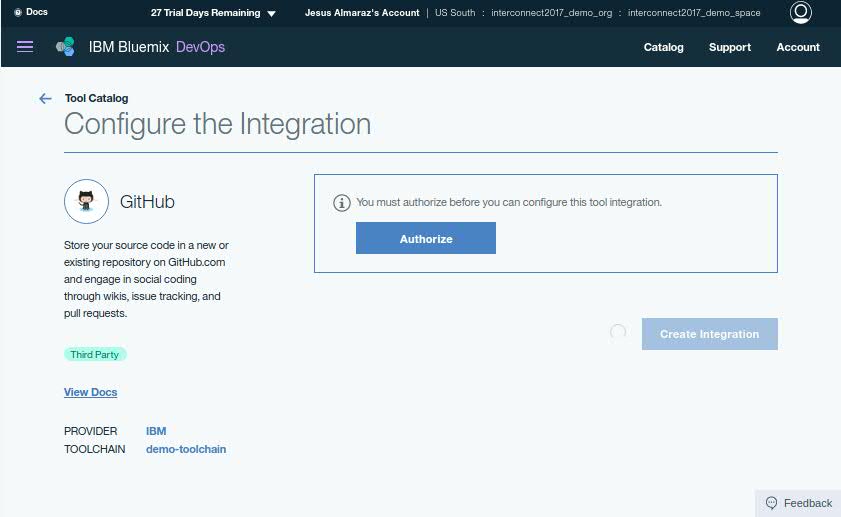
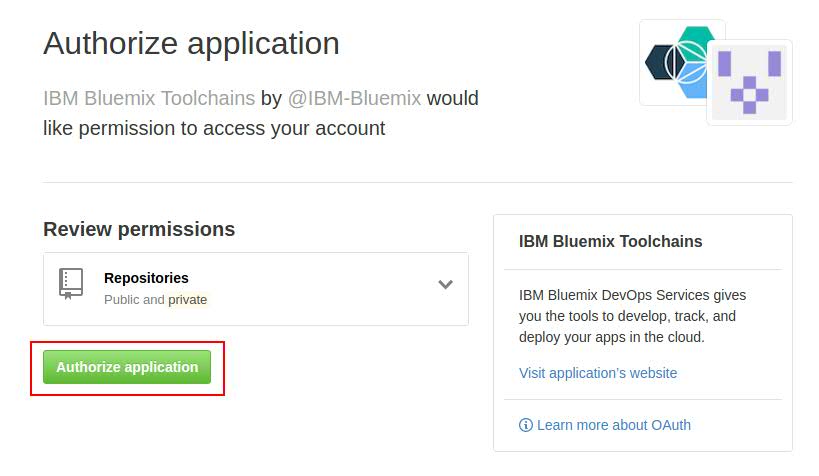
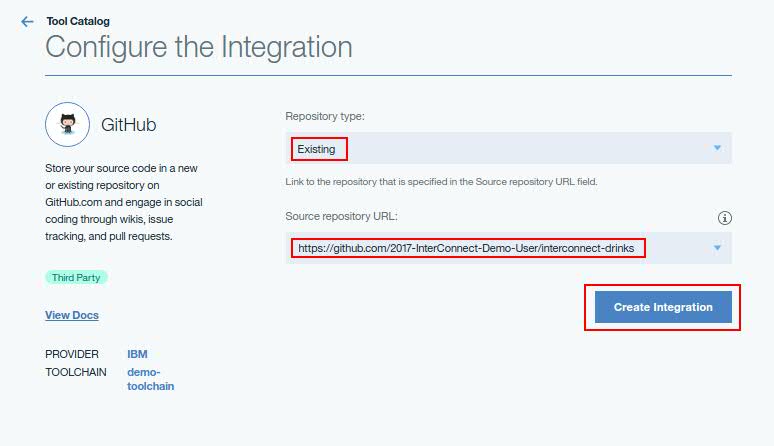
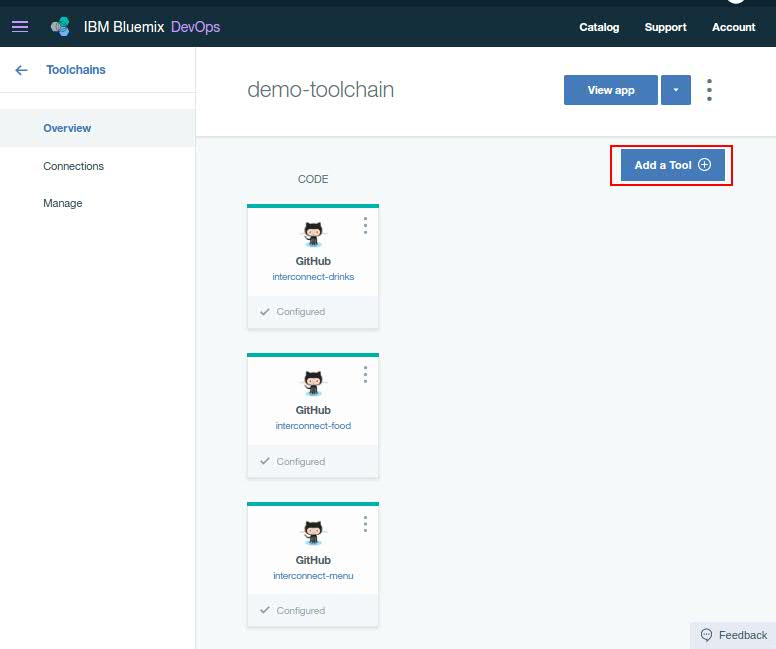
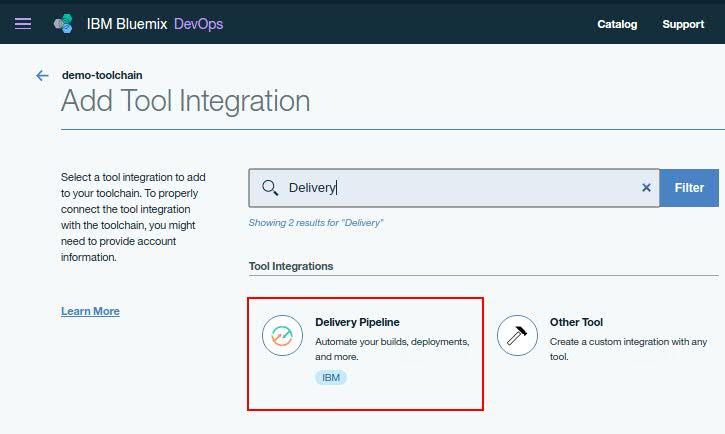
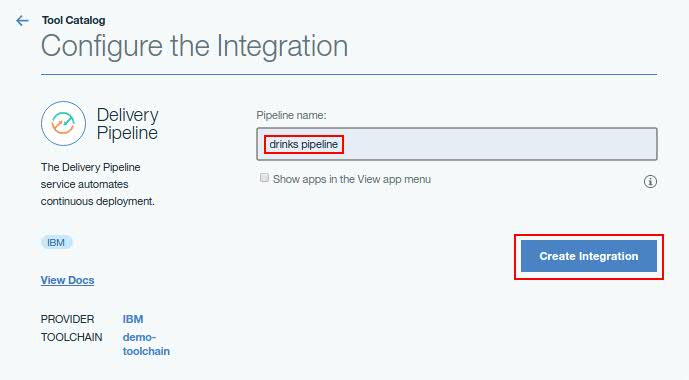
Tools

