Quick Lab: Create a Full-Stack iOS App and Backend Microservices with Swift

Chris Bailey, IBM Runtimes, baileyc@uk.ibm.com

Create Fullstack iOS Apps with Swift

Introduction

This lab walks you through the steps required to take an existing iOS application and to extend it by adding a Mobile Backend that allows the app to store data in the cloud. The Mobile Backend is built using IBM's open source Kitura framework which uses Swift, the same programming language used on iOS. This makes is very easy to share code between the iOS application and the server, and to remove potentially interoperability problems.

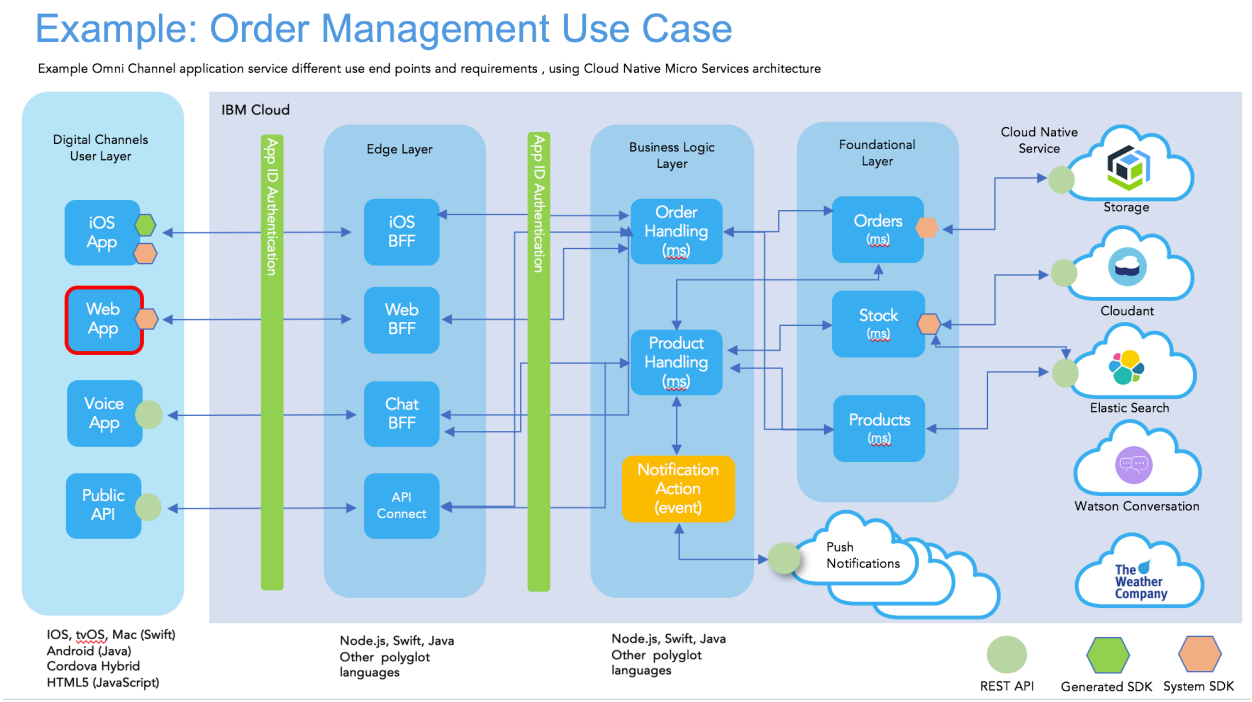
The iOS app is built for you using the FoodTracker sample app from Apple. You can build a Mobile Backend for it using either:

- A web browser using the **IBM Apple Developer Service**
  This offers a guided approach that enables you build cloud native apps in minutes. It eliminates the need to manually edit configuration files and allows you to easily add a select set of IBM Cloud Services to you project, provisioning those services as required. Moreover, you can easily attach a DevOps Toolchain to your project using the **IBM Cloud Continuous Delivery Service** that supports continuous integration and deployment to either a **Cloud Foundry** or **IBM Cloud Container Service** – a managed Kubernetes-based environment on the IBM Cloud. While the Cloud App Service is free, it simplifies the construction and deployment of cloud native apps by easily integrating with other paid IBM Cloud Services.

- The command line using the **Kitura CLI**
  This provides a local command line utility to create the Kitura cloud native application. This creates applications that are "Cloud Ready", including all the necessary configuration to create a DevOps pipeline to your project using Jenkins or the IBM Cloud Continuous Delivery Service, and for deployment to Cloud Foundry, Docker and Kubernetes technologies, such as those provided by IBM Cloud.

This lab allows you to use either approach. Note that using the **IBM Cloud App Service** requires an account with IBM Cloud. You may use your existing account or create a new account (Section 3)

For this lab, we are going to focus on an example cloud native web application architecture. We are going to build one part of this architecture: a iOS Backend For Frontend (BFF). For more information on a Reference Implementation for building a cloud native OmniChannel Application using a Microservices architecture refer to IBM Garage Method: https://github.com/ibm-cloud-architecture/refarch-cloudnative



At the conclusion of this lab, you will have completed the following steps:

1. Run the FoodTracker iOS application in a simulators
2. Create a Kitura Backend For Frontend (BFF) Application
3. Connected the iOS FoodTracker application to the Kitura backend
4. Retrieved data from the Kitura backend
5. Viewed the monitoring and metrics available for Kitura.