* 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.

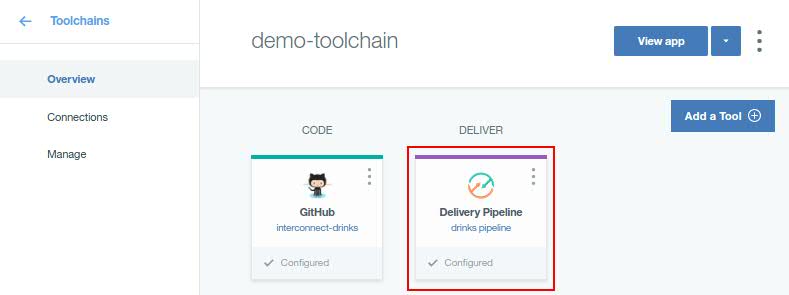
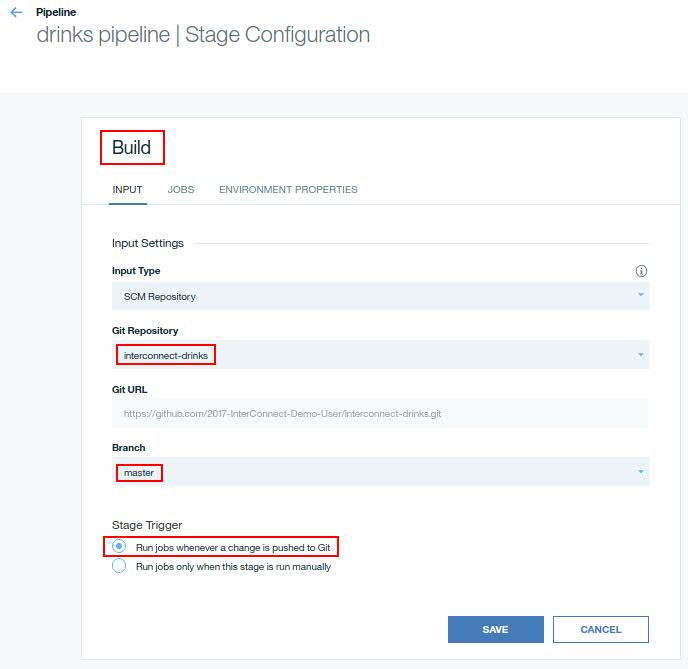
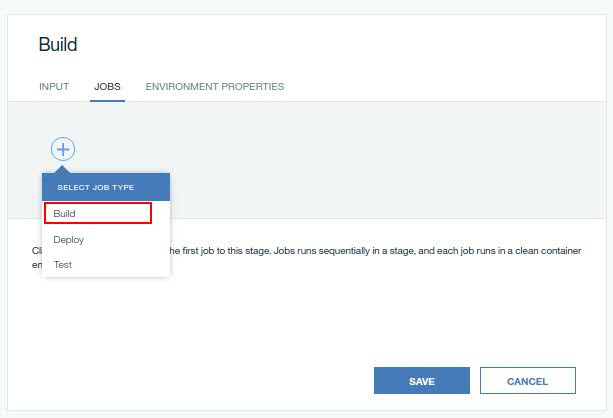
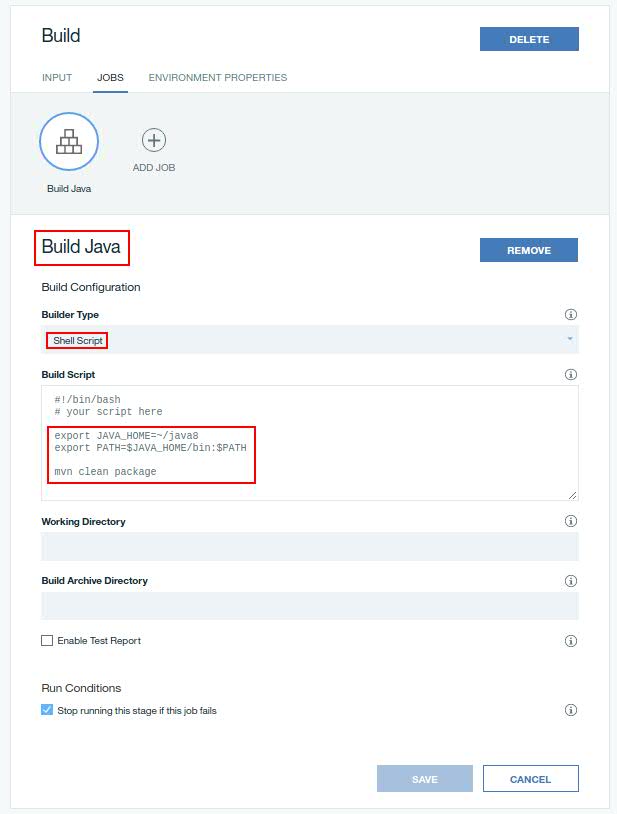
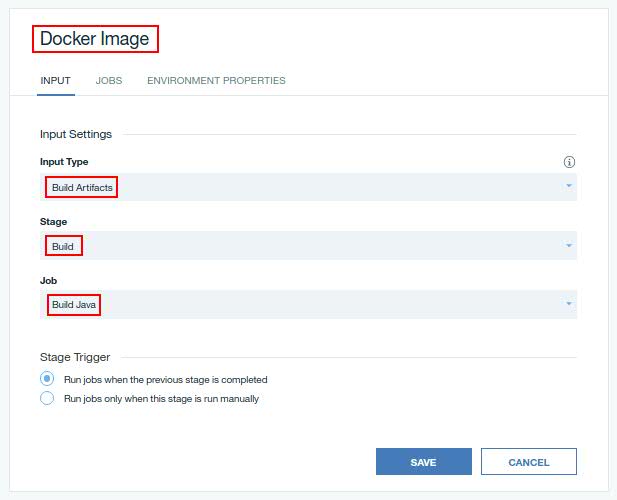
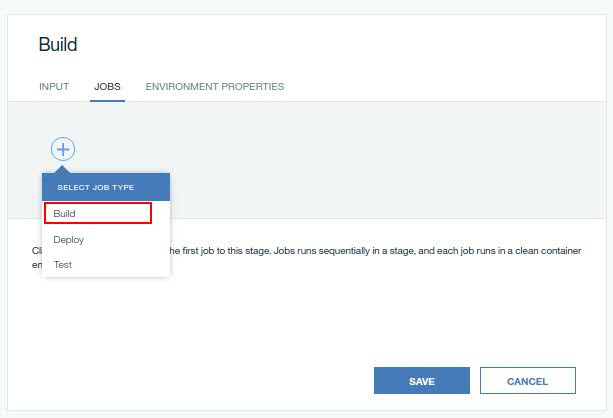
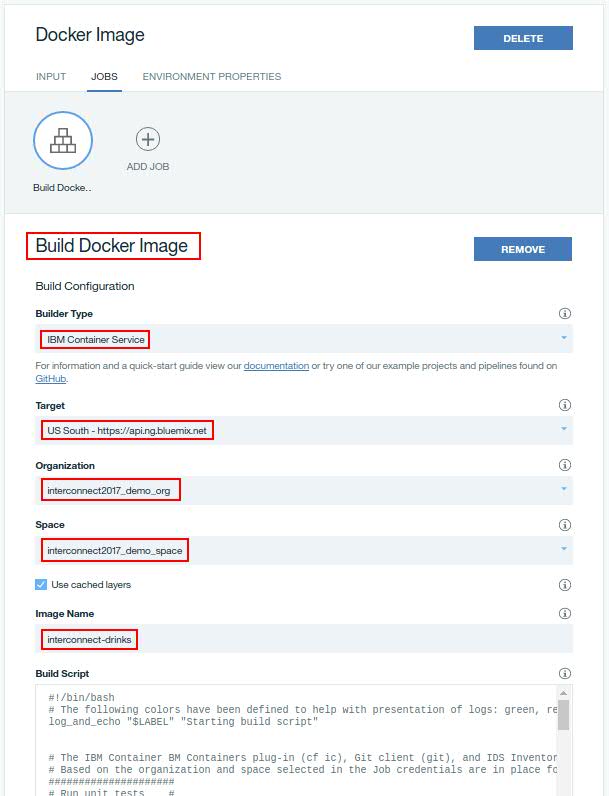