Setting Up

Before starting this lab, please do the following:

1. Go to http://ibm.biz/startmylab

2. Select **Create a Full-Stack iOS App and Backend Microservices with Swift** lab from the dropdown and click Ok
    a. You will be brought to the sign up page to register for an IBM Cloud Platform account. If you do not have an account, please register for one.
    b. This lab does not require you to have an IBM Cloud Platform account, but it allows you to do some optional steps.

3. Clone the project for this lab:
    a. Click on the Launchpad (rocket logo) in the toolbar, search for “Terminal” and click on the icon.
    b. Clone the GitHub project containing the FoodTracker application we are going to extend:

```
git clone http://github.com/seabaylea/FoodTrackerBackend
```

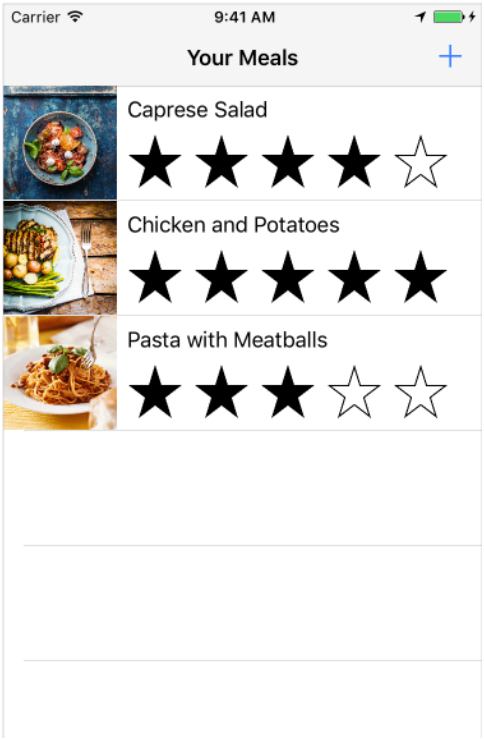
    c. A copy of these instructions are available included in the project. You can open the instructions using:

```
open “FoodTrackerBackend/ Create a Fullstack iOS app and
   Backend Microservice with Swift.pdf”
```

1. Run the FoodTracker iOS App Project

In this section, you will run the FoodTracker iOS app project, which provides you with the iOS application that we are going to extend.

The FoodTracker application is designed to allow you to create a record of Meals, with the ability to assign a name, photo and rating out of five for each of your the meals.



The meals are stored on the mobile device itself, which means that they cannot be shared with other users, and will be lost if the application was reinstalled or the device was lost.

The following steps allow you to setup and run the FoodTracker iOS application:

1. Open a Terminal window to access the command line:
   Click on the Launchpad (rocket logo) in the toolbar, search for “Terminal” and click on the icon.
2. Change into the iOS app directory:
   ```
   cd ~/FoodTrackerBackend/iOS/FoodTracker
   ```
3. Open the Xcode Project Workspace:
   ```
   open FoodTracker.xcworkspace
   ```

4. Open the Console using View > Debug Area > Activate Console
5. Run the project by pressing the Run button or use the ⌘+R key shortcut.
   If prompted to enable developer tools, select “Enable” and enter a password of “password”
   This may take a couple of minutes and will launch an iPhone simulator in which you can interact with the iOS application.
6. Add a meal in the Simulator by clicking the '+' button, providing a name, selecting a photo and a rating and clicking "Save".
7. Check that you receive a **“`Meals successfully saved.`”** message in the console
8. Stop the application by pressing the Stop button

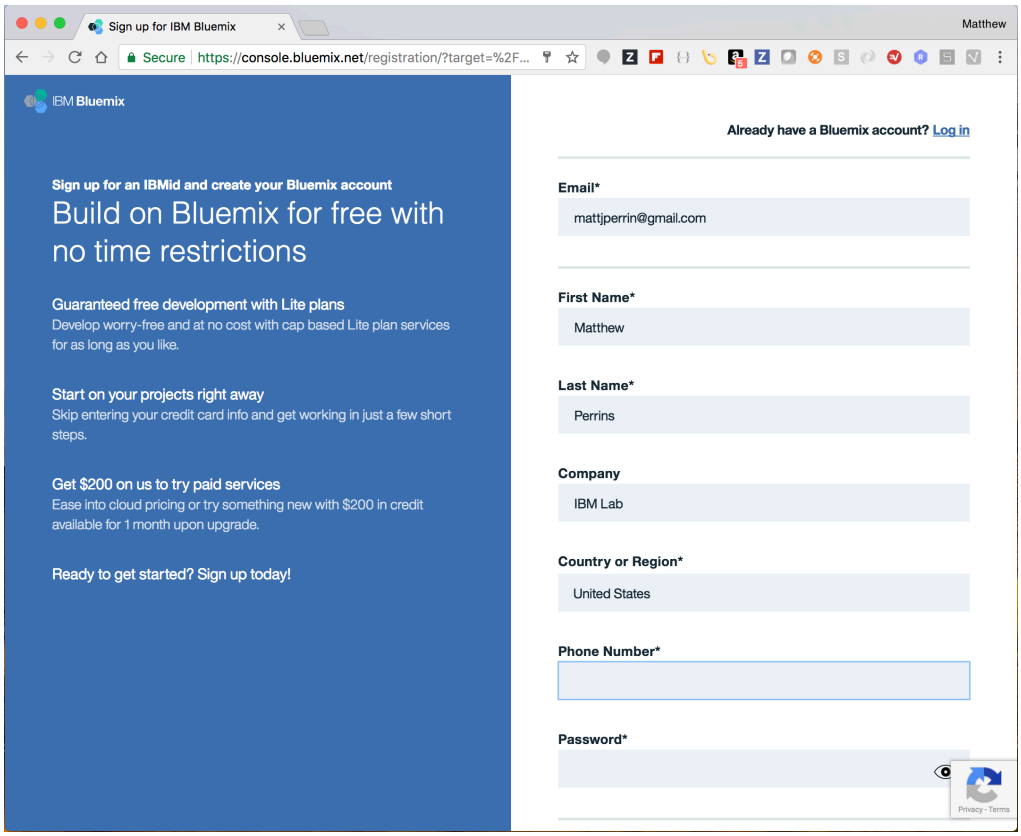
NEXT STEPS:
If you want to use the **IBM Cloud App Service**, which requires a IBM Cloud Platform ID, proceed to Section 2 (Create an Account) or Section 3 (Create a Kitura BFF on IBM Cloud)

If you want to use the **Kitura CLI**, which requires a IBM Cloud Platform ID, proceed to section 4 (Create a Kitura BFF)

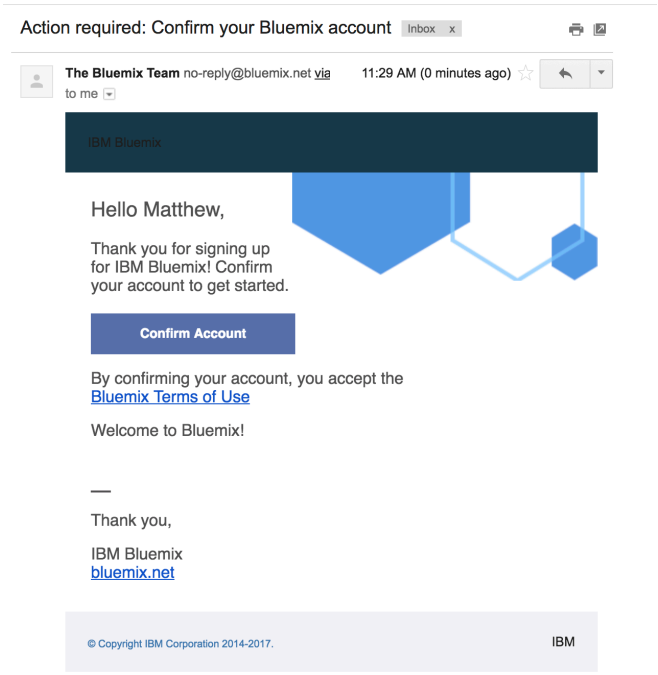
3. IBM Cloud App Service: Create an Account

If you **do** have an existing IBM Cloud account, skip to the next section.

1. To run these lab instructions, you will need to create a trial account on the IBM Cloud. You can do this free of charge and all the instructions contained in this lab do not incur any additional costs.
2. Create an IBM Cloud Trial account by navigating to this link. https://console.bluemix.net, and click on **Create a free Account.**
3. Complete the Registration Form with your details and a valid email address.

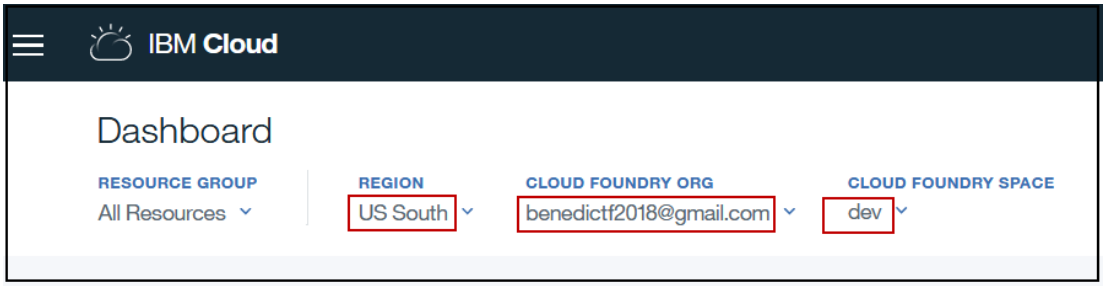


4. Click on **Create Account.**
5. Confirm your registration by accessing your email and clicking on **Confirm Account.**