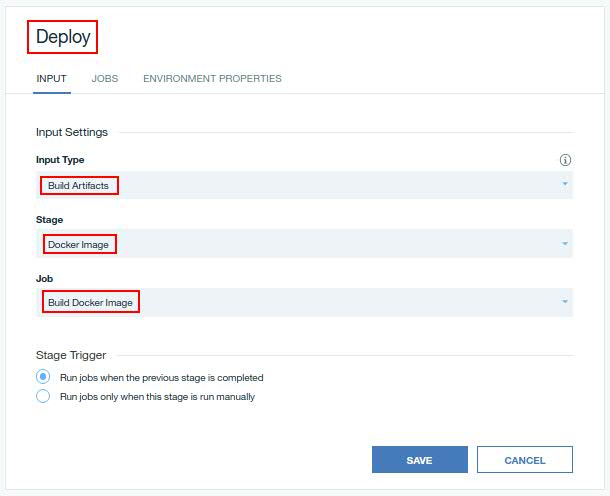
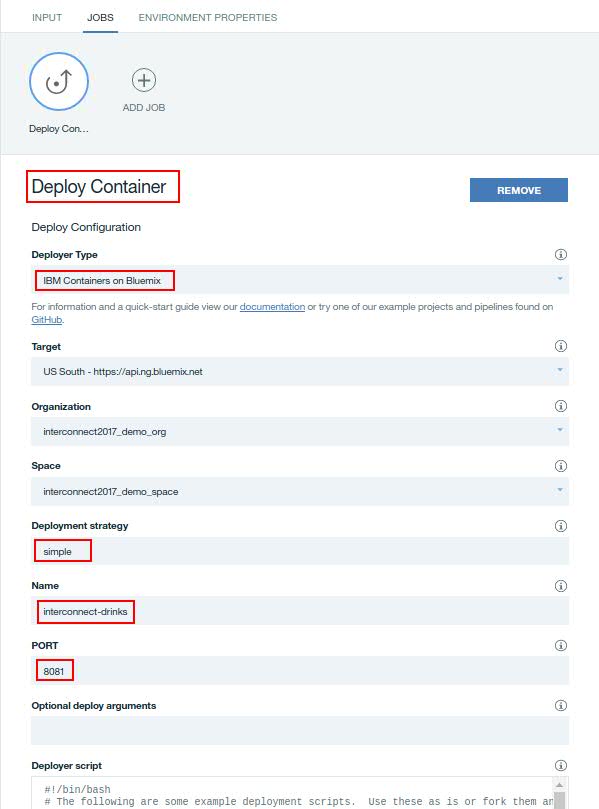
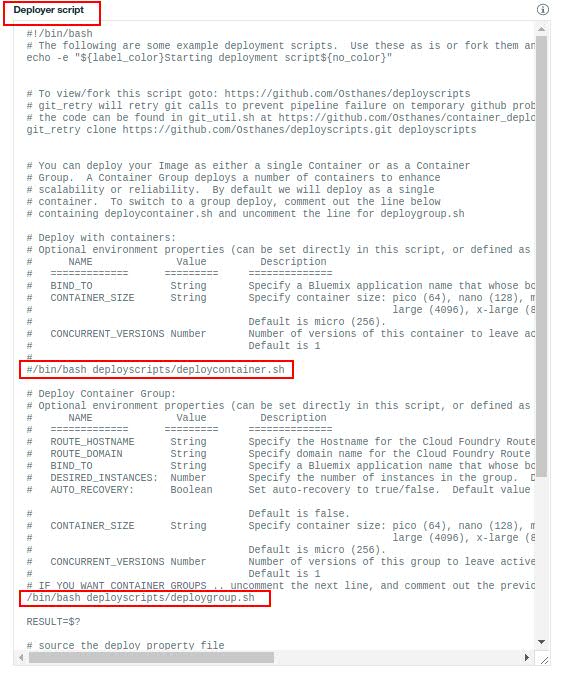
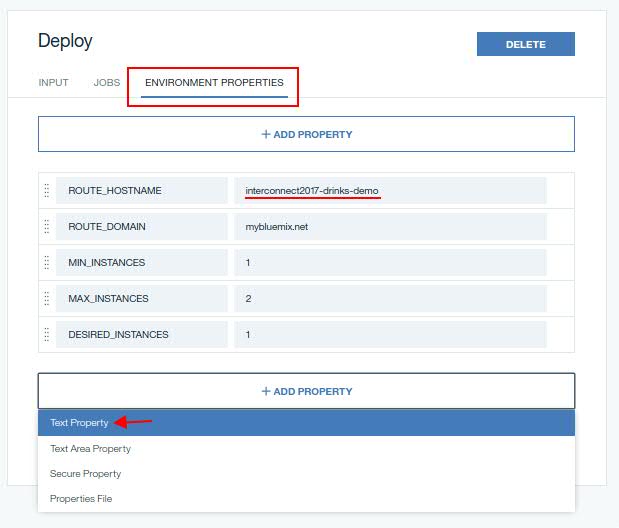
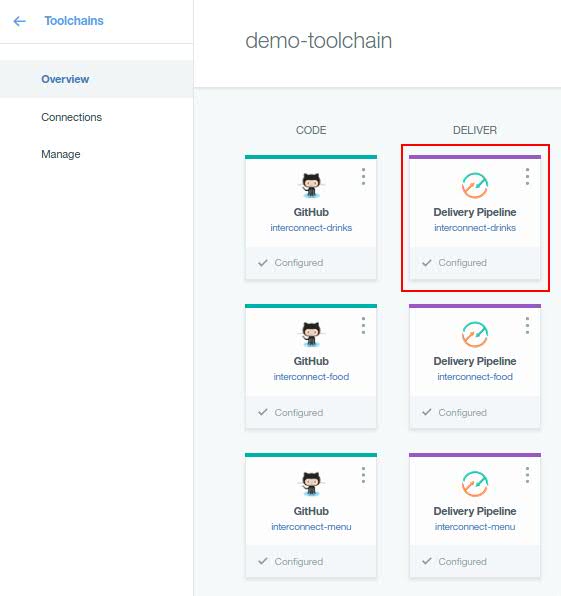
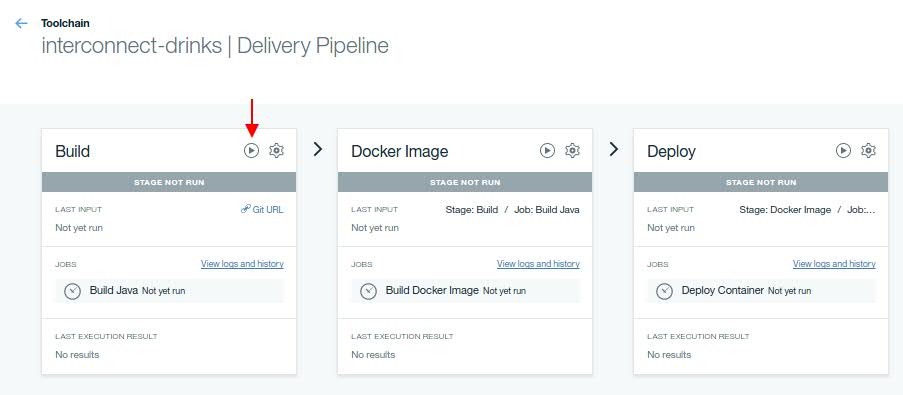
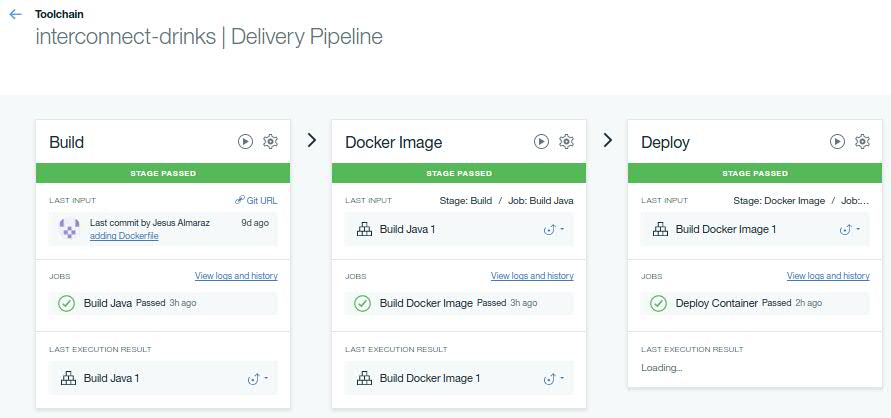
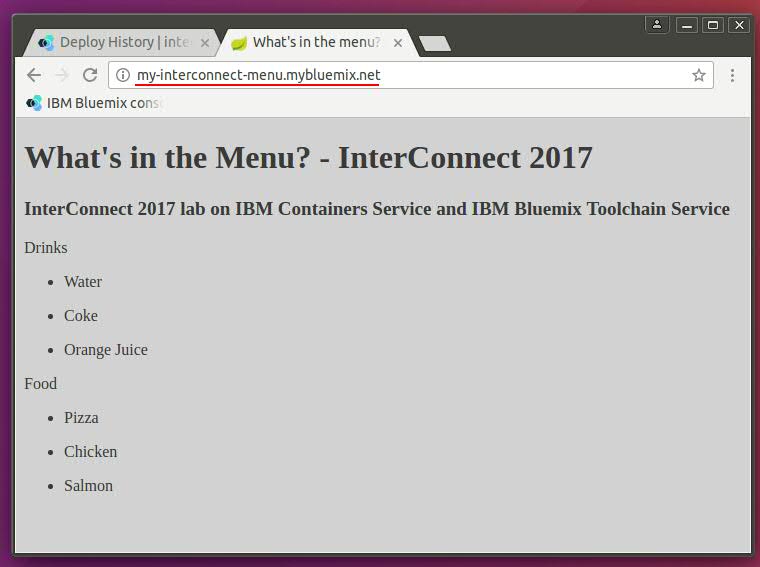
In this section, we will create a toolchain using the IBM Bluemix Continuous Delivery Service to automate the build and deployment to the IBM Bluemix public cloud of the 2017 InterConnect menu app. We will go through the creation and configuration of the GitHub and Delivery Pipeline tools for each of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