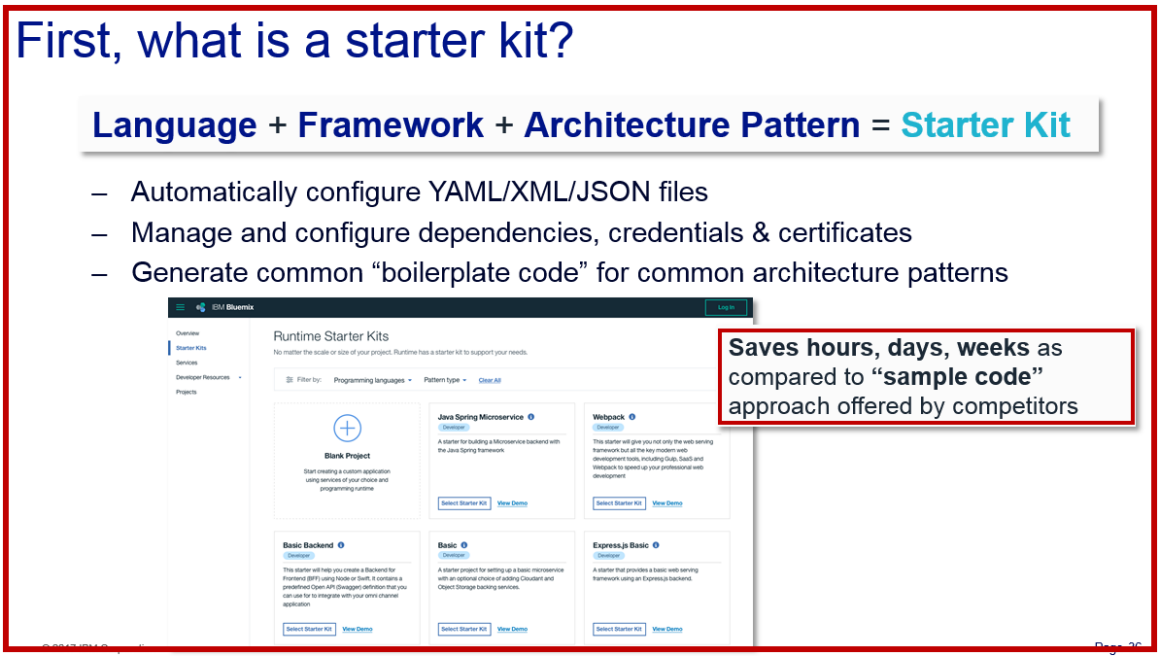
6. You can now log into IBM Cloud.
7. Click on **Login** and enter your email and password
8. You will finally see the Dashboard View, which will be empty as you have not created any services or apps at this point. Please note the **Region**, **Org** and **Space** settings.

   Typically this will be **US South**, the Org setting will be same as your email address and by default you are placed in a Space called **dev**, as shown below.



4. IBM Cloud App Service: Create a Kitura BFF on IBM Cloud

In this section, you will use the IBM Cloud to create a Kitura microservice application using the IBM Apple Developer Service from IBM.



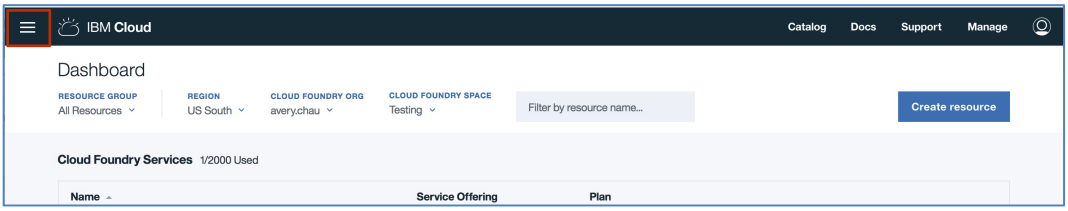
The IBM Apple Developer Service jump-starts cloud native development by allowing a developer to create a project, pick an application starter kit and deploy a production-ready application within minutes to the IBM Cloud.

The platform's code generation technology creates a starter application in the developer's preferred language and framework, which can be tailored to their needs and use case. Any services that are required in support of the use case, are provisioned automatically.

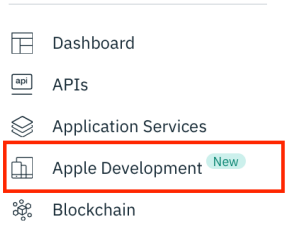
Developers can debug and test on their local workstation or in the cloud, and then use the DevOps Toolchain to collaborate with others to automate the delivery process.

1. Start by logging in to IBM Cloud at https://console.bluemix.net/

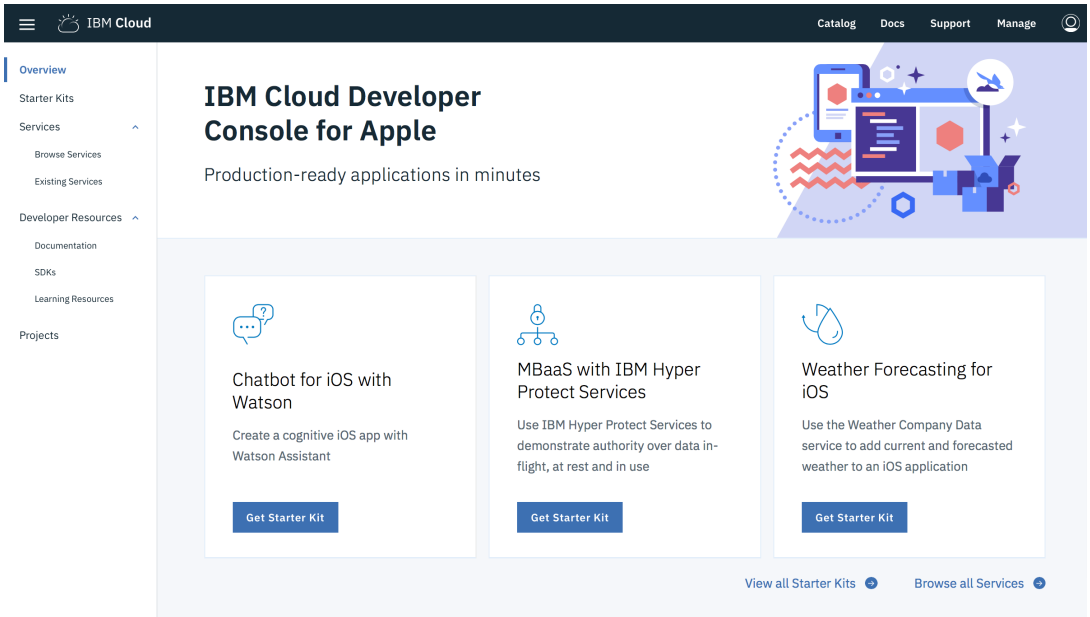
While logged into the IBM Cloud, click on the **Navigation** menu in the header:



Then click on **Apple Development** to access the **IBM Apple Developer Service** cloud-native development experience on IBM Cloud.

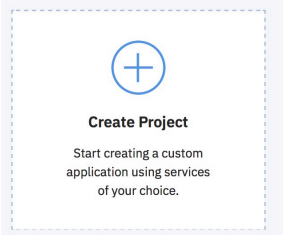


2. Take a few moments to explore the main Overview screen for the IBM Cloud App Service. From here you may access and discover hand-picked content related to cloud-native development including the **Resources**, **Community**, and **Starter Kits** sections. As explained IBM Cloud provides two approaches to building cloud native apps – a web-based and a CLI-based approach. This Lab will use the IBM Cloud App Service, which provides a web console.



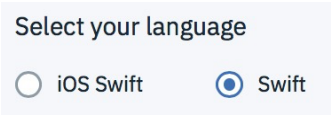
3. Navigate to the **Starter Kits** menu on the left-hand side. You will see a range of starter kits that are designed to provide production code starting point for a variety of common cloud-native development use cases. They include patterns and technology framework stacks for key programming languages that are commonly used by developers. Have a look through the list of starter kits for other types of projects you may have in the future.

4. Scroll down the list and click on the tile labelled **Create Project** to select the Starter Kit.

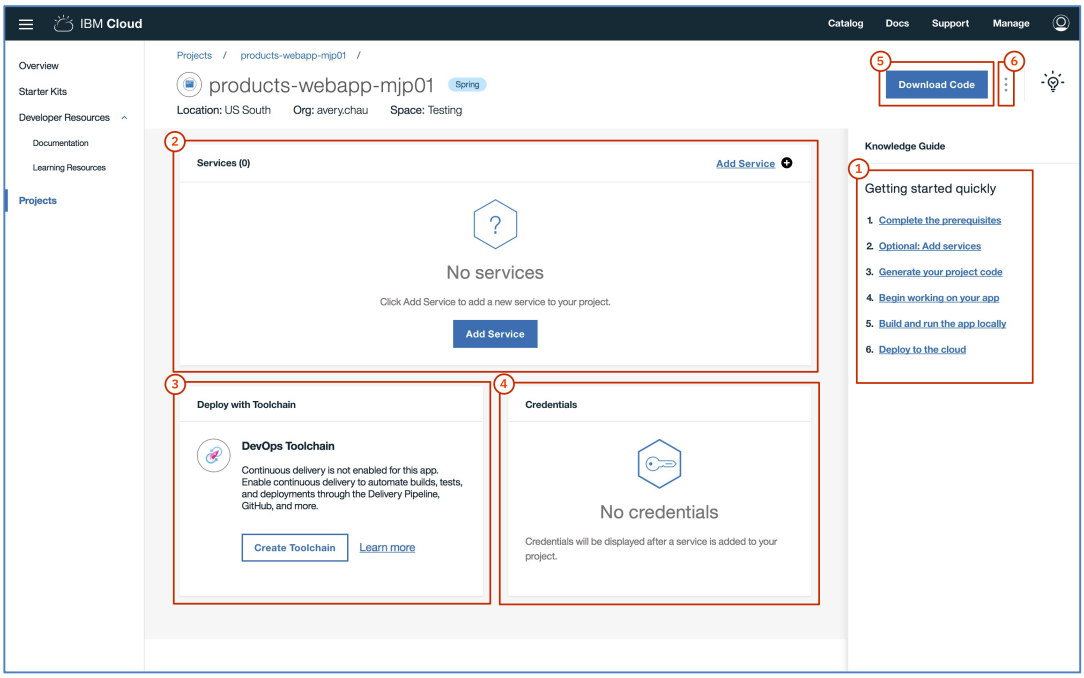


5. Within the Create New Project view, enter a project name of "**FoodServer**". The Kitura microservice will require a unique HTTP host name when it's running on the platform so ensure that the host name has a unique extension, eg "FoodServer-CNB". Once you've completed the form, click on the **Create Project** button.