1. Create a toolchain from the IBM Bluemix Continuous Delivery Service
   1. Open your Bluemix console in your web browser  
        
        
        
        
        
        
        
        
        
        
        
        
        
      
   2. Open the left hand side menu  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      
   3. Click on *Services* and then on the *DevOps* subsection of it  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      
   4. Click on *Create a Toolchain*  
        
      
   5. Click on *Build your own toolchain*
   6. Make sure you select your organization, give an appropriate name to your toolchain and click on *Create*  
        
      

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.

1. Create a GitHub tool for each of the three microservices
   1. Click on *Add a Tool*
   2. Search for GitHub and click on it  
        
      
   3. You need to authorize your toolchain’s access to your GitHub account
   4. Enter your GitHub credentials and click on *Authorize application*  
      
   5. Select *Existing* for *Repository Type* and select one of the three Java app Github repositories. Then, click on *Create Integration*  
        
      
   6. **Repeat previous steps to create a GitHub tool for each of the three Java apps GitHub repositories**
2. Create a Delivery Pipeline tool for each of the three Java apps
   1. Click on *Add a Tool*  
        
      
   2. Search for Delivery Pipeline and click on it
   3. Give an appropriate name to your delivery pipeline and click on *Create Integration*  
        
      
   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:

1. Build – where we will build the needed artefacts of the Java app from the code stored in its GitHub repository.
2. Create Docker image – where we will create a Docker image containing the built artefacts from step I.
3. Deploy – where we will deploy the Docker image created in step II onto the IBM Bluemix public cloud.
4. Configure the Delivery Pipelines
   1. Click on the Delivery Pipeline tool display on your toolchain dashboard
   2. Create *Build* stage
      1. Click on *Add Stage*
      2. Name your stage as Build and make sure it is configured to use the appropriate Git Repository and Branch  
           
         
      3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Build* for the job type  
           
         
      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*  
           
           
           
           
           
           
           
           
           
           
           
           
           
         
      5. Click on *Save*
   3. Create *Docker Image* stage
      1. Click on *Add Stage*
      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  
           
         
      3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Build* for the job type
      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**
      5. Click on *Save*
   4. Create Deploy stage
      1. Click on *Add Stage*
      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  
           
         
      3. Click on *JOBS* tab. In this tab, click on *ADD JOB* and then on *Deploy*  for the job type
      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.  
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
         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  
         
      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**  
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
           
         **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.*
      6. Click on *Save*
   5. Repeat steps 4.1. to 4.4. for the drinks and food Java apps.
   6. Repeat steps 4.1. to 4.4.3 for the menu Java app
   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 }*"  
        
      
   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!  
   1. Click on the food or drinks delivery pipeline icon in the Toolchain dashboard
   2. Click on the play button on the top right corner of the Build stage, the first stage.  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      *\* 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*
   3. After the delivery pipeline has finished, you should see the three stages in green  
        
        
        
        
        
        
        
        
        
      
   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. 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  
        
        
        
        
        
        
        
        
        
        
      