6. Select a language of Swift, which will cause a server side Swift project to be created



7. Select "Create Project" to create your project.
8. The App Configuration/Details view is now displayed. This view is used to manage various aspects of an app:
    - The right-hand side (1) offers a Getting Started Quickly guide.
    - The main body has an area where you can create new and associate existing services to the app (2)
    - The bottom of the screen offers an area where you may configure a DevOps toolchain (3)
    - The bottom of the screen also offers an area where you may access credentials to any services that you have added to this app (4)
    - At the top of the screen, it is possible to download the code (5) to the app without any DevOps configuration.
    - Using the menu in the top right (6), you can also rename of the project or delete it.



9. Click “Download Code” to download the project to your local machine.

10. Open a Terminal window to access the command line:
    Click on the Launchpad (rocket logo) in the toolbar, search for “Terminal” and click on the icon.

11. Go to the project directory and copy the contents into the local project:

```
cd ~/Downloads
cd <project name>
mkdir ~/FoodTrackerBackend/FoodServer
cp –r * ~/FoodTrackerBackend/FoodServer/
```

Note: you may need to unzip your downloaded project. You can do that using:

```
mkdir FoodServer
unzip –d FoodServer FoodServer.zip
```

If you explore the generated assets you will see that there a number of configuration files generated for you, including:

- Jenkinsfile — A pipeline for Jenkins and IBM Cloud Private
- .bluemix/* — A pipeline for IBM Cloud Continuous Delivery Service

- Dockerfile — A dockerfile to build a Docker image for the application
- chart/* — A helm chart to enable deployment to Kubernetes
- manifest.yml — A configuration file for use with Cloud Foundry

Now proceed to Section 6 (Add a REST API to the Kitura BFF)

5. Kitura CLI: Create a Kitura BFF

In this section, you will use Kitura CLI to create an Kitura BFF application that is “Cloud Ready” and can be run locally or used to any cloud that supports any of CloudFoundry, Docker and Kubernetes, including IBM Cloud.

This includes installing the Kitura CLI and using it to create an Kitura BFF.

1. Install the Kitura CLI:
   From the command line run:

   ```
   brew tap ibm-swift/kitura
   brew install kitura
   ```

   This installs the Kitura CLI using the “homebrew” installation utility for macOS.

2. Create and enter a directory with the name you want to use for your project
   The Kitura CLI can ask you to provide a project name, but defaults to the current directory name. You can use any directory name, but its best to avoid using spaces or special characters:

   ```
   mkdir ~/FoodTrackerBackend/FoodServer
   cd ~/FoodTrackerBackend/FoodServer
   ```

3. Create your Kitura BFF:
   Run the following command on the command line to create a basic skeleton Kitura project:

   ```
   kitura init
   ```

   This creates a fully working Kitura project that provides monitoring and metrics which can then be extended with your application logic.

If you explore the generated assets you will see that there a number of configuration files generated for you, including:

- Jenkinsfile — A pipeline for Jenkins and IBM Cloud Private
- .bluemix/* — A pipeline for IBM Cloud Continuous Delivery Service

- Dockerfile — A dockerfile to build a Docker image for the application
- chart/* — A helm chart to enable deployment to Kubernetes
- manifest.yml — A configuration file for use with Cloud Foundry

Now proceed to Section 6 (Add a REST API to the Kitura BFF)

6. Add a REST API to the Kitura BFF

In this section, you will add two REST APIs to the Kitura BFF that can be used by the iOS application to store and retrieve data from the server. These REST APIs will be built using the definition of the Meal objects taken from the iOS app, ensuring that there are no interoperability problems.

1. Copy the Meal.swift file from the FoodTracker app to the Server
   One of the advantages of fullstack Swift is being able so share code. Here's we're going to share the Meal definition between the client and the server.

   ```
   cd ~/FoodTrackerBackend
   cp ./iOS/FoodTracker/FoodTracker/Meal.swift ./FoodServer/Sources/Application
   ```

12. Generate an Xcode project for the Kitura BFF and open it

    ```
    cd ~/FoodTrackerBackend/FoodServer
    swift package generate-xcodeproj
    open FoodServer.xcodeproj
    ```

13. Add REST APIs to allow the iOS application to store and load Meals:
    Add a POST and GET REST APIs to allow the saving and loading of Meals:

    Open the `Sources > Application > Application.swift` file.
    Either copy the contents of the Application.swift from the Git project you cloned (recommended):

    ```
    open -a TextEdit /Users/user/FoodTrackerBackend/Application.swift
    ```

    and cut and paste the contents into the Application.swift file in your project

    Or, make updates to the Application.swift file manually:

    a. Add a simple in-memory data store using an array to store the Meals:
       Add the following code on the line below `let cloudEnv = CloudEnv():`

       ```
       private var mealStore: [String: Meal] = [:]
       ```

       This now provides a simple dictionary to store Meal data passed to the FoodServer from the FoodTracker app. In a real scenario this would be replaced with a database.

    a. Create a POST REST API to allow FoodTracker to store Meals
       A request to store data typically consists of a POST request with the data to be stored, which the server then handles and responds with a copy of the data that has just been stored.

Use the following to register a handler for a POST request on /meals that stores the data by adding the following into the `postInit()` function in the App class:

```
router.post("/meals", handler: storeHandler)
```

This causes the `storeHandler()` function to be called when a matching request is received.

Implement the `storeHandler()` that receives a Meal, and returns the stored Meal by adding the following code as a function in the App class, beneath the `postInit()` function:

```
func storeHandler(meal: Meal, completion: (Meal?, RequestError?) -> Void
) {
    mealStore[meal.name] = meal
    completion(mealStore[meal.name], nil)
}
```

b. Create a REST API to allow FoodTracker to store Meals
A request to load all of the stored data typically consists of a GET request with no data, which the server then handles and responds with an array of the data that has just been stored.

Use the following to register a handler for a GET request on /meals that loads the data
by adding the following into the `postInit()` function in the App class:

```
router.get("/meals", handler: loadHandler)
```

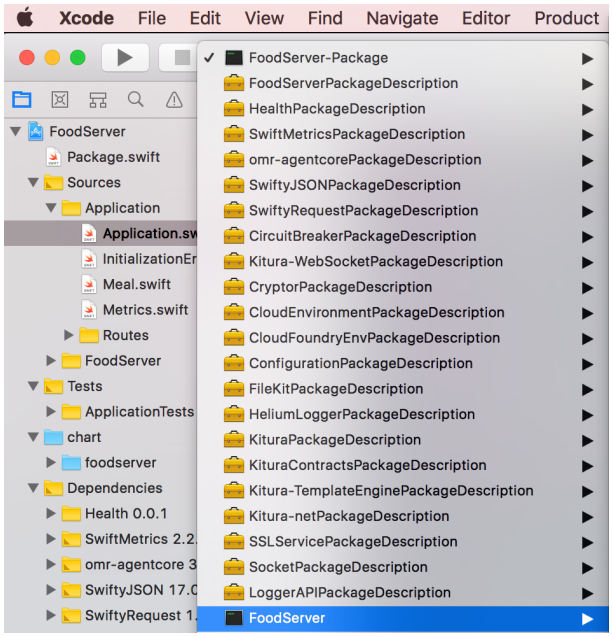
This causes the `loadHandler()` function to be called when a matching request is received.

Implement the `loadHandler()` that returns the stored Meals by adding the following code as a function in the App class, beneath the `postInit()` function:

```
func loadHandler(completion: ([Meal]?, RequestError?) -> Void ) {
      let meals: [Meal] = self.mealStore.map({ $0.value })
    completion(meals, nil)
}
```

The Application.swift file should now match the one provided in `/Users/user/FoodTrackerBackend/Application.swift`

14. Run the Kitura BFF
Edit the scheme by clicking on the "FoodServer-Package" section on the top-left the toolbar and selecting "FoodServer" (or the name of your project from the IBM Apple Developer Console) from the pulldown menu.



Press the Run button or use the ⌘+R key shortcut and select "Allow incoming network connections" if you are prompted.

15. Check that the Kitura server is running:
    Kitura Home: http://localhost:8080/
    Kitura Monitoring: http://localhost:8080/swiftmetrics-dash

16. Test the GET REST API is running correctly

    When browsers load web pages, they do so by making a GET request. That means that the easiest way of testing the GET REST API is by using a browser to request the following address:
    GET /meals endpoint: http://localhost:8080/meals

    This should return an empty array: '[]' because we aren't yet storing any data.

[ ]

7. Connecting the iOS App to the Kitura BFF

In this section, you will update the FoodTracker iOS application to connect to the Kitura BFF in order to save the Meals stored on the device. In order to simplify the connection and sending of data, Kitura provides the KituraKit client SDK. This has already been installed into the FoodTracker application for you, so you just need to uncommend the code that makes the connection.