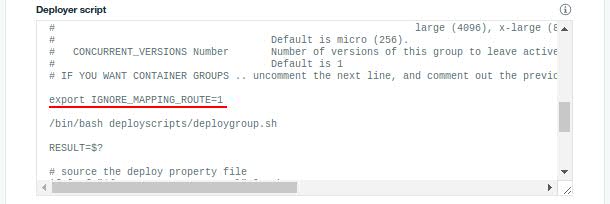
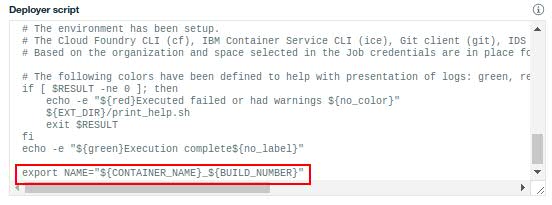
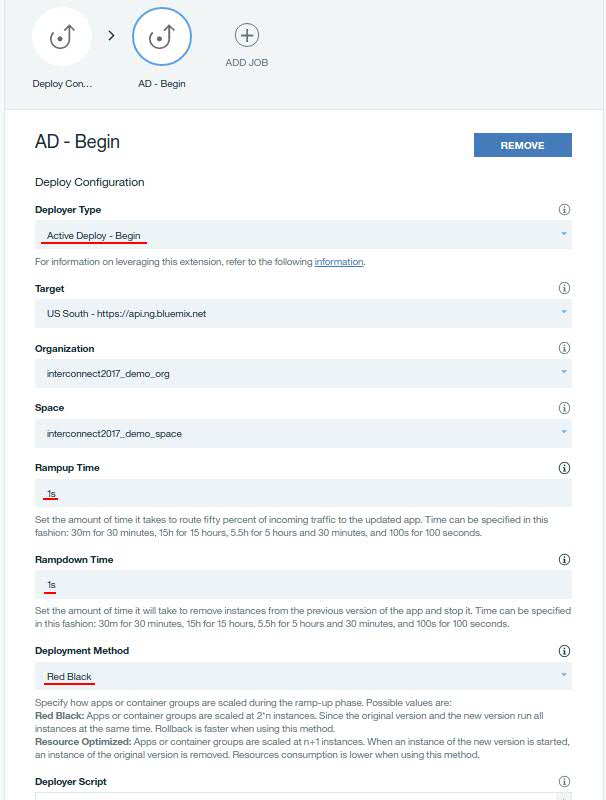
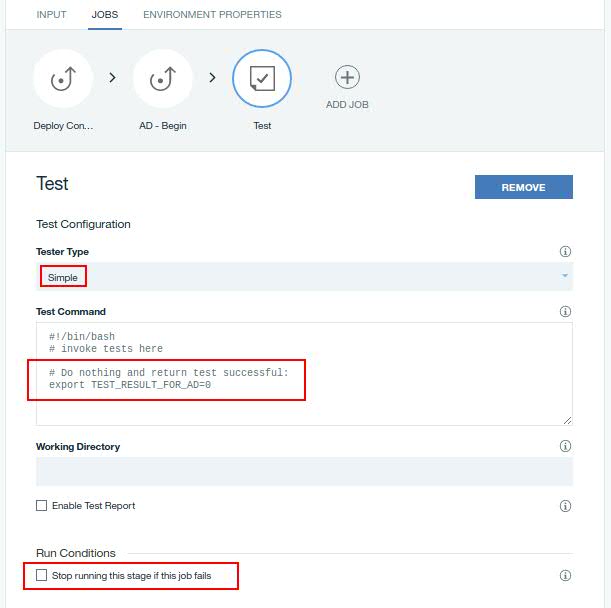
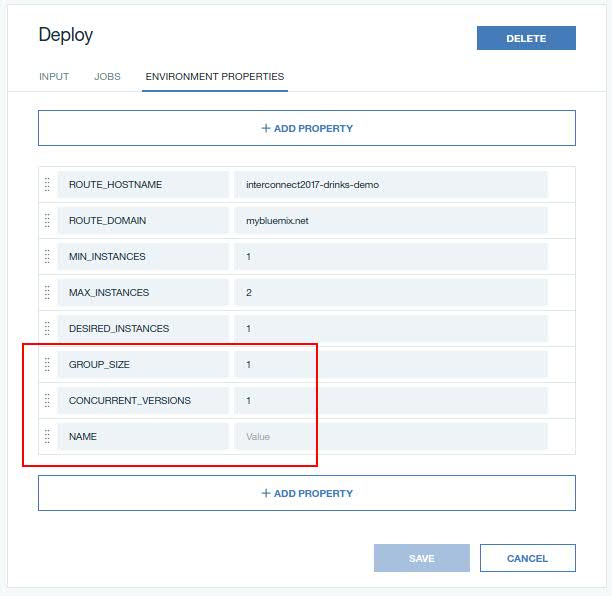
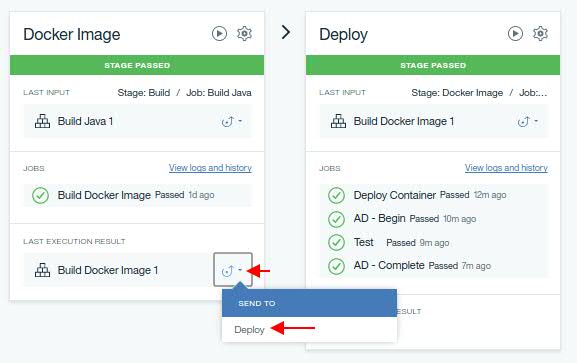
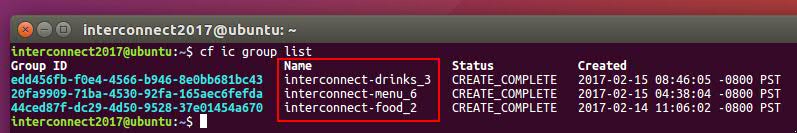
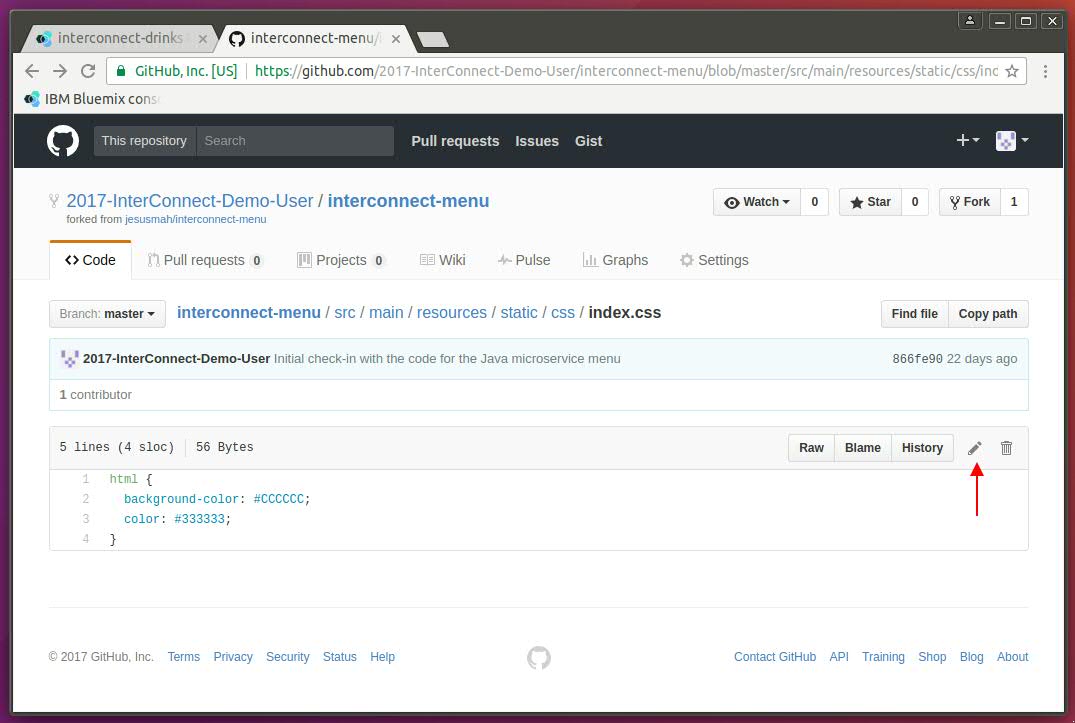
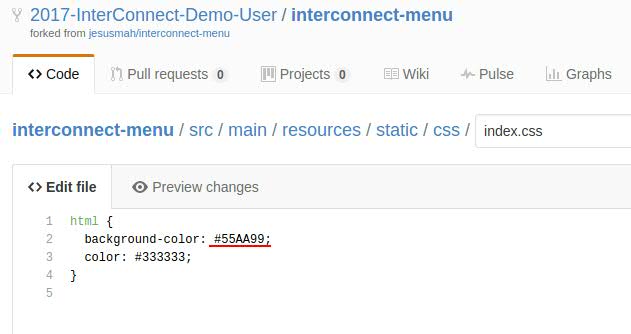
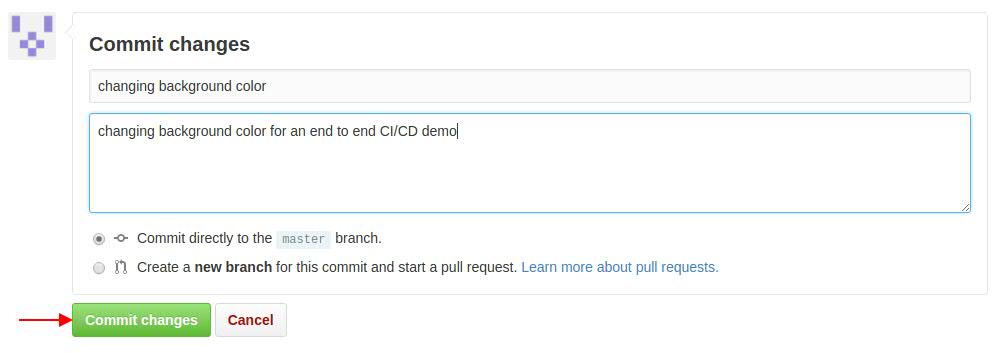
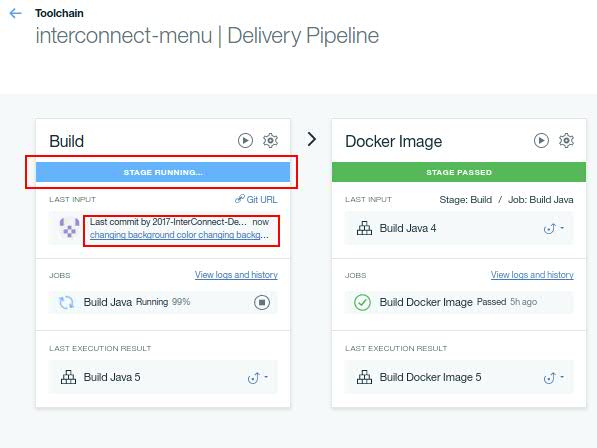
# IBM Bluemix Active Deploy Service