1. Open the Xcode workspace

```
cd ~/FoodTrackerBackend/iOS/FoodTracker/
open FoodTracker.xcworkspace
```

17. Edit the `FoodTracker > MealTableViewController.swift` file:
Uncomment the sections of code marked with “UNCOMMENT TO USE SWIFT SERVER”:
Uncomment the import of KituraKit at the top of the file.

```
import KituraKit
```

Uncomment the following at the start of the saveMeals() function:

```
        for meal in meals {
              saveToServer(meal: meal)
        }
```

Uncomment the following saveToServer(meal:) function towards the end of the file:

```
private func saveToServer(meal: Meal) {
    guard let client = KituraKit(baseURL: "http://localhost:8080") else {
        print("Error creating KituraKit client")
        return
    }
    client.post("/meals", data: meal) { (meal: Meal?, error: Error?) in
        guard error == nil else {
            print("Error saving meal to Kitura: \(error!)")
            return
        }
        print("Saving meal to Kitura succeeded")
    }
}
```

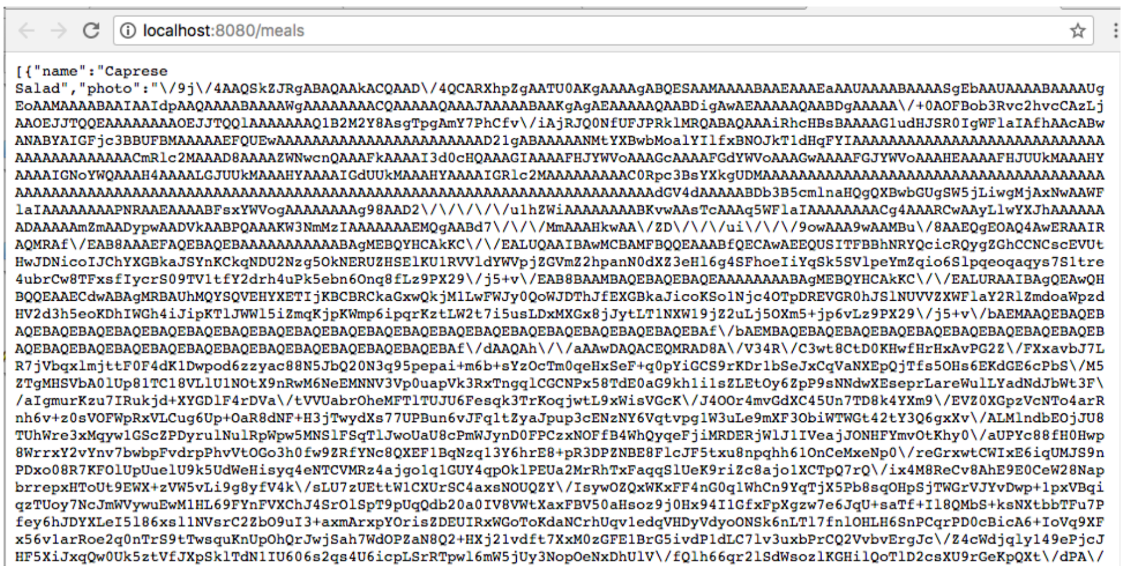
18. Run the FoodTracker app, storing data to the Kitura server

Build and run the FoodTracker app in the iOS simulator by pressing the Run button or use the ⌘+R key shortcut and the add or remove a Meal entry in the app.
You should see the following messages in the Xcode console:

```
Saving meal to Kitura succeeded
Saving meal to Kitura succeeded
```

19. Check the data has been persisted by the Kitura server
    Use your browser to make a GET request on /meals:
    GET /meals endpoint: http://localhost:8080/meals

    This should return a large amount of data. Most of what is being shown is the stored images, which are converted to a Base64 encoded string in order to be transmitted as part of a JSON object over the REST API.



Congratulations, you have successfully build a Kitura Backend for an iOS app!

This is part of a much larger tutorial for creating end-to-end Swift applications. You can find that from the tutorials section of http://kitura.io

8. Conclusion

Congratulations! You have completed the lab.

You have successfully built and run an end to end Swift application including and iOS application, and a Kitura server that is cloud-ready and can be deployed to any cloud supporting Cloud Foundry, Docker or Kubernetes , including IBM Cloud and IBM Cloud Private!

9. Resources

- IBM Cloud Developer Console for Apple: Build iOS and end-to-end Swift applications that use IBM Cloud services - https://console.bluemix.net/developer/appledevelopment/dashboard
- IBM Cloud App Service: http://bluemix.net/developer/appservice
- IBM Cloud Container Service: https://www.ibm.com/cloud/container-service
- IBM Cloud Garage Method: https://www.ibm.com/cloud/garage/
- IBM Cloud Continuous Delivery Service: https://www.ibm.com/cloud/continuous-delivery
- IBM Cloud DevOps: https://bluemix.net/devops
- IBM Cloud Developer Tools: https://www.youtube.com/watch?v=z-ByHuI41dU&t=76s
- IBM DevOps Overview: https://www.youtube.com/watch?v=psiMSUfn9oA
- Develop a Kubernetes App Toolchain: https://www.ibm.com/cloud/garage/tutorials/use-develop-kubernetes-app-toolchain?task=2
- IBM Cloud App Service: https://www.youtube.com/watch?v=VBZTwvLkGIk
- IBM Cloud Developer Tools (IDT): https://console.bluemix.net/docs/cloudnative/idt/index.html
- Using the IDT: https://www.youtube.com/watch?v=z-ByHuI41dU&t=76s
- Jump Start Your Microservice Development: https://www.youtube.com/watch?v=1HdtILoL6O4
- Switft@IBM Developer Center: http://developer.ibm.com/swift
- Kitura Homepage: http://kitura.io

Acknowledgements and Disclaimers (v8)