You can deploy a new version of your running apps or container groups with no downtime. [IBM Bluemix Active Deploy Service](https://console.ng.bluemix.net/docs/services/ActiveDeploy/about.html#about) 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. 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  
        
      
   2. Add the following line at the end of the *Deployer Script* text field  
        
      export NAME="${CONTAINER\_NAME}\_${BUILD\_NUMBER}"  
        
      
   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  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      The *Active Deploy – Begin* job will create a new container group and route traffic to it.
   4. Create a new Test job, name it appropriately and configure it as the picture below shows  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      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.
   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.
   6. Go into the environment tab and add the following ones  
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
        
      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.
   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.
   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. 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  
        
      
3. End to end CI/CD example
   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
   2. Click on the pencil to edit the file  
        
      
   3. Change the background color to something different  
        
      
   4. Scroll to the bottom of the page, include some comments about the change and commit the changes
   5. Right after you commit the changes, your delivery pipeline should automatically get triggered  
        
      
   6. Open your browser and point it to the public route your defined for the menu app
   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.
   8. Once the delivery pipeline has successfully finished, the background of your menu app should have changed on a disruptive manner  
        
        
      

# References

[Spring docs](https://spring.io/)

[Spring Boot docs](https://projects.spring.io/spring-boot/#quick-start)

[GitHub docs](https://help.github.com/)

[Docker docs](https://www.docker.com/what-docker)

[Dockerfile ref](https://docs.docker.com/engine/reference/builder/)

[IBM Bluemix](https://console.ng.bluemix.net/)

[IBM Bluemix Container Service docs](https://console.ng.bluemix.net/docs/containers/container_index.html)

[IBM Bluemix Active Deploy docs](https://console.ng.bluemix.net/docs/services/ActiveDeploy/index.html)

[DevOps and toolchains](https://www.ibm.com/devops/method/category/tools/)

[IBM Bluemix Delivery Pipeline docs](https://console.ng.bluemix.net/docs/services/DeliveryPipeline/index.html)

# External material

[IBM Cloud Architecture Center](https://developer.ibm.com/cloudarchitecture/)

[IBM Bluemix Garage Method](https://www.ibm.com/devops/method)

[IBM Microservices TV](https://developer.ibm.com/tv/microservices/)

[Meeting Microservices Blog](https://developer.ibm.com/cloudarchitecture/2016/10/20/meeting-microservices/)

[Microservices, meet DevOps Blog](https://developer.ibm.com/cloudarchitecture/2016/12/02/microservices-meet-devops/)